Encoding Preferences for Representation Learning in Computer Vision Tasks

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

M.Sc. Steffen Jung
aus Viernheim

Mannheim, 2025

Dekan: Prof. Dr. Claus Hertling, Universität Mannheim
Referent: Prof. Dr.-Ing. Margret Keuper, Universität Mannheim
Korreferent: Prof. Dr. Michael Möller, Universität Siegen
Korreferent: Prof. Dr. Bernt Schiele, Universität des Saarlandes

Tag der mündlichen Prüfung: 18. Dezember 2025

Erklärung zum Einsatz von Generativen Textmodellen

In der Erstellung dieser Arbeit wurde das generative Textmodell ChatGPT eingesetzt, um die schriftliche Präsentation dieser Thesis zu verbessern. In diesem Zusammenhang wurden einzelne und bereits formulierte Sätze und Textpassagen sprachlich und grammatikalisch überarbeitet, umformuliert, strukturiert und/oder von diesen Modellen zusammengefasst. Die erstellten Texte wurden zudem manuell geprüft und häufig weiter überarbeitet. Die Modelle wurden nicht dazu eingesetzt, neue Inhalte zu generieren. Insbesondere wurden alle in dieser Thesis eingeführten Methoden, Experimente und Resultate eigenständig – beziehungsweise mit oder von den jeweils gekennzeichneten Autoren – erarbeitet.

Abstract

Machine learning is the backbone of modern computer vision systems. The increasing availability of computational power, large-scale datasets, and advances in learning algorithms have enabled models to learn complex patterns directly from data, moving beyond traditional rule-based programming. A key factor in this success is representation learning, which transforms data into compact, meaningful representations for tasks such as image classification, segmentation, and synthesis. However, the No Free Lunch (NFL) theorems state that no universal algorithm can perform well across all tasks without incorporating task-specific knowledge. To build effective models, practitioners must introduce inductive biases that guide learning toward desirable solutions. Regularization is a central tool for this purpose, as it constrains the solution space and encodes preferences into the learning process. This thesis presents novel regularization techniques to guide representation learning in computer vision, focusing on three complementary strategies.

In the first part of this thesis, we encode discrete constraints in the representations learned by graph and image representation-learning models. These constraints, derived from the minimum cost multicut problem (a combinatorial problem partitioning graphs into subgraphs), are incorporated as penalty terms in the training loss to improve prediction consistency. For this, we first develop a method for training Graph Neural Network (GNN)-based solvers for the multicut problem itself. Then, we show that the same principle promotes closed contours in edge detection models.

In the second part of this thesis, we explore regularization strategies for generative models that match feature distributions between synthesized images and training images. First, we leverage pretrained image representations and demonstrate that using the widely adopted Fréchet Inception Distance (FID) as a regularizer reveals its limitations. Second, we show that images generated by Generative Adversarial Networks (GANs) lack fidelity in the spectral domain. Introducing an additional discriminator that extracts frequency-domain features improves generalization in this regard.

In the final part of this thesis, we study regularization by judging the importance of training data instances while training a model. This weighting induces biases that align model behavior with specific preferences. We apply this strategy to the discrete image synthesis model Vector Quantized Variational Autoencoder (VQ-VAE), enabling preference-driven control over generated images, such as enhancing facial attributes like smiling. We further develop a generative Neural Architecture Search (NAS) approach that balances accuracy with additional targets, such as efficiency and robustness, showing that generative architecture search can be aligned with multiple goals simultaneously.

Zusammenfassung

Maschinelles Lernen ist das Rückgrat moderner Systeme des maschinellen Sehens. Durch gestiegene Rechenleistung, große Datensätze und Fortschritten bei Lernalgorithmen können Modelle komplexe Muster direkt aus Daten extrahieren. Ein Faktor für diesen Erfolg ist das Repräsentationslernen, das Daten in kompakte Darstellungen für Aufgaben wie Bildklassifikation, Segmentierung und Synthese transformiert. Die No-Free-Lunch-Theoreme besagen jedoch, dass kein universeller Algorithmus existieren kann, der ohne aufgabenspezifisches Wissen über alle Aufgaben hinweg gut funktioniert. Um effektive Modelle zu entwickeln, müssen Praktiker induktive Verzerrungen einbringen, die das Lernen in Richtung gewünschter Lösungen lenken. Regularisierung ist hierfür ein zentrales Werkzeug, da sie den Lösungsraum einschränkt und Präferenzen in den Lernprozess einbringt. Diese Arbeit stellt neuartige Regularisierungstechniken vor, die das Repräsentationslernen im Bereich des maschinellen Sehens gezielt steuern und dabei drei sich ergänzende Strategien verfolgen.

Im ersten Teil dieser Arbeit kodieren wir diskrete Nebenbedingungen in das Training von Graph- und Bildrepräsentationslernmodellen. Diese Bedingungen, die sich aus dem kombinatorischen Minimum-Cost-Multicut-Problem ableiten, werden als Strafterme in den Trainingsverlust integriert, um die Konsistenz der Vorhersagen zu verbessern. Hierzu entwickeln wir eine Methode zum Training von graphbasierten Netzwerken für das Multicut-Problem selbst. Anschließend zeigen wir, dass sich mit demselben Prinzip geschlossene Konturen in Kantendetektionsmodellen fördern lassen.

Im zweiten Teil untersuchen wir Regularisierungsstrategien für generative Modelle, die Merkmalverteilungen zwischen synthetisierten und Trainingsbildern abgleichen. Zunächst nutzen wir vortrainierte Bildrepräsentationen und zeigen, dass die weit verbreitete Fréchet-Inception-Distanz als Regularisierer verwendet werden kann und damit deren Schwächen als Metrik offenbart. Zweitens zeigen wir, dass von generativ adversären Netzen erzeugte Bilder eine geringe Treue im Frequenzbereich aufweisen. Die Einführung eines zusätzlichen Diskriminators, der Merkmale im Frequenzbereich extrahiert, verbessert in dieser Hinsicht die Generalisierung.

Im letzten Teil befassen wir uns mit Regularisierung durch die Bewertung der Relevanz einzelner Trainingsdateninstanzen. Diese Gewichtung induziert eine Verzerrung, die das Modellverhalten gezielt an spezifische Präferenzen anpasst. Wir wenden diese Strategie auf ein diskretes Bildsynthesemodell an und ermöglichen so eine präferenzgesteuerte Kontrolle der generierten Bilder. Darüber hinaus entwickeln wir einen generativen Ansatz zur Suche von neuronalen Netzen, der Klassifikationsgenauigkeit mit weiteren Zielen wie Effizienz und Robustheit ausbalanciert und zeigen, dass sich Architektursuche mit mehreren Optimierungszielen gleichzeitig vereinbaren lässt.

Acknowledgements

First and foremost, I would like to express my sincere gratitude to everyone who supported me throughout my PhD journey. Thank you for the interesting talks, fruitful discussions, exciting conferences, inspiring retreats, and meaningful connections we shared during this time.

Most importantly, I am deeply grateful to my supervisor, Margret Keuper, for her guidance, encouragement, and trust over the years. She not only gave me the opportunity to pursue a PhD, but also the freedom to venture into different directions that ultimately culminated in this dissertation. Her support and feedback profoundly shaped both this work and my development as a researcher. In the same spirit, I want to thank Bernt Schiele for giving me the opportunity to be part of the outstanding research environment at the Max Planck Institute for Informatics, and for the support that enabled my research throughout these years.

I also sincerely thank my examination committee (Margret Keuper, Michael Möller, Bernt Schiele, and Simone Ponzetto) for their time, careful evaluation, and constructive feedback.

I am grateful to all the colleagues and friends I met during my time at the Max Planck Institute for Informatics, the University of Siegen, and the Data and Web Science group at the University of Mannheim. In particular, I would like to thank my teammates and collaborators (in alphabetical order): Shashank Agnihotri, Mishal Fatima, Julia Grabinski, Max Kahl, Jovita Lukasik, Tejaswini Medi, and Katharina Prasse. You contributed to making my day-to-day work both enjoyable and intellectually stimulating. Thank you for the shared ideas and teamwork, for accompanying me on my drives to Saarbrücken and Siegen, and for being there during those busy periods leading up to deadlines.

I am also grateful to everyone who helped improve this dissertation through careful proof-reading and thoughtful comments. In particular, I thank Julie Naegelen, along with Shashank, Tejaswini, Katharina, and Jovita, for their detailed feedback, which significantly strengthened the final manuscript.

Finally, I want to thank my family and friends for their continued support and encouragement through all the ups and downs I experienced during my PhD. In particular, I thank Navadha, whose endless support carried me through the finish line and beyond. Words cannot do justice to the gratitude I feel; thank you for being in my life.

Contents

List of Figures

List of Tables

List of Acronyms

AG-Net Architecture Generative Network

AP Average Precision

APGD Adaptive Projected Gradient Descent

BCE Binary Cross Entropy

BDCN Bi-Directional Cascade Network

BN Batch Normalization

BO Bayesian Optimization

CCL Cycle Consistency Loss

CE Cross Entropy

CNN Convolutional Neural Network

COB Convolutional Oriented Boundaries

CRF Conditional Random Field

CS Cloaking Score

DAG Directed Acyclic Graph

DCGAN Deep Convolutional Generative Adversarial Network

DNN Deep Neural Network

EI Expected Improvement

FCN Fully Convolutional Neural Network

FFHQ Flickr-Faces-HQ

FGSM Fast Gradient Sign Method

FID Fréchet Inception Distance

GAEC Greedy Additive Edge Contraction

GAN Generative Adversarial Network

GCN Graph Convolutional Network

GIN Graph Isomorphic Network

GMMN Generative Moment Matching Network

GNN Graph Neural Network

GTN Graph Transformer Network

HED Holistically-Nested Edge Detection

ICC Intervening Contour Cue

ILP Integer Linear Program

IS Inception Score

KLD Kullback-Leibler Divergence

KLj Kernighan-Lin with Joins

LP Linear Program

LR Logistic Regression

LSGAN Least Squares Generative Adversarial Network

LSO Latent Space Optimization

MCG Multiscale Combinatorial Grouping

MIO Mixed-Integer Optimization

MLP Multilayer Perceptron

MMD Maximum Mean Discrepancy

MP Minimum Cost Multicut Problem

MPNN Message Passing Neural Network

MS Multiscale Version

NAS Neural Architecture Search

NFL No Free Lunch

NLR Number of Linear Regions

NTK Neural Tangent Kernel

ODS Optimal Dataset Scale

OIS Optimal Image Scale

PGD Projected Gradient Descent

PP Polynomial Program

RCF Richer Convolutional Features

RGGCN Residual Gated Graph Convolutional Network

RL Reinforcement Learning

RNN Recurrent Neural Network

SD Spectral Difference

SGCN Signed Graph Convolutional Network

SGD Stochastic Gradient Descent

SNGAN Spectral Normalization Generative Adversarial Network

SVGe Smooth Variational Graph embeddings

SVM Support Vector Machine

UCM Ultrametric Contour Map

VAE Variational Autoencoder

ViT Vision Transformer

VQ-VAE Vector Quantized Variational Autoencoder

WGAN Wasserstein Generative Adversarial Network

WGAN-GP Wasserstein Generative Adversarial Network with Gradient Penalty

XGB XGBoost

Mathematical Notations

Functions and Representation Learning.

$f(\cdot)$	A function f.
$f_\theta(\cdot)$	A parameterized function f with parameters θ.
θ	Set of parameters of a function / model.
$f \circ g(\cdot)$	Composition of functions f and g.
$f * g(\cdot)$	Convolution of functions f and g.
$\mathcal{F}(\cdot)$	Fourier transform.
$\hat{\sigma}(\cdot)$	Non-linear activation function.
$\sigma(\cdot)$	Sigmoid activation function.
$\text{ReLU}(\cdot)$	Rectifier activation function.
$\mathbb{1}(\cdot)$	Indicator function.
$p(\cdot), q(\cdot)$	Probability distributions.
$\mathcal{N}(0, 1)$	Standard normal distribution.
$\mathbb{E}_p[\cdot]$	Expectation with respect to p.
$D_{\text{KL}}(p\|q)$	Kullback-Leibler divergence from p to q.
$\text{sg}[\cdot]$	Stop-gradient operator.
$\text{sign}(\cdot)$	Returns the sign of the input.
$\text{clip}_{\epsilon,\mathbf{x}}(\cdot)$	Clips input $\mathbf{x}$ in range $[\mathbf{x} - \epsilon, \mathbf{x} + \epsilon]$.

Numbers and Tensors.

$a \in \mathbb{R}$	A scalar.
$\mathbf{a} \in \mathbb{R}^{n(\times m \times \cdots)}$	A vector or tensor.
$\mathbf{A} \in \mathbb{R}^{n \times m(\times \cdots)}$	A matrix or tensor.
$\mathbf{A}_{[i,j,\cdots]} \in \mathbb{R}$	Scalar value at location $_{[i,j,\cdots]}$ of tensor $\mathbf{A} \in \mathbb{R}^{n \times m \times \cdots}$.
$\mathbf{I} \in \mathbb{R}^{n \times n}$	Square identity matrix.
$\mathbf{1} \in \mathbb{R}^n$	A vector of ones.
$\mathbf{0} \in \mathbb{R}^n$	A vector of zeros.
$\mathbf{a} \,\|\|\, \mathbf{b}$	Concatenation of two vectors.
$\mathbf{a} \odot \mathbf{b}$	Element-wise multiplication of two vectors.
$\|\cdot\|_p$	p-norm of a vector.

Continued on next page.

$tr(\cdot)$	Trace of a matrix.
$\lambda_i[\cdot]$	Returns the ith eigenvalue of a matrix.
$\mathcal{J}(\mathbf{x})$	Jacobian matrix with respect to input $\mathbf{x}$.
$\mathbf{H}(\mathbf{x})$	Hessian matrix with respect to input $\mathbf{x}$.

Data and Images.

p_{data}, $p_{\text{data}}(\mathbf{x})$	Data distribution a dataset is sampled from.
$\mathbf{x} \sim p_{\text{data}}$	Sample from p_{data}.
$\mathbf{x} \in \mathbb{R}^{H \times W (\times D)}$	Input image of height H, width W, and D channels.
$\hat{\mathbf{x}} \sim p_\theta$	Synthesized image from model p_θ.
$\tilde{\mathbf{x}} = \mathbf{x} + \epsilon$	Image perturbed with perturbation ϵ.
$\acute{\mathbf{x}} \in \mathbb{R}^{u \cdot W \times u \cdot H (\times D)}$	Image upsampled by some factor u.
$\mathcal{F}(\mathbf{x}) = \mathbf{x}^{\mathcal{F}} \in \mathbb{C}^{H \times W}$	Fourier-transformed image.
$y \in \mathbb{N}$	Image label.
$\mathcal{D}$	A dataset.
$\mathcal{D} = \{(\mathbf{x}_i)\}_{i=1}^N$	An unlabeled (image) dataset.
$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$	A labeled (image) dataset.

Model Training and Regularization.

$\text{CE}(\cdot, \cdot)$	Function computing cross-entropy loss.
$\mathcal{L}, \mathcal{L}(\cdot)$	The loss as a value or function.
$\Omega(\cdot)$	Function computing a penalty term.
$\Pi_{\mathcal{C}}(\mathbf{x})$	Projection operator, projecting a point $\mathbf{x}$ onto set $\mathcal{C}$.
$\mathcal{C} \subseteq \mathbb{R}^n$	Set of feasible values.

Continued on next page.

Graphs.

$G = (V, E)$	A graph. Throughout this thesis we mainly encounter undirected graphs and mention it otherwise.
V	Set of nodes of the graph.
$u \in V$	Node u of the graph.
$\mathcal{N}(u) \subseteq V$	Set containing all nodes connected to node u (neighborhood).
$\mathcal{N}^+(u) \subseteq \mathcal{N}(u)$	Set containing all nodes connected via positively weighted edges to node u.
$\mathcal{N}^-(u) \subseteq \mathcal{N}(u)$	Set containing all nodes connected via negatively weighted edges to node u.
E	Set of edges of a graph.
$E^- \subseteq E$	Subset of E with negative edge weights.
$E^+ \subseteq E$	Subset of E with positive edge weights.
$e = (u, v) \in E$	Some edge in E connecting nodes u and v.
$\deg(u)$	Degree of node u.
$\overline{\deg}(u)$	Signed degree of node u.
$\mathbf{A}$	Graph adjacency matrix.
$\mathbf{D}$	Graph degree matrix.
$\overline{\mathbf{D}}$	Signed graph degree matrix.
$\mathbf{L} = \mathbf{I} - \mathbf{D}^{-1/2}\mathbf{A}\mathbf{D}^{-1/2}$	Symmetric normalized graph Laplacian.

Minimum Cost Multicut Problem.

$\mathbf{y} \in \{0, 1\}^{\lvert E \rvert}$	Binary decision vector.
$\tilde{\mathbf{y}} \in \{0, 1\}^{\lvert E \rvert}$	Optimal decision vector.
$\mathbf{y}^+ \in \{0, 1\}^{\lvert E \rvert}$	Decision vector removing all positive edges.
$\mathbf{y}^- \in \{0, 1\}^{\lvert E \rvert}$	Decision vector removing all negative edges.
$\hat{\mathbf{y}} \in [0, 1]^{\lvert E \rvert}$	Relaxed decision vector.
$y_e,\ \tilde{y}_e,\ \hat{y}_e$	A corresponding decision variable of edge e.
$\mathbf{w} \in \mathbb{R}^{\lvert E \rvert}$	Vector containing weights assigned to each edge.
$w_{u,v} \in \mathbb{R},\ w_e \in \mathbb{R}$	Weight of edge $e = (u, v)$.
$c = c(\mathbf{y}) = \mathbf{w}^T\mathbf{y}$	Objective value c given by objective function $c(\cdot)$ and decision vector $\mathbf{y}$.
$c_r = c(\mathbf{y})/c(\tilde{\mathbf{y}})$	Ratio of the objective value to the optimal solution.
$\mathrm{cc}(G)$	Set of all chordless cycles in G.

Continued on next page.

$\text{cc}(G, l)$	Set of chordless cycles in G, where the length of each cycle is at most l.
$C \in \text{cc}(G)$	A chordless cycle C in G.

Graph Neural Networks.

$\mathbf{x}_u \in \mathbb{R}^*$	Node feature vector of some dimensionality.
$\mathbf{h}_u \in \mathbb{R}^*$	Node representation of node u.
$\mathbf{x}_{u,v} \in \mathbb{R}^*$	Edge feature vector of edge (u, v).
$\mathbf{h}_{u,v} \in \mathbb{R}^*$	Edge representation of edge (u, v).
f_e	Edge representation mapping function.
f_c	Edge classification function.
f_r	Mapping to set of feasible solutions.

Conditional Random Fields (CRFs).

$E(\mathbf{x} \mid \mathbf{y})$	Energy function defining the CRF.
ψ^U, ψ^{Cycle}	CRF potentials.

Chapter 1
Introduction

In an era where technology shapes nearly every aspect of our lives, machine learning emerged as a central driver of innovation. It is the major backbone of systems in computer vision that enables autonomous vehicles to navigate complex urban environments [Jan+20], supports medical professionals in the early diagnosis of diseases through advanced imaging analysis [Est+21], and assists conservationists in remotely monitoring and protecting endangered species [Tui+22]. Elevated by increased computational power, large datasets, and breakthroughs in learning algorithms, machine learning allows systems to learn patterns directly from data [HB20]. This marked a transition away from traditional rule-based programming [GBC16], whereas representation learning is one of the most important factors enabling this transition [BCV13].

Representation Learning. The aim of representation learning is to automatically discover expressive and useful representations [BCV13] of data to facilitate downstream tasks such as classification, regression, clustering, or even synthesis of new data. Instead of relying on handcrafted features, representation learning allows models to learn features directly from raw data, often revealing underlying structures and relationships [BCV13; LBH15; GBC16]. Consequently, representation learning is particularly effective in domains like computer vision, where raw data is high-dimensional and complex [TIF24]. The hope is that automatically extracted features generalize well across multiple tasks and datasets. By transferring learned features, this reduces the need for large labeled datasets or the training of feature extractors on new (niche) tasks [PY10].

No Free Lunch. At the same time, the No Free Lunch (NFL) theorems state that there cannot be a universal learning algorithm performing better than random guessing when averaged over all possible tasks [WM97; Wol02]. Wolpert and Macready [WM97] emphasize the »importance of incorporating problem-specific knowledge into the behavior of the algorithm«. Sterkenburg and Grünwald [SG21] argue that learning algorithms in the context of machine learning are model-dependent and require a model as input that represents a bias for the task at hand. Therefore, for a practitioner in machine learning, it is necessary to choose an algorithm by providing preferences over its bias. In the context of this thesis, *a (learning) algorithm* is a representation-learning neural network with its hyperparameters and training settings, such as training data, data sampling, and other related settings. The collection of all choices about the network and its training constitutes the bias we choose for our learning algorithm [SG21].

Inductive Bias. In machine learning, training a model can be understood as searching through a space of possible functions to find one that fits the training data [Mit97]. By building preferences about the function into the algorithm, we constrain this space, guiding the search toward functions that reflect our assumptions about the problem [SB14; GBC16]. This reflects the bias that arose from the NFL theorems and is called the *inductive bias* of a model. As defined by Mitchell [Mit80], the inductive bias is a set of inherent assumptions that a model has in order to infer solutions for inputs it has not seen before. This corresponds to the model having preferences for certain types of solutions over others.

Regularization. The restriction of the explorable function space (by expressing preferences over solutions) is what we refer to as regularization [GBC16]. By regularizing, we modify the inductive bias of a model. For models that learn representations from data, regularization can express preferences for the structure of learned representations, by favoring certain features over others. A straightforward way to regularize a model is to add an additional term to its training loss function [Bis06]. Other regularization techniques may affect different parts of the model training. For example, early stopping [Pre96] regularizes the optimization itself, while data augmentation affects the distribution of the training data. Some regularization effects arise implicitly and are part of the network architecture, as for example batch normalization layers [IS15], which are argued to smoothen the optimization landscape [LRP19]. All these types of regularization influence the inductive bias of a model, and therefore its inherent preference for certain types of solutions [Mit80].

Thesis Outline. In this thesis we demonstrate novel ways to regularize the training of representation learning models for selected computer vision tasks. During this journey, we expand our regularization strategy from regularization via manually designed penalty terms (Part I), to regularization by matching distributions of extracted features (Part II), and finally to regularization via data augmentation (Part III). We visualize key differences between these approaches in Figure 1.1. In particular, we divide this thesis into the following parts.

- (Part I): Here, we **penalize** graph representation-learning networks and image representation-learning networks by incorporating discrete constraints as penalty terms into their training loss function in order to express preferences over the configuration of their predictions.
- (Part II): In this part, we regularize image synthesizing models by **matching** distributions of extracted features between synthesized images and training data. By this means, we show that evaluating image synthesis with Fréchet Inception Distance (FID) is flawed, and express preferences over underexplored aspects of image quality.
- (Part III): In the last part of this thesis, we induce biases in generative models by augmenting and **judging** the importance of their training data. This approach allows us to impose a model bias through the assessment of training data instance importance.

1.1 Thesis Structure and Publications

The current chapter provides an introduction and outline to this thesis (see Figure 1.2 for a visualization). Then, in Chapter 2, we introduce fundamental concepts relevant for this thesis. These include representation learning, computer vision tasks addressed, as well as fundamental regularization concepts. The remainder of the thesis is divided into three parts that each represents a different regularization approach studied in the context of computer vision tasks, and that each introduces novel regularization approaches in their respective scope. Chapters in these parts are each based on a publication and include additional related work sections to introduce further concepts relevant to the respective topic. The parts are structured in the following way.

Part I: Penalize. In the first part of this thesis, we encode discrete constraints in the representations learned by graph representation-learning models and image representation-learning models. Both edge detection and image segmentation have a rich history of methods formulating

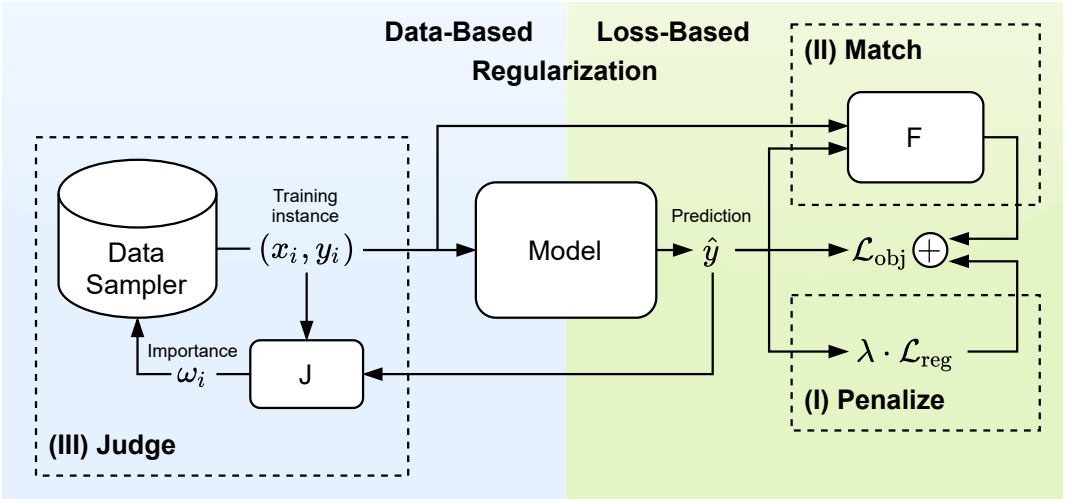


Figure 1.1: Visualization of the regularization strategies in this thesis. First, in Part I, we incorporate manually-designed penalties into the training of models. Second, in Part II, we regularize via a feature matching component (F) that provides a loss by distinguishing training and output feature distributions. Last, in Part III, we assign importance scores (ω) to training data instances via a score-assigning component (J), implicitly shaping the bias of the model.

these tasks as discrete optimization problem, and more specifically, as graph partitioning problem [SM00; KAB15; Keu17; And+11; And+13]. One of these formulations is the minimum cost multicut problem, which can be formulated as Integer Linear Program (ILP) [BBC04; Dem+06] and contains binary constraints defining feasible solutions thereof. In particular, in this part of the thesis, we incorporate these binary constraints as **penalty terms** in the loss during training of discriminative models. We develop a method incorporating these constraints in the training of a Graph Neural Network (GNN)-based [Gil+17] solver for the minimum cost multicut problem itself in Chapter 3. Then, in Chapter 4 we show that regularizing edge detection models with the same core concept improves the boundary consistency of their predictions. The following author publications contribute to this part.

[JK22] S. Jung and M. Keuper. "Learning to solve Minimum Cost Multicuts efficiently using Edge-Weighted Graph Convolutional Neural Networks". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2022

[Jun+22] S. Jung, S. Ziegler, A. Kardoost, and M. Keuper. "Optimizing Edge Detection for Image Segmentation with Multicut Penalties". In: *German Conference on Pattern Recognition (GCPR)*. 2022

Part II: Match. Generative Adversarial Networks (GANs) [Goo+14] are trained via a minimax game between a generator network and a discriminator network. The goal of the generator network is to produce images that are as close to the training data distribution as possible, attempting to fool the discriminator network. In this part we develop new approaches to regularize the training of generator networks by comparing extracted features from synthesized images and training images. In Chapter 5 we leverage pretrained image representations to

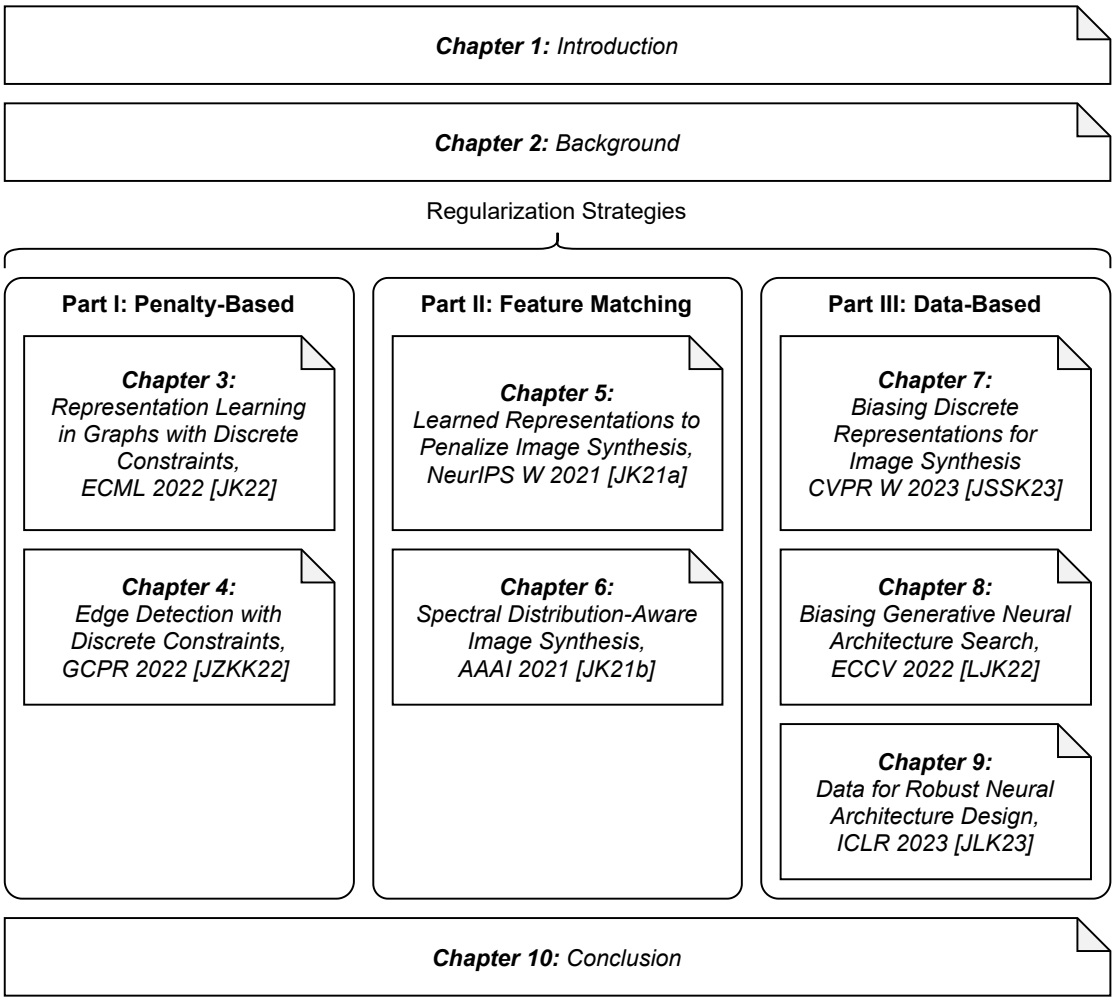


Figure 1.2: Visualization of the structure of this thesis. Part I addresses the topic of regularization by incorporating discrete constraints into the training of models that extract learned representations from graphs and images. Part II introduces approaches to regularize Generative Adversarial Networks (GANs) based on matching extracted features in the task of image synthesis. Part III investigates the potential of simultaneously augmenting and weighting training data as a form of regularization in the tasks of image synthesis and Neural Architecture Search (NAS).

guide the regularization process. We show that the popular (and still actively used) metric FID [Heu+17] measuring the quality of synthesized images is flawed by incorporating it as a regularizer when training generator networks. In Chapter 6 we show that GANs lack fidelity in the spectral domain of images and have to be regularized to generalize well under this aspect. For this, we introduce an additional discriminator network that extracts spectral features, encouraging the generator to match the spectral statistics of the data distribution. The following author publications contribute to this part.

[JK21a] S. Jung and M. Keuper. "Internalized Biases in Fréchet Inception Distance". In: *Advances in Neural Information Processing Systems (NeurIPS) Workshop on Distribution Shifts: Connecting Methods and Applications*. 2021

[JK21b] S. Jung and M. Keuper. "Spectral Distribution Aware Image Generation". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2021

Part III: Judge. In the last part of this thesis, we focus on regularizing models by augmenting their training data and judging the importance of each training instance towards specific preferences. This weighting of the training data [TDH20] induces a bias into the model that aligns its behavior with our preference. First, in Chapter 7 we enable this data-based regularization method in the discrete image synthesizing model Vector Quantized Variational Autoencoder (VQ-VAE) [OVK17]. On a case-study task, we investigate aligning the model with an increase in the degree of smiling when synthesizing images of faces. We show that this is an effective framework to optimize VQ-VAEs for arbitrary preferences. Second, in Chapter 8 we introduce a novel, generative Neural Architecture Search (NAS) approach that is able to optimize architecture design choices for the task of image classification while aligning it with multiple different target metrics. Exemplary, we find architectures that are performant on their original task (having high accuracy in image classification), while simultaneously are optimized for low latency on target hardware devices. Another desirable property of image classification models is robustness towards adversarial [GSS15] as well as common [HD19] perturbations. In this context, we introduce a robustness dataset in Chapter 9, enabling our NAS algorithm to find performant as well as robust architectures. The following author publications contribute to this part.

[LJK22] J. Lukasik*, S. Jung*, and M. Keuper. "Learning where to look–generative nas is surprisingly efficient". In: *European Conference on Computer Vision (ECCV)*. 2022

[JLK23] S. Jung*, J. Lukasik*, and M. Keuper. "Neural Architecture Design and Robustness: A Dataset". In: *International Conference on Learning Representations (ICLR)*. 2023

[Jun+23] S. Jung, J. C. Schwedhelm, C. Schillings, and M. Keuper. "Happy People–Image Synthesis as Black-Box Optimization Problem in the Discrete Latent Space of Deep Generative Models". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshop: Generative Models for Computer Vision*. 2023

[LJK22] and [JLK23] are equal contributions with Jovita Lukasik. For [LJK22] we jointly developed the idea and wrote the paper. Jovita provided the generative model and latent space optimization on single targets, and Steffen provided latent space optimization on multiple targets. For [JLK23] we jointly wrote the paper. Jovita provided experiments on training-free measurements and NAS results. Steffen developed the idea, implemented the dataset framework and evaluations, aggregated the dataset, and performed the initial analysis. The NAS results for the chapter in this thesis are revised by Steffen.

We provide outlines for each part on their respective part title page, and additionally short outlines of each publication-based chapter on the respective chapter opener. In the last chapter of this thesis, Chapter 10, we summarize our findings and provide an outlook from our work towards remaining challenges and future research directions.

1.2 Additional Publications

The author of this thesis is also an author of the following publications that are not substantially contributing to this thesis.

[Gra+22] J. Grabinski, S. Jung, J. Keuper, and M. Keuper. "FrequencyLowCut Pooling - Plug and Play Against Catastrophic Overfitting". In: *European Conference on Computer Vision (ECCV)*. 2022

[Pra+23b] K. Prasse, S. Jung, Y. Zhou, and M. Keuper. "Local Spherical Harmonics Improve Skeleton-Based Hand Action Recognition". In: *German Conference on Pattern Recognition (GCPR)*. 2023

[Pra+23a] K. Prasse, S. Jung, I. B. Bravo, S. Walter, and M. Keuper. "Towards Understanding Climate Change Perceptions: A Social Media Dataset". In: *Advances in Neural Information Processing Systems (NeurIPS) Workshops: Tackling Climate Change with Machine Learning*. 2023

[AJK24] S. Agnihotri†, S. Jung†, and M. Keuper. "CosPGD: an efficient white-box adversarial attack for pixel-wise prediction tasks". In: *International Conference on Machine Learning (ICML)*. 2024

[MJK25] T. Medi, S. Jung, and M. Keuper. "FAIR-TAT: Improving Model Fairness Using Targeted Adversarial Training". In: *Winter Conference on Applications of Computer Vision (WACV)*. 2025

[Gav+25] P. Gavrikov, J. Lukasik, S. Jung, R. Geirhos, B. Lamm, M. J. Mirza, M. Keuper, and J. Keuper. "Can We Talk Models Into Seeing the World Differently?" In: *International Conference on Learning Representations (ICLR)*. 2025

[FJK25] M. Fatima, S. Jung, and M. Keuper. "Corner Cases: How Size and Position of Objects Challenge ImageNet-Trained Models". In: *Transactions on Machine Learning Research (TMLR)* (2025)

Chapter 2
Background

CONSIDER the task of identifying whether an email is spam or not. In a traditional system, we might write rules like »if the subject contains *win money* or *free offer*, classify it as spam«. In contrast, a classic machine learning approach would involve manually extracting features from the email, such as the frequency of specific keywords (e.g., *win*, *offer*, *free*), the presence of suspicious links, or whether the email comes from an untrusted domain. These handcrafted features are then used to train a machine learning model, such as a logistic regression or decision tree, on a labeled dataset of emails marked as spam or not spam. The model learns to associate patterns in these features with the likelihood of an email being spam. In representation learning, however, the process is more automated and not reliant on manual feature engineering. For example, a neural network can be trained on the raw email text and metadata without predefined features. The model learns to represent the text as high-dimensional vectors, capturing patterns such as word relationships, contextual meanings, and stylistic differences between spam and legitimate emails. This approach reduces the need for human intervention in defining features and enables the system to adapt more quickly. With enough training data, a representation learning model can outperform traditional methods by identifying subtle and complex patterns that handcrafted rules or features might miss. Figure 2.1 sketches the described differences between those approaches.

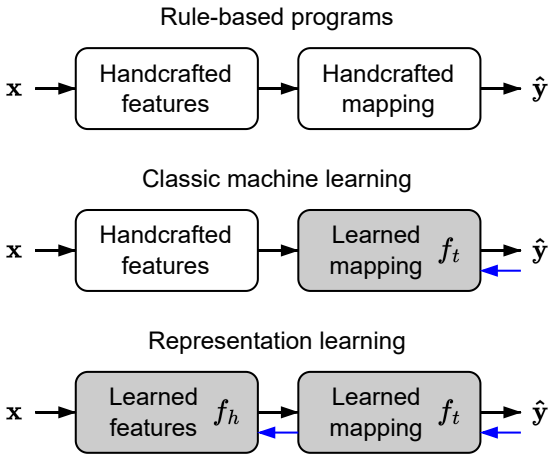


Figure 2.1: Possible pipelines that map from input data $\mathbf{x}$ to output $\hat{\mathbf{y}}$. Gray boxes depict components that automatically learn from data via training signals (blue lines), while white boxes are handcrafted components [GBC16; Sze22].

An important aspect of representation learning approaches is the choice of training signals. In this thesis, we introduce new approaches to providing such signals by regularizing representation learning models in computer vision tasks. The current chapter serves as introduction to relevant

topics in this context. We keep explanations brief and illustrative, providing more detail only when it is essential for understanding the framework of this thesis. Related work is discussed in more depth in the respective chapters. First, Section 2.1 introduces the concept of representation learning in general. Second, Section 2.2 transfers this concept to the field of computer vision and describes different tasks therein. Last, Section 2.3 motivates regularization in machine learning, and describes different ways to regularize models with a larger focus on computer vision.

2.1 Representation Learning in Machine Learning

Representation learning is one of the core concepts of modern machine learning. Its goal is to automatically discover meaningful representations from raw input data [BCV13]. In the context of this thesis, we define representation learning as a parametrized function $f_h(\mathbf{x})$ that, given some input $\mathbf{x}$, produces an n-dimensional feature vector $\mathbf{h} \in \mathbb{R}^n$:

$$f_h : \mathbf{x} \mapsto \mathbf{h} \in \mathbb{R}^n.$$

In practice, this learning of features mostly happens implicitly as part of a neural network that is trained on a certain downstream task. We can consider a neural network that is trained on a downstream task as a composition of two functions $f_t \circ f_h(\mathbf{x})$, where $f_h(\cdot)$ is the parametrized function learning to extract features from input data $\mathbf{x}$, and $f_t(\cdot)$ is a parametrized function that learns to map from the extracted features to the desired, task-dependent output $\hat{y}$:

$$\hat{y} = f(\mathbf{x}) = f_t\left(f_h\left(\mathbf{x}\right)\right). \tag{2.1}$$

Solving the XOR-Problem with Representation Learning. As a demonstrative example, the XOR (exclusive OR) problem [MP69] is a classic case in the study of neural networks that illustrates the difficulty of learning non-linearly separable data. For this problem, non-linear models like Multilayer Perceptrons (MLPs) (paired with non-linear activation functions) are required to learn non-linear decision boundaries [Cyb89]. In Figure 2.2, we depict an XOR-inspired dataset with decision boundaries learned by both a logistic regression [HLS13] and an MLP. The MLP trained here (depicted in Figure 2.3 (left)) has one hidden layer consisting of two neurons. We can assume the MLP here to be a composition as noted in Equation 2.1, whereas activations $\mathbf{h}$ after the hidden layer are the result of its implicit feature extraction: $\mathbf{h} = f_h(\mathbf{x}) = \hat{\sigma}(\mathbf{x}^\top \mathbf{w}_h)$, where $\hat{\sigma}(\cdot)$ is a non-linear activation function, and $\mathbf{w}_h$ is a vector of parameters of this layer. The last layer of this MLP then performs the classification task like a logistic regression: $\hat{y} = f_t(\mathbf{h}) = \sigma(\mathbf{h}^\top \mathbf{w}_t)$, where $\sigma(\cdot)$ is the sigmoid function and $\mathbf{w}_t$ again are parameters of this layer. When we plot the extracted features (see Figure 2.3 (right)), we can observe that the 2-dimensional, linearly non-separable input data is transformed into a space where the features now are linearly separable. Hence, the implicit feature extractor of this MLP, $f_h(\cdot)$, has learned to process the raw input data in a way to facilitate the network component concerned about the downstream classification task ($f_t(\cdot)$).

The composition noted in Equation 2.1 expresses a fundamental principle that is present in most neural network-based approaches in machine learning. In the following section, we further elaborate on this concept in the light of different computer vision tasks relevant for this thesis.

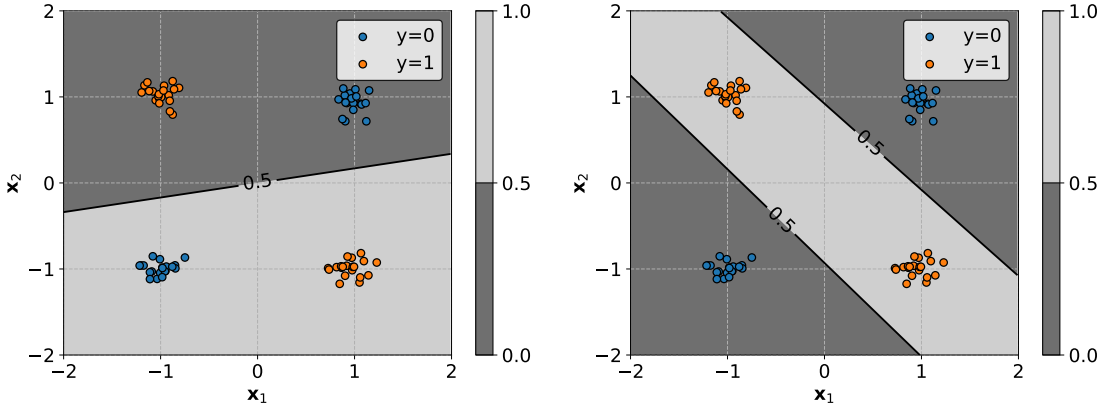


Figure 2.2: An XOR-inspired classification dataset with two classes (blue and orange), which (**left**) cannot be linearly separated by a logistic regression, but (**right**) can be separated by an Multilayer Perceptron (MLP) having non-linear activation functions. Lines represent decision boundaries, whereas data points in light gray areas are assigned to class 1 and data points in dark gray areas are assigned to class 0.

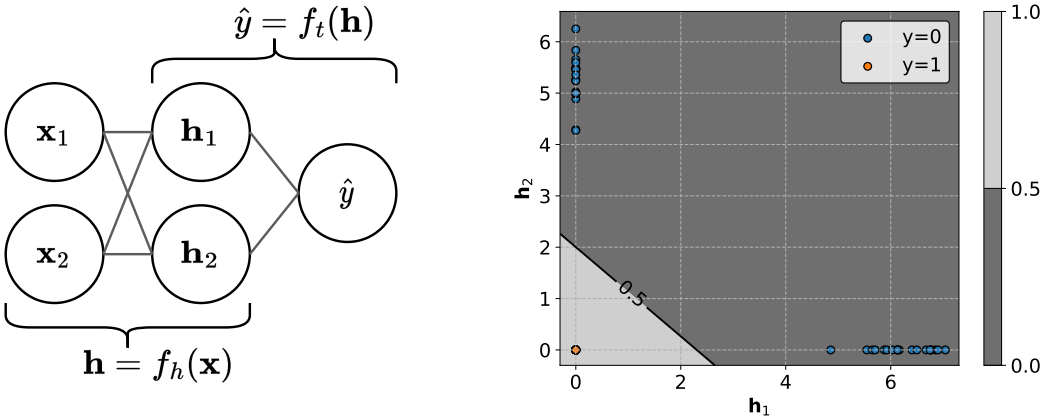


Figure 2.3: (**left**) The Multilayer Perceptron (MLP) in our example has one hidden layer consisting of 2 neurons with ReLU [NH10] activations. We can understand the activations of this hidden layer $\mathbf{h} = f_h(\mathbf{x})$ to be the result of its implicit feature extraction. (**right**) The activations of the hidden layer ($\mathbf{h}_1$ and $\mathbf{h}_2$) can be understood as features extracted for the last layer, acting as logistic regression: $\hat{y} = f_t(\mathbf{h})$. Both classes are now linearly separable.

2.2 Representation Learning in Computer Vision

In 2012, AlexNet [KSH12] was the first Deep Neural Network (DNN) applied to the large scale visual recognition challenge ImageNet [Den+09] and achieved unprecedented results in classifying images. In its core, AlexNet follows the same principle of representation learning models as presented in Equation 2.1. By showcasing the feasibility of training such models on large-scale problems, AlexNet started a paradigm shift from manually engineered features, such as SIFT [Low04] or SURF [BTG06], towards methods based on representation learning [BCV13]. This shift allowed for the automatic extraction of features from raw image data, making it foundational for many computer vision applications today [Sze22; TIF24].

In the following subsections, we describe different computer vision tasks and concepts relevant for this thesis. Subsection 2.2.1 introduces classification on image instance level, whereas Subsection 2.2.2 introduces classification on pixel level. Further, we introduce image synthesis in Subsection 2.2.3 and conclude this section by describing principle ideas of transfer learning in Subsection 2.2.4.

2.2.1 Image Classification

Image classification is one of the fundamental tasks in computer vision, where the goal is to assign a label to an input image based on the presence of an object class [Sze22; TIF24]. It serves as a cornerstone for numerous real-world applications, ranging from facial recognition [Mas+18] to medical image analysis [Li+23], and forms the foundation for more complex tasks like image segmentation [Sze22]. In image classification, we usually want our model to learn the composition from Equation 2.1 in such a way that $\hat{\mathbf{y}} = f(\mathbf{x}) \in [0,1]^C$ represents a conditional probability distribution over C classes, where

$$\hat{\mathbf{y}}_c = p(y = c \mid \mathbf{x})$$

denotes the predicted probability of image $\mathbf{x}$ belonging to class $c \in \{1, \dots, C\}$. The final classification can then be obtained via $\hat{y} = \arg\max_c \hat{\mathbf{y}}_c$.

Given a labeled dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$ with N instances, where $\mathbf{x}_i$ is an image instance with corresponding label y_i, we can train such a model with the Cross Entropy (CE) loss [Bis06]:

$$\mathcal{L}_{\text{CE}} = -\frac{1}{N} \sum_{i=1}^{N} \sum_{c=1}^{C} \mathbb{1}(y_i = c) \log p(y = c \mid \mathbf{x}_i),$$

where $\mathbb{1}(y_i = c)$ is an indicator function that is 1 if $y_i = c$ and 0 otherwise. Representation learning plays a critical role for state-of-the-art image classification models such as Convolutional Neural Networks (CNNs) and Vision Transformers (ViTs), described further below.

Convolutional Neural Networks. A CNN is a type of deep neural network that is specifically designed to process data that has a grid-like structure, such as images [Lec+98]. CNNs use convolutional layers to extract hierarchical features that are subsequently combined into broader concepts while preserving spatial information (up to a certain degree). A single (in this example 2-dimensional) feature map is computed via:

$$\mathbf{H}_{[i,j]} = \sum_{h=0}^{H_k-1} \sum_{w=0}^{W_k-1} \sum_{d=0}^{D-1} \mathbf{X}_{[i+h,j+w,d]} \cdot K_{[h,w,d]} + b. \tag{2.2}$$

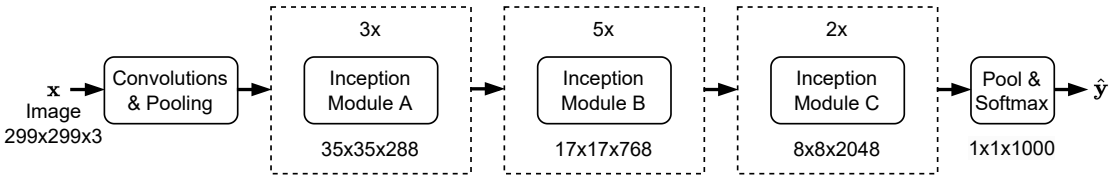


Figure 2.4: Inception v3 architecture [Sze+16] trained on ImageNet [Den+09]. Images and features are tensors of size $H \times W \times D$, whereas $H \times W$ is the spatial resolution (width and height) and D is the number of channels. Each inception module performs multiple convolution operations in parallel, with the outputs of these operations subsequently concatenated to form the final output. After the last inception module, feature maps are collapsed into a feature vector via average pooling and then fed into a softmax classifier [Bis06].

Here, $\mathbf{H}_{[i,j]}$ is the resulting 2-dimensional feature map at spatial location $[i, j]$, $\mathbf{X} \in \mathbb{R}^{H \times W \times D}$ is the 3-dimensional input tensor to the convolutional layer, K is the convolutional kernel of height H_k, width W_k, and depth D, and b is a bias term. We omitted extensions like padding and stride for brevity. Usually, the convolutional kernel and the bias are learned parameters of a layer, and a layer contains multiple learned kernels that result in a stack of feature maps [GBC16].

Due to their local connectivity and weight sharing, convolutional layers are well-suited for processing image data [GBC16]. Local connectivity ensures that neurons are connected only to spatially nearby pixels, allowing CNNs to focus on localized patterns (like edges) in an image. Weight sharing enables the same convolutional filters to be applied across different regions of the image, reducing the number of parameters and ensuring that learned features are translational equivariant. This allows CNNs to recognize objects regardless of their position within the image. Assumptions like these that are encoded in the model design are called inductive biases [Sze22]. We discuss inductive biases and their role in regularization more in Section 2.3.

A common network design for CNNs is to stack multiple convolutional layer and gradually reduce the grid size of the resulting feature maps while the number of channels of intermediate representations increases. As an example, Figure 2.4 depicts the architecture of Inception v3 [Sze+16] with the respective resolutions and number of feature maps. This design enables CNNs to learn a hierarchy of features of different granularity [ZF14]. A visualization of learned features in different depths of a CNN can be found in Zeiler and Fergus [ZF14]. They show that early layers of the network learn to capture edges, gradients, and colors. These features are shared across a wide range of images and tasks, potentially making them good candidates for generalizable features [Yos+14]. Deeper layers of the network learn to detect localized patterns, such as shapes, parts of objects, or textures. When, for example, classifying animals, these features might correspond to parts like eyes, ears, or tails.

Various CNN architectures have been proposed over the years. Popular choices include AlexNet [KSH12], which popularized CNNs in computer vision by winning the 2012 ImageNet challenge [Den+09]. It combined ReLU [NH10] activations with training techniques like dropout [Sri+14] and data augmentation to mitigate overfitting. VGGNet [SZ15] emphasized small kernel sizes of 3×3 convolutions, showing that increasing depth enhances feature representation by showing that learned features generalise to other datasets. GoogLeNet [Sze+15] introduced Inception modules, which used multi-scale feature extraction within the same layer, improving efficiency over VGGNet. Later versions of Inception (v2 and v3 [Sze+16], v4 [Sze+17]) refined the architecture with factorized convolutions and batch normalization [IS15], further reducing computational cost and improving the training process. ResNet [He+16], one of the most

influential architectures, introduced residual connections. Residual connections allow the output $\mathbf{H}^l$ of a layer l to be directly added to the output of the transformation of the next layer $f_{l+1}(\cdot)$, forming a skip connection expressed as $\mathbf{H}_{l+1} = f_{l+1}(\mathbf{H}^l) + \mathbf{H}^l$. This allowed networks to be deeper than before by mitigating the vanishing gradient problem [BSF94]. DenseNet [Hua+17] built upon ResNet by introducing dense connectivity, where each layer receives feature maps from all preceding layers (permitted by resolution), leading to improved parameter efficiency and gradient flow. MobileNet [How+17] introduced depthwise separable convolutions, which split a standard convolution into two operations. First, a depthwise convolution applies a single filter per input channel, then a pointwise convolution combines the results. This reduces computational cost and makes the model efficient for mobile and edge devices. EfficientNet [TL19] introduced compound scaling, which systematically balances depth, width, and resolution of the network. They show that this scaling can improve performance over other network designs while requiring fewer number of parameters. More recently, ConvNeXt [Liu+22] modernized CNN architectures by removing redundant complexities from ResNet, incorporating transformer-like design principles, and enhancing optimization hyperparameters. In Chapter 9 we evaluate the effect of architectural design choices of CNNs on their performance in the context of Neural Architecture Search (NAS).

Vision Transformers. Due to their inductive bias, CNNs generalize well from limited data and efficiently learn hierarchical feature representations [WW23]. However, while these inductive biases are advantageous for many tasks, they also limit the ability of CNNs towards long-range dependencies. This limitation stems from the fact that convolutions operate within a restricted receptive field [GBC16]. Due to the availability of large image datasets, a new paradigm emerged as an alterantive to CNNs in recent years: Vision Transformers (ViTs) [Kol+21]. This model adapts the principles of the Transformer model [Vas+17] (originally developed for natural language processing) to computer vision tasks. The core innovation of Transformers is their self-attention mechanism, which enables the model to attend to different parts of the input sequence and capture global dependencies.

For images, ViTs directly model global relationships by treating an image as a sequence of image patches. Each patch is treated as a token, analogous to words in a sentence, enabling the self-attention mechanism to capture interactions across the entire image. Usually, image tokens $\mathbf{z}^0$ are linear projections from an image patch, such that:

$$\mathbf{z}_i^0 = \mathbf{W}_p \mathbf{x}_i + \mathbf{e}_i,$$

where $\mathbf{x}_i$ is the flattened image patch at position i, $\mathbf{W}_p$ is a learnable projection matrix, and $\mathbf{e}_i$ is a positional encoding to retain spatial information. The image is then expressed as a sequence of tokens of length $N + 1$:

$$\mathbf{Z}^0 = [\mathbf{z}_{\text{class}}^0; \mathbf{z}_1^0; \mathbf{z}_2^0; \ldots; \mathbf{z}_N^0],$$

where $\mathbf{z}_{\text{class}}^0$ is a learnable class token that serves as global image representation. This sequence is passed through multiple transformer encoder layers, whereas each layer modifies the containing tokens via self-attention weights:

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\sqrt{d_k}}\right)\mathbf{V},$$

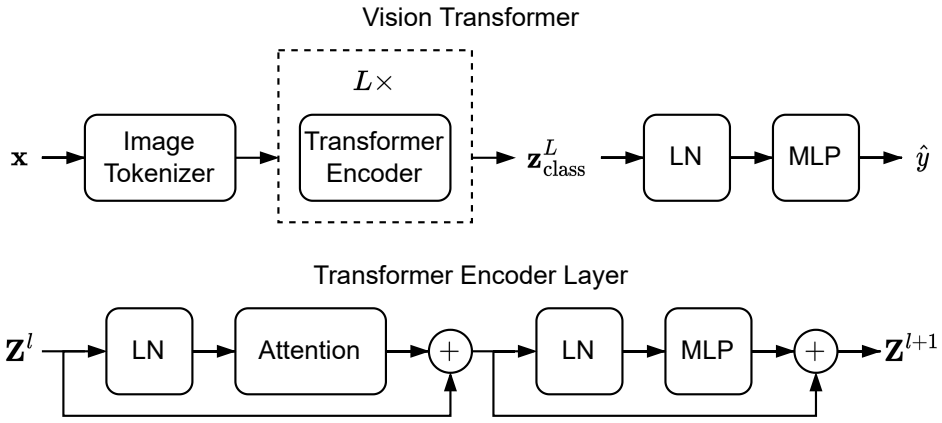


Figure 2.5: (**top**) Depiction of the Vision Transformer [Kol+21] architecture. An image $\mathbf{x}$ is tokenized by projecting image patches to image tokens and adding positional information. These image tokens are modified via L Transformer encoder layers. Then, class token $\mathbf{z}^L_{class}$ is extracted and layer normalized (LN) before performing classification via an MLP. (**bottom**) Depiction of a Transformer encoder layer. Token embeddings $\mathbf{Z}^L$ are modified via self-attention and then passed through an MLP. Layer normalizations (LN) and skip connections in-between improve gradient flow.

where d_k is a scaling factor to improve gradient flow, and matrices:

$$\begin{aligned}\mathbf{Q} &= \mathbf{Z}^l\mathbf{W}_Q, \quad \text{(Query matrix)}\\ \mathbf{K} &= \mathbf{Z}^l\mathbf{W}_K, \quad \text{(Key matrix)}\\ \mathbf{V} &= \mathbf{Z}^l\mathbf{W}_V, \quad \text{(Value matrix)}\end{aligned}$$

where $\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$ are learnable projection matrices, and $\mathbf{Z}^l$ is the token sequence after layer l. To compute the token sequence of the next layer $\mathbf{Z}^{l+1}$, the transformer encoder layer passes the attention output through an MLP, with residual connections and layer normalizations in-between. For image classification, the class token $\mathbf{z}^L_{\text{class}}$ after the last transformer encoder layer is used as input to an MLP (or a linear classifier). We depict the whole process in Figure 2.5.

Different ViT architectures and improvements have been developed in recent years. The Vision Transformer [Kol+21] was the first model to successfully apply the Transformer architecture to vision tasks by tokenizing images into fixed-size patches and processing them similarly to words, demonstrating state-of-the-art performance on larger-scale datasets like JFT-300M [Sun+17] but struggling with data efficiency. The Data-efficient image Transformer (DeiT) [Tou+21] addressed the data inefficiency problem by introducing knowledge distillation with a teacher-student training approach. Here, a vision transformer is combined with a distillation token that learns from a CNN teacher. This enables efficient training on smaller datasets like ImageNet without requiring large-scale pretraining. The Shifted Window Transformer (Swin) [Liu+21] introduced hierarchical feature maps and localized attention using non-overlapping shifted windows, making it computationally efficient and scalable for high-resolution images. The Cross-Attention Vision Transformer (CrossViT) [CFP21] enhances ViTs by using patch embeddings from different image patch sizes and cross-attention mechanisms. In this context, cross-attention refers to an attention process where the query matrix $\mathbf{Q}$ from one scale attends to

the key **K** and value **V** matrices from another scale. This allows the model to simultaneously capture fine-grained and global image features. The Token-to-Token Vision Transformer (T2T-ViT) [Yua+21] refined patch tokenization by progressively aggregating tokens through overlapping convolutions, reducing redundancy and improving feature representation, leading to better data efficiency and performance on smaller datasets.

From Image-Level to Pixel-Level Classification. While image classification assigns a single label to an entire image, many real-world applications require a more detailed, pixel-level understanding. Here, a class or value is assigned to each individual pixel in an image. This is important, for example, in applications such as medical imaging [Est+21], environmental monitoring [Tui+22], and autonomous driving [Jan+20], where precise localization of structures or objects is essential. In the next subsection, we introduce pixel-wise classification tasks, as well as key methodologies to generate high-resolution predictions.

2.2.2 Pixel-Wise Classification

Pixel-wise classification is the fundamental problem in computer vision, where each pixel in an image is assigned a class label. To assign a label to each pixel, commonly, our network learns a conditional probability for each pixel location such that:

$$\hat{\mathbf{y}}_{[i,j,c]} = p(y = c \mid \mathbf{x}), \quad \hat{\mathbf{y}} \in [0,1]^{H \times W \times C},$$

where W and H are width and height of an input image $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$, $[i, j]$ is a pixel location, and C is the number of classes. Generally, we want to produce an output that matches the resolution of the input image[1].

We visualize different possible pixel-wise classification tasks in Figure 2.6. One of the most basic pixel-wise classification tasks is edge detection. Here, the goal is to binary classify each pixel in an image whether it belongs to an object boundary or not. The result is an edge map (a), which can then be used to form coherent regions of pixels (segments) that belong together. The task of finding arbitrary groups of pixels (clusters) that belong together is called image segmentation (b). Edge detection and image segmentation are connected, as we can transform an edge map to a partition an image, and vice versa. When we assign an object label to each pixel in an image, for example tree, horse, or grass, then this task is called semantic segmentation (c). When we instead want to differentiate between individual instances of the same object class, then this task is called instance segmentation (d). Last, we can combine semantic and instance segmentation, where both a class label and an instance is assigned to each pixel, into panoptic segmentation (e).

Encoder-Decoder Networks. CNNs are subsequently reducing the resolution when extracting features (see Subsection 2.2.1) and hereby encode an image into features. Consequently, when using networks like CNNs for pixel-wise classification tasks, we have to increase the resolution after a certain point in the network again to (better) match the resolution of output to input. This part of the network is called upsampling and (if learned) a decoder. There are multiple ways to achieve upsampling, whereas common approaches include encoder-decoder structures [LSD15] as depicted in Figure 2.7. In the context of CNNs for pixel-wise classification, these are typically Fully Convolutional Neural Networks (FCNs) [LSD15; RFB15]. In contrast to conventional CNNs

[1]While it might be generally desired that input and output resolutions match, this is not a hard requirement and can depend on the resolution of ground truth labels provided or other factors.

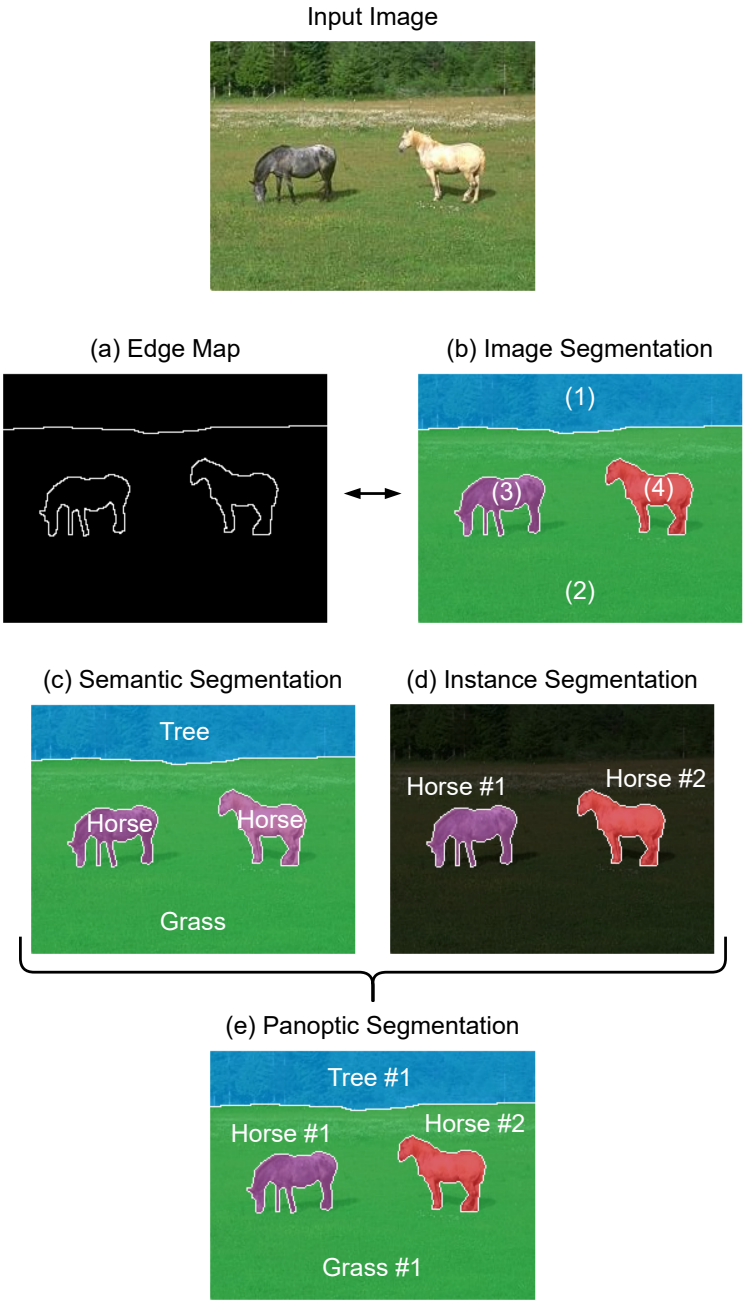


Figure 2.6: The results of edge detection and different types of segmentation tasks on an input image from BSDS300 [Mar+01]. (**a**) Edge detection produces edge maps that assign a binary label for each pixel, indicating whether it belongs to an edge or not. (**b**) The edge map in (a) can be used to partition the image into different regions. (**c**) Each pixel in the image is assigned a class label. (**d**) Each pixel in the image is assigned an instance id for a certain class, identifying which instance it belongs to (or none). (**e**) The combination of (c) and (d) produces a class label for each pixel in the image, as well as an instance id.

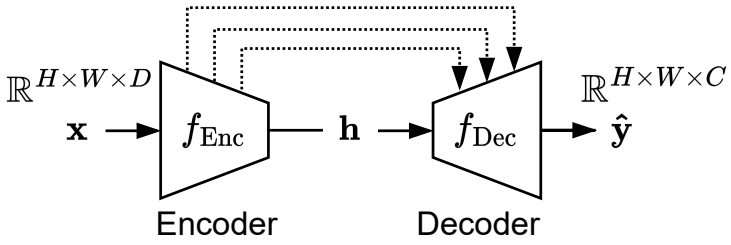


Figure 2.7: Visualization illustrating an encoder-decoder architecture. Given an input image $\mathbf{x} \in \mathbb{R}^{H\times W\times D}$ the encoder part f_{Enc} of the network downsamples the feature resolutions leading up to a bottleneck representation $\mathbf{h}$. This representation is then subsequently upsampled in resolution by decoder f_{Dec} until the desired output resolution is obtained, leading to output $\hat{\mathbf{y}} \in \mathbb{R}^{H\times W\times D}$ (assuming C classes). Optional skip connections (dotted lines) preserve finer details from earlier layers of the encoder network.

for image classification, FCNs are designed without fully-connected layers and can therefore handle arbitrary input resolutions.

Upsampling in the decoder can be achieved through different techniques. Transposed convolutions [ZF14] work by reversing the process of a standard convolutional layer (see Equation 2.2). Instead of reducing spatial dimensions, they expand feature maps by inserting learnable weights and overlapping filters to generate a higher-resolution output. This enables trainable upsampling but can introduce checkerboard artifacts [ODO16]. An alternative approach is interpolation of feature maps, for example via nearest-neighbor or bilinear upsampling [GW18]. Here, pixel values are computed based on the value of surrounding pixels. Since no learnable parameters are introduced, interpolation is less adaptive compared to transposed convolutions. This can be mitigated, for example, by combining bilinear upsampling with a subsequent standard convolutional layer as demonstrated by Odena, Dumoulin, and Olah [ODO16]. There are also methods that reverse the pooling operation from the encoder network in the decoder. For example Badrinarayanan, Kendall, and Cipolla [BKC17] proposed using the max-pooling indices from the encoder to perform non-learned upsampling, which places activations back into their original spatial locations. This produces sparse feature maps that are then densified through trainable convolutional layers.

Chapter 4 covers the tasks of edge detection and image segmentation. Hence, we focus the following related works on these two subjects.

Edge Detection Methods. Solving the task of edge detection relied on classical techniques such as the Sobel operator [KVB88], Canny edge detector [Can86], and Laplacian of Gaussian [MH79], which use handcrafted filters to detect edges based on image gradients. However, image gradients are sensitive to noise and varying illumination conditions [GW18]. More recent edge detection methods are based on representation learning-neural networks that can learn to extract more robust features in this regards [BST15; She+15; XT15; Man+16a; Liu+19; He+19; Pu+22]. For example, Bertasius, Shi, and Torresani [BST15] combine the Canny edge detector with a CNN classifying contour candidates. Shen et al. [She+15] categorize image patches into possible contour shapes (or no contour) with a CNN. Holistically-Nested Edge Detection (HED) [XT15] is a CNN that uses side outputs at multiple layers to learn hierarchical edge features. Resulting edge maps from side outputs are upsampled via bilinear interpolation and aggregated

to compute a final output. Richer Convolutional Features (RCF) [Liu+19] extends HED by aggregating hierarchical features across all convolutional layers, instead of only selected ones. In Chapter 4 we improve the training scheme of RCF with penalty terms that encourage closed contours, resulting in finer-grained edge contours.

Image Segmentation Methods. The goal of image segmentation is to partition an image into coherent regions. For this, classical methods like Watershed Transform [Beu79; VS91] define segmentation as a topographic watershed problem, using image gradients to separate regions. Image segmentation can also be understood as clustering pixels in the image. Hence, some methods are based on clustering approaches such as Mean Shift [CM02], or based on graph-partitioning methods such as Normalized Cuts [SM00; Arb+14] and methods formulating image segmentation as Minimum Cost Multicut Problem (MP) [BBC04; Kim+11; And+11; And+13; BHK15; Keu+15]. Here, the MP is defined on a weighted graph that is partitioned by removing edges. The optimal partitioning is given by minimizing the sum of the weights of edges that are removed. For example, Andres et al. [And+11] propose a probabilistic approach that connects superpixels based on adjecancy and assigns edge weights based on learned boundary probabilities. Beier et al. [Bei+16] extend this approach by introducing higher-order edges, improving segmentation quality in complex images. The MP is essential for Chapter 3 and Chapter 4, as it gives rise to discrete constraints that define valid solutions for it. We formulate penalty terms based on these constraints in both chapters, where we also provide a formal definition of the MP.

Any of the beforementioned methods that predict edge maps can also be used for image segmentation. For this, predicted edge maps have to be transformed into segmentations [Arb+09]. For example Arbelaez et al. [Arb+09] compute initial regions via a watershed transform and then construct a Ultrametric Contour Map (UCM) [Arb06], which is a hierarchy merging local regions based on their similarity. Arbeláez et al. [Arb+14] extends this idea by considering multiple image resolutions.

From Categorical to Continuous Outputs. While pixel-wise classification assigns a discrete label to each pixel, some tasks require continuous values at pixel level. These are necessary for applications such as depth estimation, optical flow, surface normal estimation, and image synthesis [Sze22; GBC16]. Since image synthesis is the key task for Chapter 5, Chapter 6, and Chapter 7, we are introducing principles and architectures thereof in the following subsection.

2.2.3 Image Synthesis

Unlike discriminative computer vision models (which predict labels from images), generative computer vision models for image synthesis produce new images [TIF24]. Such models learn an approximate distribution of the training images and can sample novel images from this distribution. Hence, given an image dataset $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^N$ that originates from an underlying data distribution $p_{\text{data}}(\mathbf{x})$, we are interested in learning its approximate distribution $p_\theta(\hat{\mathbf{x}}, \mathbf{h}) \approx p_{\text{data}}(\mathbf{x})$ over a latent representation space, where $\mathbf{h} \sim p(\mathbf{h})$ is the latent representation of an image sampled from a prior distribution $p(\mathbf{h})$. We can then sample new images via:

$$\hat{\mathbf{x}} = f_{\text{Gen}}(\mathbf{h}),$$

where $\hat{\mathbf{x}}$ is the generated image and f_{Gen} is a function that maps from the latent representation space to the image space. While this formulates the setting of unconditional image synthesis, there is also a conditional setting for which we model $p_\theta(\hat{\mathbf{x}}, \mathbf{h}, \mathbf{c}) \approx p_{\text{data}}(\mathbf{x} \mid \mathbf{c})$ and generate samples via $\hat{\mathbf{x}} = f_{\text{Gen}}(\mathbf{h}, \mathbf{c})$ for some conditioning $\mathbf{c}$ [TIF24]. Here, the image generation process is guided by additional information, such as class labels [MO14] or natural text descriptions

[Man+16b]. In the context of this thesis, we primarily focus on the unconditional image synthesis task. Hence, for simplicity, we assume the unconditional setting throughout the remainder of this thesis.

In recent years, several generative frameworks have been developed for image synthesis, each with different strategies for modeling the data distribution $p_\theta(\hat{\mathbf{x}}, \mathbf{h})$. Prominent approaches that are relevant for this thesis are Variational Autoencoders (VAEs) [KW14], Vector Quantized Variational Autoencoders (VQ-VAEs) [OVK17], and Generative Adversarial Networks (GANs) [Goo+14]. We briefly introduce each of these, and additionally mention diffusion-based models [Soh+15], which are the generative backbone of many modern image synthesis methods [Ram+22; Sah+22a; Rom+22]. Finally, we discuss common metrics for evaluating image synthesis quality [Sal+16; Heu+17].

Variational Autoencoders. VAEs are a fundamental approach to image synthesis introduced by Kingma and Welling [KW14]. A VAE consists of an encoder-decoder pair of networks (forming an autoencoder architecture [HS06], see Figure 2.8) with a stochastic element: The encoder $f_{\text{Enc}}(\mathbf{x})$ models a probabilistic mapping $q_\theta(\mathbf{h}|\mathbf{x})$ from an input image to a latent representation, and the decoder $f_{\text{Dec}}(\mathbf{h})$ models a generative function $p_\theta(\hat{\mathbf{x}}|\mathbf{h})$ reconstructing an image from the latent sample drawn. In practice, the encoder produces parameters (for example, mean and variance) of a simple distribution (typically a centered isotropic multivariate Gaussian $p_\theta(\mathbf{h}) = \mathcal{N}(\mathbf{h}; \mathbf{0}, \mathbf{I})$) in a low-dimensional latent space. A random latent vector is then sampled (using the reparameterization trick [KW14] for differentiability during training), and passed through the decoder to reconstruct the image. We can define this process as follows:

$$\mathbf{h} = \mu_\theta(\mathbf{x}) + \sigma_\theta(\mathbf{x}) \cdot \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

and $\mu_\theta(\mathbf{x})$ and $\sigma_\theta(\mathbf{x})$ are mean and standard deviation produced by $f_{\text{Enc}}(\mathbf{x})$. The model is trained by maximizing a variational lower bound of the data likelihood. Given the Kullback-Leibler Divergence (KLD) $D_{\text{KL}}(p\|q)$ between two distributions p and q, the loss is computed via:

$$\mathcal{L}_{\text{VAE}} = \mathbb{E}_{q_\theta(\mathbf{h}|\mathbf{x})}[-\log p_\theta(\hat{\mathbf{x}} \mid \mathbf{h})] + D_{\text{KL}}(q_\theta(\mathbf{h} \mid \mathbf{x}) \parallel p(\mathbf{h})).$$

This objective balances two terms: (1) a reconstruction term that encourages the decoded image to match the input image, and (2) a regularization term that pushes the latent distribution toward a chosen prior (usually a standard normal). By this means, VAEs learn an explicit probabilistic model of the data. One can then sample the latent prior distribution and feed it to the decoder to generate new images.

VAEs provide a principled framework for generative modeling with latent variables. They often produce images that capture the coarse structures of the data, but tend to be blurrier compared to other generative methods. Dosovitskiy and Brox [DB16] attribute this blurriness to the reconstruction error, which becomes equivalent with the squared Euclidean error in VAEs when the latent space is assumed to be Gaussian distributed. Despite this, VAEs are valuable for their explicit representation of uncertainty in the latent space and the ability to interpolate between images by traversing the smooth latent space [KW14]. Numerous extensions and follow-up works [Gul+17b; Hig+17; OVK17; Chi21] to VAEs exist to improve, for example, generation quality. One of these extensions that we use in Chapter 7 are VQ-VAEs, which we discuss next.

Vector Quantized Variational Autoencoders. VQ-VAEs is a variant of the variational autoencoder that introduces a discrete latent space, combining ideas from VAEs and discretization via vector quantization. Proposed by Oord, Vinyals, and Kavukcuoglu [OVK17], VQ-VAE replaces the continuous latent variables of a standard VAE with d-dimensional latent code vectors drawn

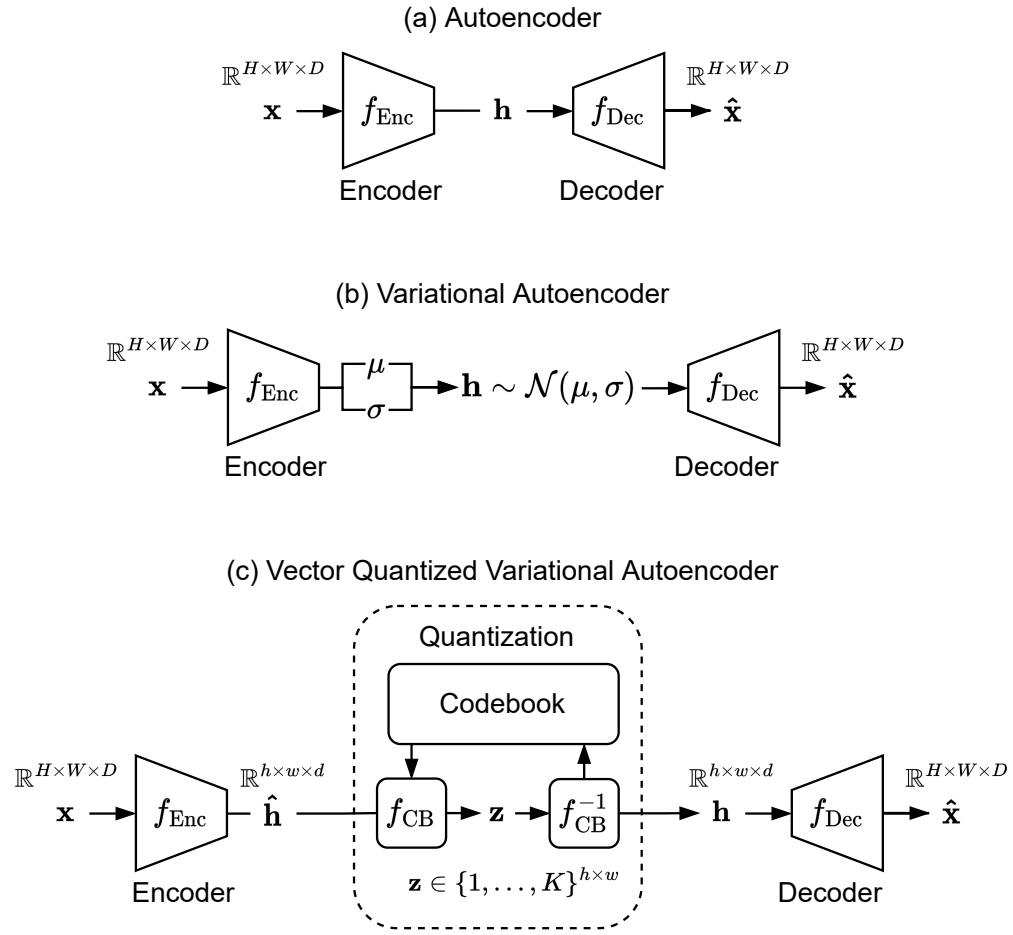


Figure 2.8: Visualization comparing the architectures of autoencoders, VAEs, and VQ-VAEs. **(a)** An encoder f_{Enc} maps input image $\mathbf{x} \in \mathbb{R}^{H\times W\times D}$ to latent representation $\mathbf{h}$ from which f_{Dec} reconstructs image $\hat{\mathbf{x}}$. **(b)** Instead of mapping to a latent representation directly, f_{Enc} computes parameters of a latent distribution from which $\mathbf{h}$ is sampled. **(c)** Here, the output of encoder f_{Enc} is quantized (via a finite codebook containing K learnable latent code vectors $\mathbf{e}_k$) into a grid of categorical latent variables $\mathbf{z}$, whereas $\mathbf{z}_{[i,j]} = \arg\min_k \|\hat{\mathbf{h}}_{[i,j]} - \mathbf{e}_k\|^2$. For the decoder, each latent variable is replaced with the corresponding codebook vector $\mathbf{h}_{[i,j]} = \mathbf{e}_{\mathbf{z}_{[i,j]}}$.

from a finite codebook $\{\mathbf{e}_k\}_{k=1}^K$, $\mathbf{e}_k \in \mathbb{R}^d$ of length K. In a VQ-VAE, the encoder f_{Enc} outputs a continuous latent representation $\hat{\mathbf{h}} \in \mathbb{R}^{h \times w \times d}$ given an input image $\mathbf{x}^{H \times W \times D}$, which is then quantized (see Figure 2.8). Each latent vector is replaced by the nearest learnable codebook

$$\mathbf{z}_{[i,j]} = \arg\min_k \|\hat{\mathbf{h}}_{[i,j]} - \mathbf{e}_k\|_2^2, \quad \mathbf{z} \in \{1, \dots, K\}^{h \times w}, \quad \mathbf{h}_{[i,j]} = \mathbf{e}_{\mathbf{z}_{[i,j]}},$$

where $[i, j]$ is the spatial location of the feature vector, and $\mathbf{z}$ is a grid of categorical latent variables. The decoder then takes these codebook embeddings as input to reconstruct the image via:

$$\hat{\mathbf{x}} = f_{\text{Dec}}(\mathbf{h}).$$

Because the latent space is discrete, the VQ-VAE training uses a straight-through estimator or a codebook update mechanism (instead of the reparameterization trick) to propagate gradients through the quantization step. The total loss for VQ-VAE consists of three terms:

$$\begin{aligned} \mathcal{L} &= \mathcal{L}_{\text{recon}} + \mathcal{L}_{\text{vq}} + \mathcal{L}_{\text{commit}}, \\ \mathcal{L}_{\text{recon}} &= \|\mathbf{x} - \hat{\mathbf{x}}\|_2^2, \\ \mathcal{L}_{\text{vq}} &= \|\text{sg}[\hat{\mathbf{h}}] - \mathbf{h}\|_2^2, \\ \mathcal{L}_{\text{commit}} &= \beta \|\hat{\mathbf{h}} - \text{sg}[\mathbf{h}]\|_2^2, \end{aligned}$$

where $\text{sg}[\cdot]$ is the stop-gradient operator, which prevents gradients from flowing into the encoder, and hyperparameter β controls the strength of the commitment loss. The reconstruction loss measures how well the decoder reconstructs $\mathbf{x}$, the vector quantization loss ensures latent vectors are replaced by their nearest codebook vectors, and the commitment loss encourages the encoder output $\hat{\mathbf{h}}$ to stay close to the selected codebook vector $\mathbf{h}$.

During training, the prior distribution of VQ-VAEs is typically uniform [OVK17]. After training, a learned autoregressive prior can be fitted for high-quality image generation, for example using a PixelCNN [Oor+16]. This two-stage approach has proven effective, for example in advanced image synthesis systems like DALL-E [Ram+21]. In Chapter 7 of this thesis, we bias the discrete latent space of VQ-VAEs by sampling globally optimal codes from it, based on user-defined preferences.

Generative Adversarial Networks. GANs are a class of generative models introduced by Goodfellow et al. [Goo+14]. A GAN consists of two neural networks, a generator network and a discriminator network, that play a minimax (zero-sum) game against each other. The generator network takes a random input (noise vector) and learns to produce real images (from the training distribution), while the discriminator network tries to distinguish between real images and images produced by the generator (see Figure 2.9). Through this adversarial training process, the generator ideally learns to generate images that are indistinguishable from the training data.

Formally, generator network f_{G} learns to produce images from randomly sampled representations $\mathbf{h} \sim p(\mathbf{h})$, whereas a common choice for $p(\mathbf{h})$ is the standard normal distribution $\mathcal{N}(0, 1)$. Discriminator network f_{D} is a binary classifier, mapping input samples $\mathbf{x}$ to realness probabilities $f_{\text{D}} : \mathbf{x} \to [0, 1]$. While f_{D} is trained to maximize its ability to distinguish real from generated samples, f_{G} is trained to minimize it. Formulated as a minimax game [Goo+14]:

$$\min_{f_{\text{G}}} \max_{f_{\text{D}}} V(f_{\text{G}}, f_{\text{D}}), \quad \text{with objective function}$$
$$V(f_{\text{G}}, f_{\text{D}}) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log f_{\text{D}}(\mathbf{x})] + \mathbb{E}_{\mathbf{h} \sim p_{\mathbf{h}}(\mathbf{h})}[\log(1 - f_{\text{D}}(f_{\text{G}}(\mathbf{h})))].$$

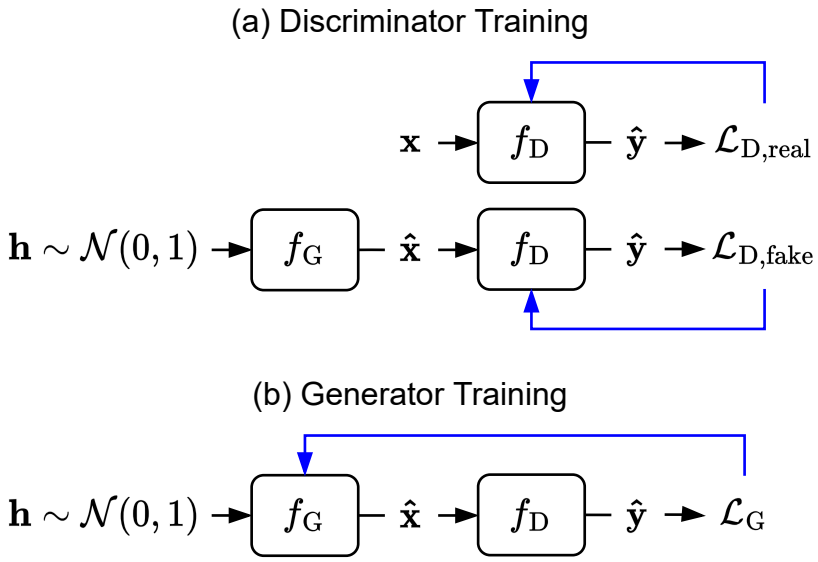


Figure 2.9: Visualization of GAN training. Discriminator network f_{D} is distinguishing training data samples $\mathbf{x}$ from generated samples $\hat{\mathbf{x}}$. Blue lines indicate the backpropagation target. (a) The discriminator network is trained to predict whether an image is from the training data or not. (b) The generator network f_{G} tries to fool the discriminator by generating samples close to the training data, while the discriminator provides the training signal to the generator network.

The core idea of GAN training is that the generator network does not receive direct supervision on how far its output is from a ground-truth image, but instead it improves by learning to make the discriminator network unable to tell training data images and generated images apart. This indirect training signal can yield sharp and detailed results, as the generator is encouraged to model high-fidelity details that convince the discriminator [Kar+20b; BDS19]. However, training GANs can be challenging due to unstable training dynamics. For example, the generator might collapse to producing limited varieties of images (mode collapse), when discriminator and generator are not synchronized well during training [Goo+14].

Since their introduction, numerous improvements have been proposed for GANs to enhance their stability and sample quality. For example, Deep Convolutional Generative Adversarial Network (DCGAN) [RMC15] introduced a guideline for CNN-based GAN architectures that improve training stability over fully-connected GANs. Mao et al. [Mao+17] attribute vanishing gradients to the sigmoid loss in the typical loss formulation [Goo+14]. They introduce Least Squares Generative Adversarial Network (LSGAN), which replaces the standard binary cross-entropy loss with a least squares loss for the discriminator, improving training stability and sample quality further. Similarly, for Wasserstein Generative Adversarial Network (WGAN) [ACB17] and Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [Gul+17a], the Jensen-Shannon divergence used in standard GANs is replaced with the Wasserstein distance to improve training stability. Spectral Normalization Generative Adversarial Network (SNGAN) [Miy+18] stabilize training by controlling the Lipschitz constant of the discriminator with spectral normalization, which constraints the spectral norm of each weight matrix. Karras et al. [Kar+17] introduces a progressively growing architectural design

for GANs, which gradually increases image resolution over the duration of the training. This enables to produce images with higher resolution than before. BigGAN [BDS19] scales GAN to larger models, enabling class-conditioned generation of high-resolution images based on ImageNet [Den+09]. They also introduce the truncation trick, which limits the range of the noise vector an image is sampled from, trading off sample diversity for quality. In StyleGAN [KLA19; Kar+20b], the generator samples from an additional (style-based) latent space, which enables fine-grained control over image synthesis. In Projected GAN [Sau+21], the discriminator network uses features extracted by pretrained image classification networks, which leads to a substantial reduction in training runtime. In Chapter 5 we also use pretrained features to penalize GANs, and in Chapter 6 we add an additional discriminator network in the frequency domain of images to cover this blind spot of existing GAN architectures.

All previously mentioned approaches employ parameterized functions implemented as neural networks to discriminate between training data images and generated images. Another line of approaches compares distributions of both image sources by comparing their statistics in a distance-based manner instead [LSZ15; Li+17]. Generative Moment Matching Networks (GMMNs) replace the learned discriminator in traditional GANs with a statistical test based on Maximum Mean Discrepancy (MMD), essentially employing a parameter-free discriminator. Li, Swersky, and Zemel [LSZ15] train the generator to produce samples that match all moments of the training data distribution by minimizing MMD using a fixed kernel, for example a Gaussian kernel. This design eliminates adversarial dynamics and enables stable training, but comes at the cost of quadratic computational complexity with respect to the number of samples when evaluating the objective. Similarly, in Chapter 5 we employ a parameter-free discriminator by using Fréchet Inception Distance (FID) as distance metric comparing features extracted by a parameterized but fixed feature extractor to train generative models.

Challenges, such as the discussed training instabilities leading to mode collapse, remain fundamental limitations of GANs. In contrast, diffusion-based models [Soh+15] have emerged as an alternative, leveraging a probabilistic denoising process to generate high-quality samples with improved stability and sample diversity [DN21]. While not a part of this thesis, we briefly introduce the idea behind diffusion models for completeness in the next section.

Diffusion-Based Models. Diffusion models are a newer class of deep generative models that have grown in importance for image synthesis [Soh+15; Che+25]. These models generate images by learning to iteratively denoise random noise. During training, noise is gradually added to images, transforming the data distribution into pure noise. The model learns to invert this process by iteratively removing noise to recover the original data. Once trained, image synthesis is performed by starting from a random noise sample that is progressively denoised using the learned reverse process. Ho, Jain, and Abbeel [HJA20] demonstrate with their Denoising Diffusion Probabilistic Model (DDPM) that diffusion-based generators can produce high-quality images. In fact, Dhariwal and Nichol [DN21] show that diffusion models can achieve image sample quality comparable to or even surpassing GANs on certain benchmarks. However, a downside of these models is that generating an image is typically slower, as it requires many iterative denoising steps. Despite this, due to their high synthesis quality, stable training, and scalability to large datasets [DN21], they became the underlying approach of recent high-profile systems [Ram+22; Rom+22; Sah+22b; Kar+22; Xie+25].

Image Synthesis Evaluation Metrics. Evaluating the quality of generated images is a non-trivial task, as it involves assessing both how realistic the images are and how diverse they are relative to the training data. Human evaluation is the gold standard for judging image realism,

but it is time-consuming and not scalable. Instead, the research community relies on a few quantitative evaluation metrics to compare generative models. Two of the most commonly used metrics in image synthesis are Inception Score (IS) [Sal+16] and FID [Heu+17].

Inception Score. IS [Sal+16] uses a pretrained Inception-v3 image classification network [Sze+16] to evaluate generated images. The basic idea is that a good generative model should produce images that (a) look like clear examples of some recognizable object/class (high confidence predictions from the classifier, meaning the predicted label distribution of each image has low entropy), and (b) collectively cover a wide variety of classes (the overall distribution of predicted labels across many generated images has high entropy, indicating diversity). Mathematically, we can formalize IS as:

$$\text{IS} = \exp\left(\mathbb{E}_{\hat{\mathbf{x}}\in\mathcal{D}}\left[\text{KL}\left(p(y|\hat{\mathbf{x}}) \parallel p(y)\right)\right]\right),$$

where $\mathbb{E}_{\hat{\mathbf{x}}\in\mathcal{D}}$ is the expectation over a dataset of generated images $\mathcal{D} = \{\hat{\mathbf{x}}_i\}_{i=1}^{N}$, $p(y|\hat{\mathbf{x}})$ is the conditional label distribution for a generated image $\hat{\mathbf{x}}$, $p(y)$ is the marginal class distribution, and $KL(p(y|\hat{\mathbf{x}})\|p(y))$ is the Kullback-Leibler divergence between the conditional and marginal distributions. Intuitively, it rewards images that the classifier finds distinct and classifiable and penalizes lack of diversity. A higher IS suggests better quality and diversity. However, IS has some known shortcomings, for example its sensitivity to the specific weights of the pretrained classifier and its unreliable behavior when applied to datasets beyond those it was trained on [BS18]. It also evaluates only the generated images without directly comparing them to training data [Heu+17]. To overcome these shortcomings, Heusel et al. [Heu+17] propose an alternative metric called Fréchet Inception Distance (FID).

Fréchet Inception Distance. The FID has become the de facto standard for evaluating image synthesis in recent years [Heu+17; Bor22; Jay+24]. FID also leverages a pretrained Inception network, but compares extracted image features instead of predicted labels. Concretely, a set of generated images and a set of training data images are passed through the Inception v3 model $f_h(\cdot)$ to obtain feature representations. The FID treats the set of Inception features for training images and for generated images as two distributions (modeled as multivariate Gaussians) and computes the Fréchet distance between both:

$$\text{FID} = \|\boldsymbol{\mu}_{\mathcal{D}_1} - \boldsymbol{\mu}_{\mathcal{D}_2}\|_2^2 + \text{tr}(\boldsymbol{\Sigma}_{\mathcal{D}_1} + \boldsymbol{\Sigma}_{\mathcal{D}_2} - 2\sqrt{\boldsymbol{\Sigma}_{\mathcal{D}_1}\boldsymbol{\Sigma}_{\mathcal{D}_2}}),$$

between two image sources $\mathcal{D}_1$ and $\mathcal{D}_2$, where $tr(\cdot)$ is the trace operator, and

$$\boldsymbol{\mu}_{\mathcal{D}} = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} f_h(\mathbf{x}) \qquad \text{and} \qquad \boldsymbol{\Sigma}_{\mathcal{D}} = \sum_{\mathbf{x}\in\mathcal{D}} \frac{\left(f_h\left(\mathbf{x}\right) - \boldsymbol{\mu}_{\mathcal{D}}\right)\left(f_h\left(\mathbf{x}\right) - \boldsymbol{\mu}_{\mathcal{D}}\right)^{\top}}{|\mathcal{D}| - 1}$$

are the mean vector and covariance matrix of feature vectors of dataset $\mathcal{D}$.

The resulting distance reflects the difference in feature means and covariance between real and generated image sets. Lower FID indicates that the generated images are more similar to real images in the feature space, hence higher fidelity and diversity of the generation. FID has been shown to correlate better with human judgment compared to IS [Heu+17]. However, FID is also criticized for certain biases and shortcomings, which we discuss further in Chapter 5.

2.2.4 Transfer Learning

In traditional machine learning, we train a model from scratch for each new task, assuming training and test data come from the same data distribution [PY10; SB14]. Transfer learning, by contrast, facilitates learnings that were made from previous tasks [GBC16]. Formally, Pan and Yang [PY10] define transfer learning as aiming to improve the target predictive function f on the target task T_T in target domain D_T using knowledge from a source domain D_S and source task T_S, where either the domains or tasks are not the same (hence, either $D_S \neq D_T$ or $T_S \neq T_T$, or both). In other words, transfer learning is applicable when the data distribution differs between domains, or the learning objectives or label space differs. In the case of matching tasks, hence $T_S = T_T$, but distinct data distributions, hence $D_S \neq D_T$ (for example from daytime to nighttime images), transfer learning is also called domain adaptation [PY10]. It is particularly useful in representation learning, where training models from scratch requires substantial computational resources and large datasets. In computer vision, where datasets are often high-dimensional and complex, transfer learning has become an essential technique [JT21] by extracting features from pretrained models or by using their weights as initialization for training.

Feature Extraction. Pretrained models, trained on diverse datasets like ImageNet [Den+09], provide feature representations that can be reused for different vision tasks [Raz+14; Yos+14]. The idea is to take advantage of the representations learned from the source task by extracting them from some layer of a network and feeding them into a new classifier or regressor for the target task. Feature extraction is effective when the source and target tasks are sufficiently related such that the learned representations are relevant [Yos+14]. Razavian et al. [Raz+14] showed that off-the-shelf features from a pretrained ImageNet CNN achieve good performance on various image recognition tasks, often outperforming traditional hand-crafted features like SIFT [Low04]. Feature extraction plays a crucial role in transfer learning for GANs, which we examine further in Chapter 5.

Finetuning. Initializing a model with pretrained weights often leads to faster convergence during training and can also help to advoid overfitting [JT21]. The pretrained model provides a sensible starting point such that the training on the new task can focus on refining these features rather than learning from random ones. A common technique is to pretrain the backbone of vision models on a large benchmark like ImageNet, then adapt it to the specific vision task [Zop+20]. For example, Girshick et al. [Gir+14] demonstrated that using a CNN pretrained on the ImageNet dataset and then finetuning it on a small object detection dataset led to an improvement in detection accuracy over training the CNN from scratch. For CNNs, a common recipe is to freeze early layers and finetune only the higher layers initially, since lower layers are often universally useful [Yos+14; Lon+15].

In the next section we conclude Chapter 2 with a fundamental discussion about generalization and regularization in representation learning.

2.3 Generalization and Regularization

When training a model for a given task, we are usually interested in its ability to generalize. Generalization is the ability of a model to perform well on unseen data [GBC16]. One way of estimating generalization performance is using a separate test dataset that the model did not see during training. A low error on this dataset then indicates good generalization towards this distribution. The test dataset can then either be a disjunct sample of the training data distribution (in-distribution), or a variation thereof (out-of-distribution) [TIF24]. When we are interested in a model that performs well on variations of its training data distribution, specifically in terms of common perturbations in images and adversarial perturbations that are crafted to fool a model, we call it robust [GSS15; HD19; Cro+21]. We introduce common corruptions and adversarial perturbations in more detail in Chapter 9, where we introduce a dataset evaluating robustness of different image classification network architectures.

Two of the fundamental concepts that can explain conditions necessary for models to generalize well are the bias-variance tradeoff [GBD92] and the No Free Lunch (NFL) [WM97] theorem. We discuss both below.

2.3.1 Overfitting, Underfitting, and the Bias-Variance Tradeoff

Two common problems that practitioners in machine learning are confronted with when training models are overfitting and underfitting. The bias–variance tradeoff [GBD92] is a fundamental concept explaining the balance between these two problems. Here, the assumption is that we train the same model multiple times with a squared loss, each time using a different dataset that is independently sampled from the same underlying data distribution [Bis06]. Evaluating the expected predictions over all of these models, we can decompose the expected loss of this ensemble into [Bis06]:

$$\text{expected loss} = (\text{bias})^2 + \text{variance} + \text{noise},$$

where the noise term is constant for a given dataset. This decomposition describes the balance between two extremes. Bias measures the differences between the expected predictions and the true function, reflecting the error caused by incorrect assumptions or oversimplified models. For example, fitting a linear model to nonlinear data results in systematic off-target predictions, and therefore high bias. Models with high bias underfit the data, achieving high errors on both the training dataset as well as the test dataset. Variance measures how much predictions vary in-between all models on the same input, reflecting the error caused by sensitivity to noise in the training data. High variance corresponds to overfitting, which happens when a model is too complex relative to the amount of training data, so it memorizes the training data (including noise) instead of learning more generalizable features. Because of that, an overfit model has low training error but high test error.

For example, Zhang et al. [Zha+17a] show that deep networks can fit randomly shuffled labels on training images, achieving near-zero training error. Similarly, Kawaguchi, Bengio, and Kaelbling [KBK22] show that a large network trained on only a few hundred images can overfit by memorizing each image. These examples demonstrate how severe overfitting is possible when model complexity far exceeds the informative content of the data.

Bias-Variance Tradeoff. By changing model complexity, we can trade off between bias and variance (see Figure 2.10 (left)). As we increase model complexity, bias tends to decrease but variance tends to increase. The tradeoff is that we cannot simultaneously minimize both bias and

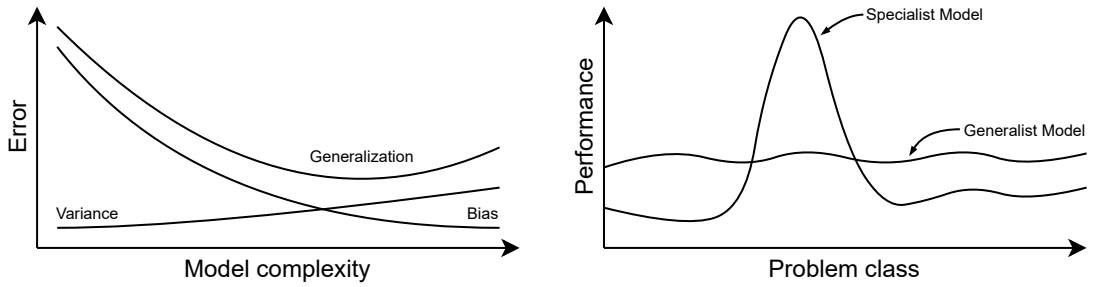


Figure 2.10: Inspired by [LS08]: (**left**) Depiction of the bias-variance tradeoff. The generalization error of a model is decomposed into a variance component and a bias component. The tradeoff between these two can be steered by changing model complexity. (**right**) The NFL theorem states that (i) no algorithm is superior over another algorithm averaged over all problem classes, and hence (ii) if an algorithm performs well on one problem class, it is offset by lower performance on other problem classes. The area under both curves are equal.

variance, since making the model more flexible to reduce bias will typically increase variance, and vice versa. The optimal model complexity is usually the one that achieves the lowest total generalization error (bias + variance) [GBC16].

Regularization Steers the Bias-Variance Tradeoff. One way of adjusting complexity of a model is to change the number of its parameters. Another way is to modify its flexibility via regularization [DFO20]. As Goodfellow, Bengio, and Courville [GBC16] state: »[W]e might find [...] that the best fitting model (in the sense of minimizing generalization error) is a large model that has been regularized appropriately«. Regularization techniques add bias to reduce variance: By favoring simpler models (for example, by adding a penalty or constraint), we accept more bias (the solution may not fit perfectly) in exchange for lower variance (the model is not fitting every training example). By doing so, regularization helps control generalization by managing the trade-off between underfitting and overfitting. For instance, penalizing the magnitude of weights in a neural network via weight decay trades off fitting the training data exactly and the weights being small [GBC16]:

$$\mathcal{L}_{\text{regularized}} = \mathcal{L}_{\text{data}} + \lambda \cdot \sum_i w_i^2,$$

where $\mathcal{L}_{\text{data}}$ is the original loss (e.g., cross-entropy or mean squared error), w_i are the model parameters, and $\lambda > 0$ is a hyperparameter that controls the regularization strength. This is an example for a regularization technique where we incorporate our assumptions about the weights of the model into the loss function. In this case, we encode a preference for small model weights.

While the bias-variance tradeoff highlights the challenge of balancing model complexity to achieve optimal generalization on a specific task, it assumes that the underlying data distribution is fixed and learnable [Bis06]. However, it is not concerned about whether any learning algorithm can perform well on all possible data distributions. The NFL theorem [WM97] addresses this from a theoretical perspective, showing that no learning algorithm can outperform others when averaged over all possible data distributions. In the next section, we discuss the NFL theorem and its implications about regularization further.

2.3.2 No Free Lunch and Inductive Biases

The NFL theorem [WM97] is a fundamental concept in optimization and machine learning that states: *No single learning algorithm performs optimally on all possible data distributions*. In other words, when averaging the performance of any algorithm over all possible problems, every possible algorithm will perform random guessing on average. The implication is that the success of any algorithm is bound to specific assumptions it makes about the problem at hand. The NFL theorem thus highlights the importance of adding problem-specific knowledge to the algorithm. Sterkenburg and Grünwald [SG21] corroborate that learning algorithms in machine learning are also dependent on a model, not only data distributions, and are therefore restricted by the inductive bias of the model. There, the term inductive bias describes any assumption that a learning algorithm has to predict outputs for inputs it has not yet seen [Mit80]. For example, CNNs assume local spatial coherence in images, an assumption that is powerful for vision tasks, but might not help for arbitrary data. In essence, adding tailored assumptions into a model is necessary to perform well on specific data (see Figure 2.10 (right)).

Regularization as Inductive Bias. In the context of this thesis, we define a learning algorithm as a representation-learning neural network and the collection of its hyperparameters and training settings, including the sampled data it is trained on and how the data is sampled. Training this model can be viewed as a search over a hypothesis space to find a function that best fits the training data [Mit97]. We can restrict this hypothesis space by encoding preferences about the function into the model or its training, which modifies the inductive bias of the model [SB14; GBC16]. For instance, with CNNs we assume local connectivity, or when applying weight decay prefer learned parameters having a smaller magnitude. In general, We can encode such preferences via regularization, which Goodfellow, Bengio, and Courville [GBC16] describe as »[...] any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error«. Kukačka, Golkov, and Cremers [KGC17] argue that the definition is too restrictive given that regularization techniques can also reduce the training error, such as when applying weight decay. They instead define regularization as »any supplementary technique that aims at making the model generalize better, i.e. produce better results on the test set.«. In this thesis, we follow their definition. In summary, by encoding our preferences into the model (regularization) we enable it to align the assumptions it has (inductive bias) with the structure of the problem it tries to solve (specialization).

Soft Constraints vs. Hard Constraints. One way to distinguish certain regularization techniques is by how strictly a model is required to follow an imposed constraint. We can differentiate between hard constraints that must strictly be satisfied and soft constraints that are encouraged but not enforced [MSF17; Beu+21; CCL25]. Soft constraints encourage the model towards a certain behavior, for example by adding penalties to the loss function, such as weight decay. Consequently, these constraints are not guaranteed, but they guide the learning process such that the model favors solutions that align more with the encoded structure of the problem. Hard constraints, on the other side, are enforced by design and offer guarantees that an expected behavior is achieved [MSF17; Beu+21; CCL25]. This prunes out nonsensical predictions by, for example, satisfying physical constraints of possible solutions [MSF17].

Explicit vs. Implicit Regularization. Machine learning literature focusing on regularization techniques sometimes distinguishes these by how assumptions are imposed on a model. Loosely summarized, assumptions can be directly encoded (explicitly), for example as penalties, or emerge from network design choices and training dynamics (implicitly). Unfortunately, existing

literature lacks justification about what constitutes explicit vs. implicit regularization, mostly relying on intuitive understanding [NTS15; Zha+17a; WKM20; Zha22]. To the best of our knowledge, the article by Hernández-García and König [HK20] is the only instance of an attempt to define both terms and their relation, albeit still criticized for its vagueness by its reviewers. Here, they define explicit regularization as techniques that affect the *representational capacity* of a network, and implicit regularization as any other effect that affects its *effective capacity*. They follow the definitions given by Goodfellow, Bengio, and Courville [GBC16] for these terms. There, representational capacity is the space of functions that a model is hypothetically able to learn. However, due to limitations of training complex models, the capacity of a model is restricted to its effective capacity. The lack of literature on the topic solidifies the assumption that finding a clear distinction between explicit and implicit regularization is hard, and separating lines become fuzzy. Therefore, we use the terms explicit and implicit according to the sentiment that is reflected by recent literature.

Taxonomy for Regularization. Kukačka, Golkov, and Cremers [KGC17] infer a taxonomy for regularization by observing the objective in typical machine learning problems:

$$\arg\min_{\theta} \frac{1}{|\mathcal{D}|} \sum_{(\mathbf{x}_i, y_i) \in \mathcal{D}} \mathcal{L}_{\text{data}}\left(f_\theta(\mathbf{x}_i), y_i\right) + \mathcal{L}_{\text{penalty}}$$

They divide their taxonomy into elements of the training objective that influence generalization. For this they identify: (a) training data $(\mathbf{x}_i, y_i) \in \mathcal{D}$ for data-based regularization, (b) the model family f for regularization via the network architecture, (c) regularization via the error function $\mathcal{L}_{\text{data}}$, (d) regularization via the penalty term $\mathcal{L}_{\text{penalty}}$, and last (e) the optimization $\arg\min_\theta$ itself for optimization-based regularization. For this thesis, we combine (c) and (d) into loss-modifying regularization, and adopt the rest (mostly) as is. In the next section, Section 2.4, we describe some of the common regularization techniques found in representation learning for computer vision, and we structure the section based on this taxonomy. At some points we deviate from their categorizations and provide justification for doing so.

2.4 Regularization Approaches in Representation Learning

In this section, we provide descriptions for a selection of commonly used methods to regularize representation learning approaches in computer vision, structured in the taxonomy of Kukačka, Golkov, and Cremers [KGC17]. The selection is not complete (as this would overwhelm the scope of this thesis) and intends to give an overview over methods that already exist and are important in computer vision. While some topics are more general, other topics are quite specific (for example multiple discriminators and weighted retraining), because they are key topics of upcoming chapters and intend to provide context for these. Table 2.1 provides an overview of discussed techniques, indicating those that constitute a primary focus in the subsequent chapters. First, we start with optimization-based regularization techniques in Subsection 2.4.1, where we describe implicit regularizing effects of optimization algorithms, early stopping, and projection of model parameters. Subsection 2.4.2 contains architecture-based regularization techniques, which includes the general choice of network architecture, but also more specifically projection layers, dropout, and batch normalization. Further, in Subsection 2.4.3 we describe regularization techniques that modify the loss function, namely penalty terms and adding additional discriminator networks in GANs. Last, in Subsection 2.4.4 we dive into data-based regularization, where we discuss data augmentation and weighted retraining.

Table 2.1: Examples of regularization techniques in the taxonomy by Kukačka, Golkov, and Cremers [KGC17]. We also indicate which parts of this thesis focus on the respective technique.

Optimization-Based	**Architecture-Based**
Subsection 2.4.1: • Optimizer bias • Early stopping • Parameter projection	Subsection 2.4.2: • Architectural design choices • Projection layers • Batch normalization • Dropout
Loss-Modifying	**Data-Based**
Subsection 2.4.3: • Weight norm penalties • Penalty terms (Part I) • Discriminators (Part II)	Subsection 2.4.4: • Data augmentation • Weighted retraining (Part III)

2.4.1 Optimization-Based Regularization

Optimizer Biases. The choice of optimization algorithm influences the training dynamics, and therefore the final performance of the trained model. Potential biases of the optimizer might be reflected in the result, for example when using the commonly used and well-studied optimizer Stochastic Gradient Descent (SGD). There are several works that study implicit regularizing effects arising from optimization with SGD: Goodfellow, Bengio, and Courville [GBC16] argue SGD provides implicit regularization due to noisy gradient updates, especially when combined with small batch sizes. Keskar et al. [Kes+17] claim that SGD with small batch size tends to converge to flatter minima (as compared to sharper minima with larger batch sizes) that often generalize better, while Barrett and Dherin [BD21] attribute this to an implicit gradient penalty of SGD. The effect of this implicit regularization in overparameterized networks is studied by Peleg and Hein [PH24], who show that increasing the width of a network improves generalization, but not increasing its depth. In contrast, Andriushchenko et al. [And+23] demonstrate that the commonly assumed correlation between flatter minima and generalization itself does not necessarily hold when factors such as data distributions, parameter initialization, and function families are taken into account.

In summary, these findings suggest that the choice of optimization algorithm can bias which solution a network converges to. There is evidence that SGD implicitly penalizes large gradients, which in turn biases the optimizer toward flatter minima. Consequently, the choice of SGD as optimizer inherently encodes a preference for such solutions. However, the relationship between flatter minima and generalization remains not well understood.

Early Stopping. Early stopping monitors the performance of the model on a validation dataset during training. When the validation error starts increasing while the training error decreases, training is halted to prevent overfitting [Pre96]. The intuition behind early stopping is that we select the model parameters from an earlier epoch before the model starts to fit to the noise in the training data, thereby restricting its flexibility.

Bishop [Bis06] states that the effective number of degrees of freedom of a network starts small and grows during training, concluding: »Halting training before a minimum of the training error has been reached then represents a way of limiting the effective network complexity.« Goodfellow, Bengio, and Courville [GBC16] highlight that early stopping has the practical advantage of automatically choosing the degree of regularization, whereas penalties such as weight decay require manual tuning of the regularization strength.

In summary, applying early stopping encodes our preference towards a model that performs well on validation data early in training, for which we hope the model generalizes better to unseen data.

Projecting Parameters onto a Feasible Set. One form of regularization that is not considered yet in the taxonomy of Kukačka, Golkov, and Cremers [KGC17] is projecting parameters onto a feasible set. The feasible set $\mathcal{C}$ is the set of all points that satisfy certain constraints we have about our problem. Projection onto the feasible set then means finding the closest point within $\mathcal{C}$ to a potentially arbitrary point outside $\mathcal{C}$ that we are given. Formally, given a feasible set $\mathcal{C} \subseteq \mathbb{R}^n$, the projection operation of a point $\mathbf{x} \notin \mathcal{C}$ onto $\mathcal{C}$ with some distance measure $d(\cdot, \cdot)$ (typically Euclidean) can be defined as [BV04]:

$$\Pi_{\mathcal{C}}(\mathbf{x}) = \arg\min_{\mathbf{z} \in \mathcal{C}} d(\mathbf{z}, \mathbf{x}).$$

This operation ensures that the projected parameters stay within the feasible region, strictly enforcing constraints we have about them. This acts as a form of regularization because it restricts the space of solutions, and thus the flexibility of the network. We categorize imposing such constraints as optimization-based regularization technique, as it affects the training dynamics of the optimizer without modifying the loss, in contrast for example to weight decay.

Projecting model parameters to a feasible set an be used to enforce hard constraints about them, such as non-negativity, boundedness, or unit norm [FNW07; VR22]. Srivastava et al. [Sri+14] show that constraining the parameter norm of a network during training via max-norm regularization is useful when training networks in combination with dropout. For this, parameters are rescaled (projected back onto the feasible set) after each update. In this sense, max-norm regularization can be seen as a hard-constraint version of weight decay, since weight vectors can only grow up to a fixed, specified norm. Another example is weight clipping [ACB17]. Here, after each gradient update during training, all parameters are clipped to lie within a predefined interval, typically $[-\tau, \tau]$ for some constant $\tau > 0$. This procedure ensures that no parameter can grow beyond the specified bounds. One notable use case of weight clipping is in Wasserstein Generative Adversarial Networks (WGANs) [ACB17], where it is used to enforce the Lipschitz continuity condition required for the theoretical foundations of the Wasserstein distance.

In summary, by defining a feasible set that reflects our preference over model parameters, we can directly enforce that the model adheres to these constraints.

2.4.2 Architecture-Based Regularization

Network Architecture Design Choices. According to the universal approximation theorem [Cyb89], fully-connected MLPs can, in principle and under certain conditions, approximate any function given sufficient capacity. While this is a promising result, it does not provide guidance on how to learn such functions efficiently from data. By designing the connectivity of the units in a neural network in specific ways, we can incorporate prior assumptions about the task.

Hence, the architecture of a neural network can already steer the model toward certain classes of solutions that align with these assumptions, potentially improving learning efficiency and generalization. For example, due to weight sharing in the convolutional layers of CNNs, the same filter weights are applied at every image location. This design encodes the assumption that the same features can appear anywhere in an image, resulting in models that are equivariant to translation [GBC16]. In addition to incorporating this prior knowledge, weight sharing also has a regularizing effect by reducing the number of parameters compared to fully-connected networks. As a result, data efficiency is improved [GBC16], provided that the translational equivariance assumption holds.

Similarly, using pooling layers in CNNs (an example was shown in Figure 2.4) can provide approximate translational invariance, for which the exact position of a feature matters less after pooling [GBC16]. This acts as a regularizer by limiting the model to focus on whether a feature is present rather than its precise location, reducing sensitivity to shifts in images. This is particularly helpful for classification networks, where we can encode that the exact location of the object is not relevant for our task.

In summary, designing the network architecture of our model allows us to directly encode preferences about the structure of the input data and the nature of the task, thereby guiding the learning process toward more efficient and generalizable solutions.

Projecting Layer Outputs onto a Feasible Set. In some cases, it is desirable to incorporate hard constraints directly into the network architecture, ensuring that the model adheres to known properties of the problem. For example, Beucler et al. [Beu+21] incorporate constraints as custom layers throughout the network that guarantee the satisfaction of physical conservation laws. This ensures that the model cannot produce physically implausible outputs, regardless of the data. Enforcing such hard constraints via network design choices limits the flexibility of the model.

Another common design is to constrain the output of a network by incorporating a projection as final layer [Che+18; Blo19; PM20; Hua+21; CFL24]. The network produces an unconstrained output (such as a vector of real numbers), and then this output is projected onto a feasible set $\mathcal{C}$, representing the space of valid labels or structures. For example, if the task is to predict a probability distribution, the feasible set $\mathcal{C}$ can be the probability simplex. Projecting an arbitrary score vector $\mathbf{x}$ onto the probability simplex yields a proper probability distribution. Effectively, this is what the softmax layer does implicitly. In more complex structured predictions, the feasible set $\mathcal{C}$ might be defined by a combinatorial structure. In such cases, the prediction by the network can be guaranteed to satisfy constraints by, for example, implementing the projection layer via an additional solver [DTP21], or rounding of relaxed solutions with subsequent repairing of violated constraints [TKD25]. In Chapter 3 we project relaxed solutions provided by the proposed minimum cost multicut solver to integer solutions.

In summary, by projecting the outputs of a model onto a feasible set, we can encode our preference for valid solutions that strictly adhere to problem-specific constraints, thereby improving reliability and applicability of the model in constrained prediction tasks.

Dropout. Dropout [Sri+14] is a regularization technique developed for neural networks, where a subset of activations is set to zero during training. Formally, a network layer f_h at layer t computes the next representations $\mathbf{h}_t$ given representations $\mathbf{h}_{t-1}$ from a previous layer

$$\begin{aligned} \text{without dropout:} \quad & \mathbf{h}_t = f_\mathrm{h}(\mathbf{h}_{t-1}), \text{ and} \\ \text{with dropout:} \quad & \tilde{\mathbf{h}}_t = f_\mathrm{h}(\mathbf{h}_{t-1} \odot \mathbf{m}), \quad \mathbf{m} \sim \text{Bernoulli}(p), \end{aligned}$$

where representations $\tilde{\mathbf{h}}_t$ are now noisy, because $\mathbf{h}_{t-1}$ is multiplied with a binary mask m that is sampled from a Bernoulli distribution given probability p. This effectively creates an ensemble of smaller networks, reducing interdependence among neurons, and can thus improve generalization [Sri+14]. At test time, no units are dropped. Instead, the weights are scaled down to account for the averaging effect.

Batch Normalization. Batch Normalization (BN) [IS15] is primarily introduced to stabilize and accelerate training by normalizing layer inputs $\mathbf{h}$ via:

$$\tilde{\mathbf{h}}_{[i]} = \gamma_{[i]} \cdot \frac{\mathbf{h}_{[i]} - \mathbb{E}[\mathbf{h}_{[i]}]}{\sqrt{\text{Var}[\mathbf{h}_{[i]}]}} + \boldsymbol{\beta}_{[i]},$$

where $\tilde{\mathbf{h}}$ are normalized representations, and γ and $\boldsymbol{\beta}$ are learnable scale and shift parameters. Mean and variance are computed as moving averages during training with mini-batches (for example with SGD). Statistics are kept fixed during inference to enable determinism.

Ioffe and Szegedy [IS15] argue that BN mitigates exploding gradients, because it prevents activations to grow in magnitude, which in turn allows for higher learning rates. Additionally, because BN normalizes based on batch statistics, each mini-batch introduces noise in the estimation of of mean and variance, especially when the batch size is small [GBC16]. This noise can be similar to the effect of dropout, preventing the network from becoming too sensitive to the absolute scale of certain neurons. Ioffe and Szegedy [IS15] observe that BN provides sufficient regularization on its own, making dropout obsolete. Luo, Ren, and Peng [LRP19] note that BN improves generalization in addition to speeding up convergence. Their reasoning is that BN smooths the optimization landscape and also adds slight noise to activations, which together help avoid overfitting.

Categorization of Dropout and Batch Normalization. Kukačka, Golkov, and Cremers [KGC17] categorize dropout and batch normalization as data-based regularization techniques, because these methods transform intermediate representations (in a stochastic sense) within the network. However, they also acknowledge the ambiguity that these can also be seen as part of the network architecture, as they are often referred to as layers. We choose the latter, and describe both techniques in the current section.

2.4.3 Loss-Modifying Regularization

Weight Norm Penalty Terms. Weight norm penalties (such as weight decay) encode a preference for small parameter values into the objective function by penalizing their magnitude. The general form of a regularized loss function is:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{data}}(\theta) + \lambda \cdot \Omega(\theta),$$

where θ are model parameters, $\mathcal{L}_{\text{data}}(\theta)$ is the training objective loss, and $\Omega(\theta)$ is the regularizer whose strength is controlled via hyperparameter λ. Common choices for weight norm penalties are:

$$\begin{aligned} L_1 \text{ Penalty (Lasso):} \quad & \Omega(\theta) = \|\theta\|_1, \text{ and} \\ L_2 \text{ Penalty (weight decay):} \quad & \Omega(\theta) = \|\theta\|_2^2. \end{aligned}$$

Weight decay (L_2) is commonly used during training of neural networks and encourages small parameter values, as the gradients of the penalty shrink the parameters with each update [GBC16].

The L_1 penalty, on the other hand, encourages weights to become exactly zero and therefore promotes sparsity [Tib96]. This is caused by constant gradients, which apply uniform pressure on all weights regardless of their magnitude (in contrast to weight decay, where gradients depend on the magnitude of weights). As such, L_1 regularization can also be used to perform variable selection [Tib96].

Representation-Based Penalty Terms. While weight norm penalties explicitly penalize model parameters, it is also possible to construct penalty terms that involve feature representations of the model. By doing so, we can directly encode preferences we have over features learned by the model. For example, an L_1 penalty on feature representations $\Omega(\mathbf{h}) = \lambda \cdot \|\mathbf{h}\|_1$ again encourages sparsity in the feature representations the model extracts. This is, for example, applied to the code layer of sparse autoencoders [Le+12], where we want the autoencoder to compress input data into a sparse representation.

One can also penalize properties of representations, such as their sensitivity to input changes as done in contractive autoencoders [Rif+11]. These are trained with the following penalty term:

$$\Omega(\mathbf{h}) = \|\nabla_{\mathbf{x}}\mathbf{h}\|_{\mathrm{F}}^2. \tag{2.3}$$

Here, the penalty term is composed of the Frobenius norm $\|\cdot\|_{\mathrm{F}}$ applied to the Jacobian of the output representations of the encoder with respect to the input. This penalty encourages representation $\mathbf{h}$ to only allow small local changes for small perturbations in input $\mathbf{x}$, which in essence means the network learns more robust features that are less sensitive to small input perturbations [JG18].

The idea of penalizing certain structures in representations is widely adopted. For example, Cogswell et al. [Cog+16] penalize the covariance of representations of a layer, and thereby encourages decorrelated features. They claim that such a penalized model is less prone to overfitting and can improve generalization.

Output-Based Penalty Terms. An example of penalizing the output of a network is label smoothing [Sze+16]. This technique modifies the (usually) one-hot encoded ground truth vector by mixing it with a uniform distribution over classes:

$$\tilde{\mathbf{y}} = (1-\alpha)\cdot\mathbf{y} + \alpha\cdot\frac{1}{C}\cdot\mathbf{1}, \tag{2.4}$$

where α is the smoothing parameter, C is the number of classes, and $\mathbf{1} \in \mathbb{R}^C$ is a vector of ones. Considering cross-entropy $\mathrm{CE}(\cdot,\cdot)$ as loss function, we can rewrite the loss of the model by treating $\tilde{\mathbf{y}}$ as targets:

$$\mathcal{L} = \mathrm{CE}(\tilde{\mathbf{y}},\hat{\mathbf{y}}) \overset{(2.4)}{=} (1-\alpha)\cdot\mathrm{CE}(\mathbf{y},\hat{\mathbf{y}}) + \alpha\cdot\mathrm{CE}\left(\frac{1}{C}\cdot\mathbf{1},\hat{\mathbf{y}}\right).$$

This can be interpreted as adding a penalty term that is proportional to the KL divergence between the predicted distribution and a uniform prior [Sze+16]. This encourages the network to avoid overconfident predictions and promotes softer probability distributions. Pereyra et al. [Per+17] extend this idea by explicitly adding an entropy-based penalty term to the loss, showing that both label smoothing and explicit penalty improve generalization. Label smoothing is another instance for which we deviate from the taxonomy by Kukačka, Golkov, and Cremers [KGC17]. While categorized as data augmentation there, we follow the interpretation that label smoothing penalizes the outputs of the model.

In this thesis, we design output-based penalty terms in both chapters of Part I. In both cases, we incorporate discrete constraints defining feasible compositions of the output vector as penalties, encouraging the minimum cost multicut solver in Chapter 3 to produce more feasible solutions, and the edge detection network in Chapter 4 to predict more consistent edgemaps.

Multiple Discriminators in GANs. While the training of GANs usually consists of a single generator that is trained alongside a single discriminator network, an emerging line of research instead uses multiple discriminator networks to improve training stability and generative performance. The core idea is that the generator network is trained by an ensemble of discriminator networks, which has been shown to alleviate common GAN pathologies like mode collapse and discriminator overfitting [Goo+14]. Training with multiple discriminator networks changes the training objective, and therefore acts as loss-modifying regularization technique.

For example Durugkar, Gemp, and Mahadevan [DGM17] extend GAN training to K discriminators and aggregate their feedback (e.g. via weighted average). They show that this stabilizes training using the original minimax GAN loss and also accelerates convergence. Nguyen et al. [Ngu+17] propose using two discriminator networks with complementary objectives. One discriminator encourages the generator to produce samples similar to real data, while the other one rewards generated data that diverges from the real distribution. They show that this scheme is able to capture multiple modes of the data and thereby alleviates mode collapse. Albuquerque et al. [Alb+19] frame the training of GANs with multiple discriminators as multi-objective optimization problem, and achieve a better trade-off between sample quality and training cost compared to naive averaging of multiple discriminator network training signals. They also show that quality and diversity of samples increases with the number of discriminators. Choi and Han [CH22] propose multiple discriminator training, where each discriminator specializes on a subset of the real data distribution, while sharing a common backbone network. Again, this encourages the generator to cover the entire data distribution while mitigating mode collapse.

All these approaches have in common that discriminators are trained simultaneously alongside the generator network. The training loss for the generator is typically a combination (sum, average, or weighted aggregate) of the losses from each discriminator, which means the generator can only be successful by fooling all discriminator networks at the same time. The common motivation behind this approach is to mitigate mode collapse and thereby improve sample diversity. Another effect is that ensembling discriminators provides more robust gradients. Even if one discriminator overfits and its feedback vanishes at some region, another discriminator may still provide a signal.

In Chapter 6, we add an additional discriminator network that acts in the spectral domain in order to encourage the model towards increased fidelity in this domain.

2.4.4 Data-Based Regularization

Data Augmentation. One powerful regularizer in computer vision to improve generalization is data augmentation [GBC16; Hen+20]. By transforming input images in ways that preserve their label, we can effectively multiply the training set size and expose the model to a wider variety of situations. Common augmentations include random crops, horizontal flips, rotations, color jitter, scaling, cutouts, to name just a few [SK19]. With data augmentation, the model cannot simply memorize the training set, because each epoch it is shown altered versions of the training images. Goodfellow, Bengio, and Courville [GBC16] emphasize that »[t]he best way to make a model generalize better is to train it on more data«, and notes that injecting noise or perturbations into inputs is effectively a form of data augmentation. Augmentation can be seen as imposing prior

knowledge that certain transformations should have no influence on the class label, introducing a bias into the model towards which invariances it should learn. However, augmentations must be applied carefully, as semantically altering transformations can degrade performance [SK19].

Modern augmentation strategies learn policies that optimize the selection of transformations applied on a given dataset [Cub+19; Cub+20; Hen+20]. Other lines of augmentations, like mixup [Zha+18a], create virtual training samples composed out of multiple training instances (including the class label).

In summary, data augmentation not only improves generalization by increasing the amount of data, but also encodes our preferences for which input variations the model should learn to be invariant to.

Weighted Retraining. Instead of treating items in the training data set uniformly, we may have tasks where certain data items are more informative than others. In such cases, we may want to assign importance to data, which implicitly tells the model what we care about. Tripp, Daxberger, and Hernández-Lobato [TDH20] showed that weighting training data and retraining a pretrained generative model with the weighted data can be used to optimize its latent space. Intuitively, the latent space of the generative model is optimized in such a way that the model produces training instances that are assigned higher importance with higher probability. Tripp, Daxberger, and Hernández-Lobato [TDH20] argue that weighted retraining corresponds to training with a weighted emirical mean:

$$\mathcal{L}_{\text{weighted}} = \sum_{i=1}^{n} \omega_i \cdot \mathcal{L}_{\text{train}}(\mathbf{y}_i, \hat{\mathbf{y}}_i), \tag{2.5}$$

where ω_i is the weight for training sample i, $\mathbf{y}_i$ are the true labels, and $\hat{\mathbf{y}}_i$ are the predictions. However, in order to employ weighted retraining with SGD, it is implemented by sampling each training item from the dataset according to its assigned weight (with replacement) [TDH20]. In contrast to transforming the content of data items as done in data augmentation, here we transform the *sampling distribution* of the dataset.

Sampling from the dataset in a weighted manner can be interpreted as incorporating a prior belief that certain samples are more informative or valuable. This acts as a form of regularization, because it changes the inductive bias of the model and potentially leads to better generalization on tasks where certain examples are more descriptive of the desired outcome. The taxonomy by Kukačka, Golkov, and Cremers [KGC17] is not covering the case where the way that training data is sampled is modified in such a way. While weighted retraining could be categorized as optimization-based regularization (see Equation 2.5), in essence, we encode a preference over *how we sample from the training dataset*. Hence, we classify it as data-based regularization.

In Part III, we use weighted retraining to perform latent space optimization in the discrete latent space of VQ-VAEs in Chapter 7, and to perform neural architecture search with a generative model in Chapter 8.

2.5 Summary

In this chapter, we introduced key concepts from representation learning, computer vision tasks, and regularization. We motivated the importance of regularization as a tool to encode knowledge in the form of preferences about the features a model learns, or how it learns them. Then, we described different approaches of regularization techniques that are applied in computer vision via the taxonomy introduced by Kukačka, Golkov, and Cremers [KGC17]. This taxonomy illustrates the range of possibilities available to encode our preferences into the model. In Part I we incorporate binary constraints from a combinatorial optimization problem into the training of representation-learning networks. Although these are hard constraints for the underlying integer linear program of the problem, we reformulate them into penalty terms (loss-modifying regularization), hence into soft constraints that the model is encouraged but not enforced to adhere to. However, since we need feasible integer solutions for inference in Chapter 3, we also incorporate a component into the model that projects the relaxed outputs of the proposed minimum cost multicut solver onto integer-valued solutions (architecture-based regularization). In Part II we regularize generative models via methods that match extracted features of generated images with the training data. These also modify the loss of the respective generator and encode our preferences towards certain aspects of quality in the generated images. Lastly, in Part III we discuss methods that employ weighted retraining (data-based regularization), and thereby encode preferences about the latent space of these generative models. Most of the other techniques discussed here, like data augmentation, early stopping, batch normalization, and also optimizer biases, are key components of most modern computer vision approaches. And, although not specifically discussed, these are also applied throughout upcoming chapters. We now proceed with the first part of the thesis.

Part I

Penalize: Regularization with Discrete Constraints

In the first part of this thesis we explore how discrete constraints that define feasible solutions of the minimum cost multicut problem can be used to regularize representation learning approaches. The minimum cost multicut problem is a combinatorial optimization problem in which edges are removed from a graph, and by that, the graph is decomposed into subgraphs. This problem can be defined as integer linear program with cycle consistency constraints. These constraints ensure that for each removed edge the formerly connected nodes are assigned to distinct components. We first incorporate cycle consistency constraints as penalty term in the training of a graph representation learning-based solver for the multicut problem itself in Chapter 3. Here, we demonstrate that the graph representations learned by the regularized network facilitate the feasibility of proposed solutions. Second, in Chapter 4, we incorporate cycle consistency constraints as penalty term into the training of convolutional neural networks in the task of edge detection. Here, these constraints encourage the network to predict edgemaps in which object contours are closed. We show that this leads to cleaner edgemaps, and hence, better image segmentation results.

This page intentionally left blank.

Chapter 3

Representation Learning in Graphs with Discrete Constraints

Contents of This Chapter

Chapter Topic. This chapter is based on Jung and Keuper [JK22]. Here, we investigate how discrete constraints from combinatorial optimization can regularize graph-based representation learning methods. In particular, a graph neural network-based solver for the minimum cost multicut problem is developed. This problem decomposes a graph into subgraphs by removing edges and gives rise to cycle consistency constraints that define the composition of feasible, integer solutions. The introduced approach integrates these constraints, encouraging the network to learn representations that provide feasible solutions. Experiments demonstrate that this regularization enhances the performance and generalization of the learned representations in various segmentation tasks.

Chapter Outline. We first introduce this chapter in Section 3.1. Then, we briefly review the minimum cost multicut problem in Section 3.2, and provide an overview on graph neural networks and their application in combinatorial optimization in Section 3.3. In Section 3.4, we present the proposed approach for solving the minimum cost multicut problem with graph neural networks, including model adaptations and the derivation of the proposed penalty term. Section 3.5 provides an empirical evaluation of the proposed approach, after which we conclude this chapter in Section 3.6.

3.1 Introduction

RECENT years have shown great advances of neural network-based approaches in various application domains from image classification [KSH12] and natural language processing [Vas+17] up to very recent advances in decision logics [Ara+21]. While these successes indicate the importance and potential benefit of learning from data distributions, other domains such as symbolic reasoning or combinatorial optimization are still dominated by classical approaches. Recently, first attempts have been made to address NP-hard combinatorial problems in a learning-based setup [Sel+19; Dai+17b; LCK18; Pra+19]. Specifically, such papers employ (variants of) Message Passing Neural Networks (MPNNs) [Gil+17] defined on graphs [Sca+09; Mic09; KW17] in order to model, for example, the boolean satisfiability of conjunctive normal form formulas (SAT) [Sel+19], or address the travelling salesman problem [Pra+19] (both highly important NP-complete combinatorial problems). These first advances employ the ability of graph convolutional networks to efficiently learn representations of entities in graphs and prove the potential to solve hard combinatorial problems.

In this chapter, we address the Minimum Cost Multicut Problem (MP), also known as the weighted correlation clustering problem [Dem+06; BBC04]. This grouping problem is substantially different from the aforementioned examples as it aims to assign binary edge labels based on a signed edge cost function. Such graph partitioning problems are ubiquitous in practical applications such as image segmentation [SM00; Arb+11; And+11; And+12; Keu+15], motion segmentation [Keu17; KAB15], stereo matching [Kap+15a], inpainting [Kap+15a], object tracking [Keu+20; Ho+20], pose tracking [Ins+16], or entity matching [OB19]. We provide an illustrative example in Section A.1 in Appendix A.

The MP is NP-hard, as well as APX-hard [BBC04], which makes it a particularly challenging subject to explore. Its main difficulty lies in the exponentially growing number of constraints that define feasible solutions, especially whenever non-complete graphs are considered. Established methods solve its binary linear program formulation or linear program relaxations [Kap+15a]. However, deriving optimal solutions is oftentimes intractable for large problem instances. In such cases, heuristic, iterative solvers are used as a remedy [Keu+15]. A significant disadvantage of such methods is that they can not provide gradients that would allow to train downstream tasks in an end-to-end way.

To address this issue, we propose a formulation of the minimum cost multicut problem as an MPNN. While the formulation of the multicut problem as a graph neural network seems natural, most existing MPNN-based approaches are designed to aggregate *node* features, potentially under edge constraints [SK17]. In contrast, instances of the multicut problem are purely defined through their *edge weights*. Graph Convolutional Networks (GCNs) [KW17] rely on diverse node embeddings normalized by the graph Laplacian and an isotropic aggregation function. Yet, edge weights in general, and signed edge weights in particular, are not modelled in standard GCNs. In this chapter, we propose an extension of GCNs and show that the *signed* graph Laplacian can provide sufficiently strong initial node embeddings from signed edge information. This, in conjunction with an anisotropic update function which takes into account signed edge weights, facilitates GCNs to outperform more recent models such as Signed Graph Convolutional Networks (SGCNs) [DMT18], Graph Isomorphic Networks (GINs) [Xu+19] as well as models that inherently handle real-valued edge weights, such as Residual Gated Graph Convolutional Networks (RGGCNs) [JLB19] and Graph Transformer Networks (GTNs) [Shi+21] on the multicut problem.

To facilitate effective training, we consider a polynomial programming formulation of the minimum cost multicut problem to derive a penalty term that encourages the network to issue

valid solutions. Since currently available benchmarks for the minimum cost multicut problem are notoriously small, we propose two synthetic datasets with different statistics, for example w.r.t. the graph connectivity, which we use for training and analysis. We further evaluate our models on the public benchmarks BSDS300 [Mar+01], CREMI [Bei+17], and Knott3D [And+12].

3.2 The Minimum Cost Multicut Problem

The MP [CR93; DL97] is a binary edge labelling problem defined on a graph $G = (V, E)$, where the connectivity is defined by edges $e \in E \subseteq \binom{V}{2}$, i.e. G is not necessarily complete. It allows for the definition of real-valued edge costs $w_e \forall e \in E$. Its solutions decompose G such as to minimize the overall cost. Specifically, the MP can be defined by the following Integer Linear Program (ILP) [CR93]: For a simple, connected graph $G = (V, E)$ and an associated cost function $c\colon E \to \mathbb{R}$, written below is an instance of the multicut problem

$$\min_{\mathbf{y} \in \{0,1\}^{|E|}} \quad c(\mathbf{y}) = \mathbf{y}^T \mathbf{w} = \sum_{e \in E} w_e y_e, \tag{3.1}$$

with $\mathbf{y}$ subject to the linear constraints

$$\forall C \in \text{cycles}(G), \quad \forall e \in C\colon \quad y_e \leq \sum_{e' \in C \setminus \{e\}} y'_e, \tag{3.2}$$

where cycles$(\cdot)$ enumerates all cycles in graph G. The resulting $\mathbf{y}$ is a vector of binary decision variables for each edge. Equation 3.2 defines the cycle inequality constraints and ensures that if an edge is cut between two nodes, there can not be another path in the graph connecting them. Chopra and Rao [CR93] further showed that the facets of the MP can be sufficiently described by cycle inequalities on all chordless cycles of G. The problem in Equation 3.1-3.2 can be reformulated in a more compact way as a Polynomial Program (PP):

$$\min_{\mathbf{y} \in \{0,1\}^{|E|}} \sum_{e \in E} w_e y_e + K \sum_{C \in \text{cc}(G)} \sum_{e \in C} y_e \prod_{e' \in C \setminus \{e\}} (1 - y'_e), \tag{3.3}$$

where K is a sufficiently large penalty constant. The above problem is well behaved for complete graphs where it suffices to consider all cycles of length three and Equation 3.3 becomes a quadratic program. For sparse graphs, sufficient constraints may have arbitrary length $\leq |V|$ and their enumeration might be practically infeasible. Finding an optimal solution is NP-hard and APX-hard [BBC04]. Therefore, exact solvers are intractable for large problem instances. Linear program relaxations as well as primal feasible heuristics have been proposed to overcome this issue, which we briefly review in the following.

Related Work on Multicut Solvers. To solve the ILP from Equation 3.1-3.2, one can use general purpose linear program solvers, like Gurobi [Gur21] or CPLEX, such that optimal solutions might be in reach for small instances if the enumeration of constraints is tractable. However, no guarantees on the runtime can be provided. To mitigate the exponentially growing number of constraints, various cutting-plane [Kap+11; Kim+11; Kim+14] or branch-and-bound [And+12; Kap+15a] algorithms exist. For example, [Kap+11] employ a relaxed version of the ILP in Equation 3.1 without cycle constraints. In each iteration, violated constraints are searched and added to the ILP. This approach provides optimal solutions to formerly intractable instances - yet without any runtime guarantees. Linear program relaxations [Kim+11; Kap+15a; SA17] increase

the tightness of the relaxation, for example using additional constraints, and provide optimality bounds. While such approaches can yield solutions within optimality bounds, their computation time can be slow and the proposed solution can be arbitrarily poor in practice. In contrast, heuristic solvers can provide runtime guarantees and have shown good results in many practical applications. The primal feasible heuristic Kernighan-Lin with Joins (KLj) [Keu+15] iterates over pairs of partitions and computes local moves which allow to escape local optima. Competing approaches have been proposed, for example by Beier et al. [Bei+14] and Kardoost and Keuper [KK18] or Beier et al. [Bei+16]. The highly efficient Greedy Additive Edge Contraction (GAEC) [Keu+15] approach aggregates nodes in a greedy procedure with an $O(|E|\log|E|)$ worst case complexity. While such primal feasible heuristics are highly efficient in practice, they share one important draw-back with ILP solvers that becomes relevant in the learning era: they can not provide gradients that would allow for backpropagation, for example to learn edge weights.

In contrast, a third order conditional random field based on the PP in Equation 3.3 has been proposed by Song et al. [Son+19] and adapted by Jung et al. [Jun+22] (discussed in Chapter 4), which can be optimized in an end-to-end fashion using mean field iterations. This approach expects optimization on complete graphs. Our approach employs MPNNs to overcome this limitation and provides a general purpose end-to-end trainable multicut approach.

3.3 Message Passing Neural Networks

Gilmer et al. [Gil+17] provide a general framework to describe convolutions for graph data spatially as a message-passing scheme. In each convolutional layer, each node is propagating its current node features via edges to all of its neighboring nodes and updates its own features based on the messages it receives. The update is commonly described by an update function

$$\mathbf{h}_u^{(t)} = g^{(t)}\left(\mathbf{h}_u^{(t-1)}, \hat{\sum_{v\in\mathcal{N}(u)}} f^{(t)}\left(\mathbf{h}_u^{(t-1)}, \mathbf{h}_v^{(t-1)}, \mathbf{x}_{v,u}\right)\right), \tag{3.4}$$

where $\mathbf{h}_u^{(t)} \in \mathbb{R}^F$ is the feature representation of node u in layer t with dimensionality F, and $\mathbf{x}_{v,u}$ are edge features. Here, f and g are differentiable functions, and $\hat{\Sigma}$ is a differentiable, permutation invariant aggregation function, mostly sum, max, or mean. Commonly, the message function f and the update function g are parameterized, and apply the same parameters at each location in the graph, similar to weight sharing in Convolutional Neural Networks (CNNs).

Various formulations have been proposed to define g. GCN [KW17] normalizes messages with the graph Laplacian and linearly transforms their sum to update node representations. SGCN [DMT18] aggregates messages depending on the sign of the connectivity and keeps two representations per node, one for balanced paths and one for unbalanced paths. GIN [Xu+19] learns an injective function by defining message aggregation as a sum and learning the update function as an Multilayer Perceptron (MLP). RGGCN [JLB19] computes edge gates to aggregate messages in an anisotropic manner and learns to compute the residuals to the previous representations. Edge conditioned GCNs [SK17] aggregate node features using dynamic weights computed from high-dimensional edge features. GTN [Shi+21] also aggregate messages anisotropically by learning a self-attention model based on the transformer models in NLP [Vas+17]. While the latter three can directly handle real-valued edge weights, all are tailored towards aggregating meaningful node features. In the following, we review recent approaches to employ such models for combinatorial optimization.

MPNNs and Combinatorial Optimization. Recently, MPNNs have been applied to several hard combinatorial optimization problems, such as the minimum vertex cover [LCK18], maximal clique [LCK18], maximal independent set [LCK18], the satisfiability problem [LCK18], and the travelling salesman problem [JLB19]. Their objective is either to learn heuristics such as branch-and-bound variable selection policies for exact or approximate inference [Gas+19; Din+20], or to use attention [VFJ15], reinforcement learning [Bel+17; Dai+17b], or both [Naz+18; KHW19; Ma+20] in an iterative, autoregressive procedure. Joshi, Laurent, and Bresson [JLB19] address the 2D Euclidean travelling salesman problem using the RGGCN model to learn edge representations. Other recent approaches address combinatorial problems by *decoding*, using supervised training such as Chen and Zhang [CZ20]. The proposed approach is related to the work of Joshi, Laurent, and Bresson [JLB19], since we cast the minimum cost multicut problem as a binary edge classification problem that we address using MPNN-based approaches, including RGGCN. We train our model in a supervised way, yet employing a dedicated penalty term which encourages feasible solutions w.r.t. Equation 3.2.

3.4 Multicut Neural Network

We cast the multicut problem into a binary edge classification task, where label $y_{u,v} = 1$ is assigned to an edge (u, v) if it is cut, and $y_{u,v} = 0$ otherwise. The task of the model is to learn a probability distribution $\hat{y}_{u,v} = p(y_{u,v} = 1 \mid G)$ over the edges of a given graph, inferring how likely it is that an edge is cut. Based on these probabilities, we derive a configuration of edge labels, $\mathbf{y} = \{0, 1\}^{|E|}$. In contrast to existing autoregressive MPNN-based models in combinatorial optimization, we derive a solution after a *single forward pass* over the graph to achieve an efficient bound on the runtime of the model. In this scenario, our model can be defined by three functions, i.e., $\mathbf{y} = f_r(f_c(f_e(G, \mathbf{w})))$. First, f_e is the edge representation mapping, assigning meaningful embeddings to each edge in the graph given a multicut problem instance. This function is learned by an MPNN. Second, f_c assigns to every edge its probability to be cut. This function is learned by an MLP. Last, function f_r translates the resulting configuration of edge probabilities to a feasible configuration of edge labels, hence, computes a feasible solution.

Edge Representation Mapping. Given a multicut problem instance $(G, \mathbf{w})$, the edge representation mapping f_e learns to assign meaningful edge embeddings via MPNNs. One specific case of MPNN is GCN [KW17], where the node representation update function is defined as follows:

$$\mathbf{h}_u^{(t)} = g_\theta^{(t)} \left(\mathbf{h}_u^{(t-1)} + \sum_{v \in \mathcal{N}(u)} \mathbf{L}_{[v,u]} \mathbf{h}_v^{(t-1)} \right), \tag{3.5}$$

where $\mathbf{h}_u^{(t)} \in \mathbb{R}^F$ denotes the feature representation of node u in layer t with channel size F. In each layer, node representations of all neighbors of u are aggregated and normalized by $\mathbf{L}_{[v,u]} = 1/\sqrt{\deg(u)\deg(v)}$, where $\mathbf{L} = \tilde{\mathbf{D}}^{1/2}\tilde{\mathbf{A}}\tilde{\mathbf{D}}^{1/2}$ is the normalized graph Laplacian with additional self-loops in the adjacency matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}$ and degree matrix $\tilde{\mathbf{D}}$. Conventionally, $\mathbf{h}_u^{(0)}$ is initialized with node features $\mathbf{x}_u$. Intuitively, we expect normalization with the graph Laplacian to be beneficial in the MP setting, since B1) its eigenvectors encode similarities of nodes within a graph [SM00] and B2) even sparsely connected nodes can be assigned meaningful representations [Arb+11]. However, MP instances consist of real-valued edge-weighted graphs and the normalized graph Laplacian is not defined for negative node degrees.

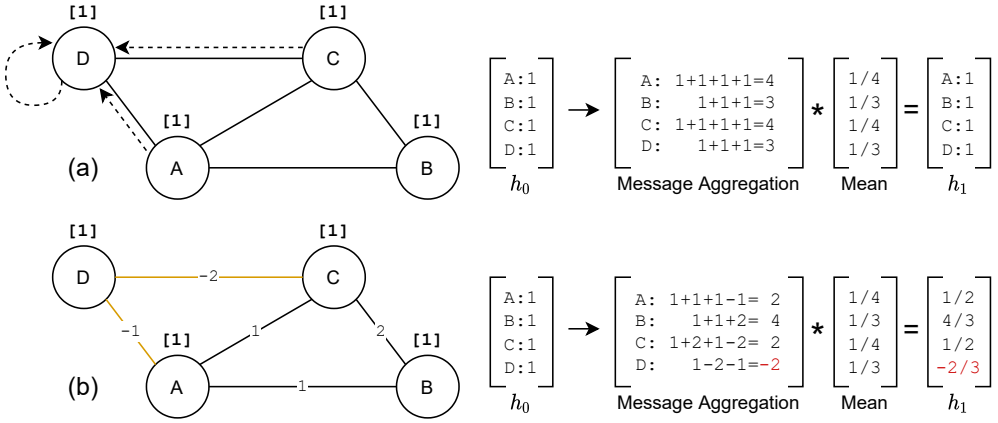


Figure 3.1: Example for message aggregation in an undirected, weighted graph where node features are initialized as $h_0 = 1$. **(a)** Standard message aggregation in an isotropic fashion leads to no meaningful node embeddings, hence $h_1 = h_0$. **(b)** Our proposed method takes edge weights into account leading to anisotropic message aggregation and meaningful node embeddings. A simple decision boundary at $h = 0$ can now partition the graph.

Real-valued Edge Weights. Hence, our first task is to enable negative-valued edge weights in GCN. We can achieve this via the *signed* normalized graph Laplacian [Hou05; Kun+10]:

$$\overline{\mathbf{L}}_{[v,u]} = \left(\overline{\mathbf{D}}^{1/2}\tilde{\mathbf{W}}\overline{\mathbf{D}}^{1/2}\right)_{[v,u]} = w_{v,u} \cdot \left(\overline{\deg}(u)\,\overline{\deg}(v)\right)^{-1/2}, \tag{3.6}$$

where $\tilde{\mathbf{W}}$ is the weighted adjacency matrix and $\overline{\mathbf{D}}$ is the signed node degree matrix with $\overline{\deg}(u) = \sum_{v \in \mathcal{N}(u)} |w_{u,v}|$. Gallier [Gal16] shows that this formulation preserves the desired properties from the graph Laplacian w.r.t. encoding pairwise similarities as well as representation learning on sparsely connected nodes (see B1) and B2) above).

Incorporating Equation 3.6 into Equation 3.5, we get

$$\mathbf{h}_u^{(t)} = g_\theta^{(t)}\left(\mathbf{h}_u^{(t-1)} + \sum_{v \in \mathcal{N}(u)} w_{[v,u]} \cdot \left(\overline{\deg}(u)\,\overline{\deg}(v)\right)^{-1/2} \cdot \mathbf{h}_v^{(t-1)}\right). \tag{3.7}$$

Here, we can observe two new terms. First, each message is weighted by the edge weight $w_{v,u}$ between two nodes enabling an anisotropic message-passing scheme. Figure 3.1 motivates why this is necessary. While Xu et al. [Xu+19] show that Graph Neural Networks (GNNs) with mean aggregation have theoretical limitations, they also note that these limitations vanish in scenarios where node features are diverse. Additionally, Xu et al. [Xu+19] only consider the case where neighboring nodes are aggregated in an isotropic fashion. As we show here, diverse node features are not necessary when messages are aggregated in the anisotropic fashion we propose. The resulting node representations enable distinguishing nodes in the graph despite the lack of meaningful node features. This is important in our case, since the multicut problem does not provide node features. Second, we are now able to normalize messages via the Laplacian in real-valued graphs. The normalization acts stronger on messages that are sent to or from nodes whose adjacent edges have weights with large magnitudes. Large magnitudes on the edges usually indicate a confident decision towards joining (for positive weights) or cutting (negative weights). Thus, the normalization will allow nodes with less confident edge cues to converge to

Table 3.1: Modifications (blue) to MPNNs to account for signed edge weights.

Network	Modified Update Function
GCN	$\mathbf{h}_u^{(t)} = g_\theta^{(t)}\left(\mathbf{h}_u^{(t-1)} + \sum_{v\in\mathcal{N}(u)} w_{v,u}\cdot\left(\overline{\deg}(u)\,\overline{\deg}(v)\right)^{-1/2}\cdot\mathbf{h}_v^{(t-1)}\right)$
SGCN	$\mathbf{h}_u^{B(t)} = g_\theta^{B(t)}\left(\mathbf{h}_u^{B(t-1)} + \sum_{v\in\mathcal{N}^+(u)}\frac{w_{v,u}\cdot\mathbf{h}_v^{B(t-1)}}{\lvert\mathcal{N}^+(u)\rvert} + \sum_{v\in\mathcal{N}^-(u)}\frac{w_{v,u}\cdot\mathbf{h}_v^{U(t-1)}}{\lvert\mathcal{N}^-(u)\rvert}\right)$
	$\mathbf{h}_u^{U(t)} = g_\theta^{U(t)}\left(\mathbf{h}_u^{U(t-1)} + \sum_{v\in\mathcal{N}^+(u)}\frac{w_{v,u}\cdot\mathbf{h}_v^{U(t-1)}}{\lvert\mathcal{N}^+(u)\rvert} + \sum_{v\in\mathcal{N}^-(u)}\frac{w_{v,u}\cdot\mathbf{h}_v^{B(t-1)}}{\lvert\mathcal{N}^-(u)\rvert}\right)$
GIN	$\mathbf{h}_u^{(t)} = g_\theta^{(t)}\left(\left(1+\epsilon^{(t)}\right)\cdot\mathbf{h}_u^{(t-1)} + \sum_{v\in\mathcal{N}(u)} w_{v,u}\cdot\mathbf{h}_v^{(t-1)}\right)$

a meaningful embedding while, without such normalization, the network would notoriously focus on embedding nodes with strong edge cues, i.e. on easy decisions.

In addition to GCN [KW17], we extend SGCN [DMT18] and GIN [Xu+19] in a similar way to enable these to handle real-valued edge weights too. Table 3.1 shows our modifications to update functions of MPNNs we consider in this chapter that cannot handle real-valued edge weights by default.

Node Features. Conventionally, node representations at timestep 0, $\mathbf{h}_u^{(0)}$, are initialized with node features $\mathbf{x}_u$. However, multicut instances describe the magnitude of similarity or dissimilarity between two items via edge weights and provide no node features. Therefore, we initialize node representations with a two-dimensional vector of node degrees as:

$$\mathbf{x}_u = \left(\sum_{v\in\mathcal{N}^+(u)} w_{u,v},\ \sum_{v\in\mathcal{N}^-(u)} w_{u,v}\right), \tag{3.8}$$

where $\mathcal{N}^+(u)$ is the set of neighboring nodes of u connected via positive edges, and $\mathcal{N}^-(u)$ is the set of neighboring nodes of u connected via negative edges.

Node-to-Edge Representation Mapping. To map two node representations to an edge representation, we use concatenation $\mathbf{h}_{u,v} = f_e(\mathbf{h}_u, \mathbf{h}_v) = \binom{\mathbf{h}_u}{\mathbf{h}_v} \in \mathbb{R}^{2\cdot F}$, where $\mathbf{h}_{u,v}$ is the representation of edge (u,v) and F is the dimensionality of node embeddings. Since we consider undirected graphs, the order of concatenation is ambiguous. Therefore, we generate two representations for each edge, one for each direction. This doubles the number of edges to be classified. The final classification is then averaged over both representations.

Edge Classification. We learn edge classification function f_c via an MLP that computes likelihoods $\hat{\mathbf{y}} \in [0,1]^{|E|}$ for each edge in graph G, expressing the confidence whether an edge should be cut. A binary solution $\mathbf{y} \in \{0,1\}^{|E|}$ is retrieved by thresholding the likelihoods at 0.5.

Projection to Feasible Solution. Since there is no strict guarantee that the edge label configuration $\mathbf{y}$ is feasible w.r.t. Equation 3.2, we postprocess $\mathbf{y}$ to *round* it to a feasible solution via a heuristical mapping f_r. For this, we compute a connected component labelling on G after removing cut edges from E and reinstate removed edges for which both corresponding nodes remain within the same component. For efficiency, we implement the labelling as a message-passing layer and can therefore assign cluster identifications to each node efficiently on the GPU.

Cycle Consistency Loss. Since we cast the multicut problem to a binary edge labelling problem, we can formulate a supervised training process that minimizes the Binary Cross Entropy (BCE) loss w.r.t. the optimal solution $\tilde{\mathbf{y}}$, which we denote $\mathcal{L}_{\text{BCE}}$. The BCE loss encodes feasibility only implicitly by comparison to the optimal solution. To explicitly learn feasible solutions, we take recourse to the PP formulation of the multicut problem in Equation 3.3 and formulate a *feasibility penalty term* that we denote Cycle Consistency Loss (CCL):

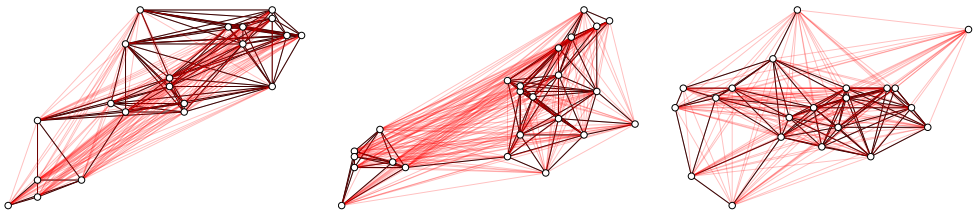
$$\mathcal{L}_{\text{CCL}} = \alpha \cdot \sum_{C \in \text{cc}(G,l)} \sum_{e \in C} \hat{y}_e \prod_{e' \in C \setminus \{e\}} (1 - \hat{y}'_e), \tag{3.9}$$

where α is a hyperparameter, balancing BCE and CCL, and $\text{cc}(G,l)$ is a function that returns all chordless cycles in G of length at most l. The CCL term effectively penalizes infeasible edge label configurations during training. It adds a penalty of at most α for each chordless cycle that is only cut once. In practice, we only consider chordless cycles of maximum length l, and we only consider a cycle if e is cut, hence $\hat{y}_e \geq 0.5$. This is necessary to ensure practicable training runtimes. The total training loss is given by $\mathcal{L} = \mathcal{L}_{\text{BCE}} + \mathcal{L}_{\text{CCL}}$.

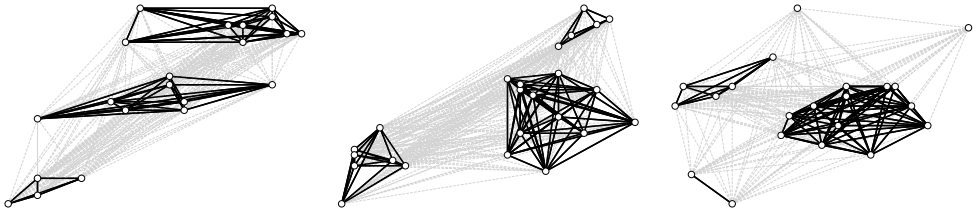
3.4.1 Training Datasets

While the multicut problem is ubiquitous in many real world applications, the amount of available annotated problem instances is scarce and domain specific. Therefore, in order to train and test a general purpose model, we generated two synthetic datasets, *IrisMP* and *RandomMP*. Both have complementary connectivity statistics (see Section A.2 in Appendix A) of 22 000 multicut instances each. In the following, we describe their generation process in more detail.

IrisMP. The first dataset we generated, IrisMP, consists of multicut problem instances on complete graphs based on the Iris flower dataset [Fis36]. This dataset is a well-known multivariate dataset containing 4 different measurements, namely the width and length of sepal and petal for 3 different species of flowers. For each species the dataset contains 50 samples. For each graph we drew 2 dimensions uniformly at random, and then uniformly drew 16 to 24 data points that we used as nodes. We connected all nodes and computed edge weights for each connection. Edge weights are computed in three steps. First, we computed the L_2 distance between two nodes. Then we used a Gaussian kernel with $\sigma = 0.6$ to convert the distances into similarity measures. Since the resulting similarity is in $[0,1]$, we applied the logit function to retrieve positive as well as negative edge weights. Since all graphs are fully connected it is sufficient for this dataset to only consider triangles to ensure cycle consistency. Since the number of edges increases quadratically with the number of nodes in complete graphs, we kept the number of drawn nodes small. IrisMP instances consist of 20 nodes on average. The resulting dataset

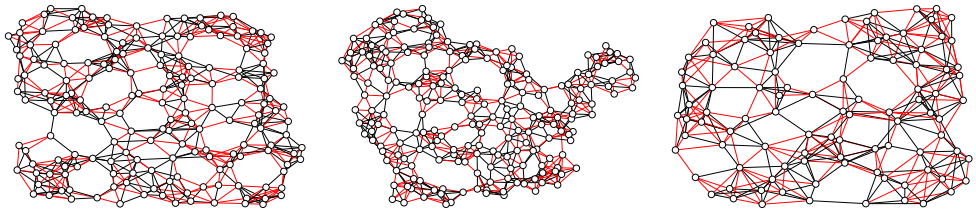


(a) Graph 1. (b) Graph 2. (c) Graph 3.

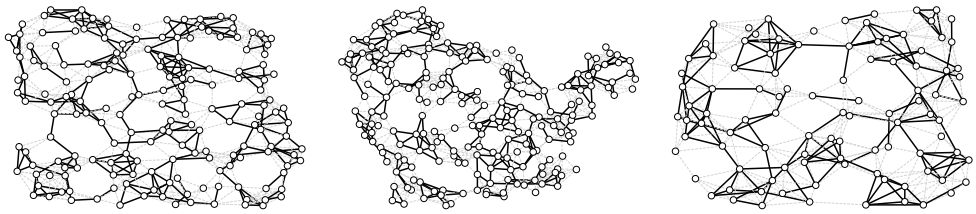


(d) OPT 1. (e) OPT 2. (f) OPT 3.

Figure 3.2: Samples of the IrisMP dataset. **(a)-(c)** Depict problem instances, where red edges have negative weights, and black edges have positive weights. **(d)-(f)** Depict optimal solutions. Gray edges are cut, and black edges retained.



(a) Graph 1. (b) Graph 2. (c) Graph 3.



(d) OPT 1. (e) OPT 2. (f) OPT 3.

Figure 3.3: Samples of the RandomMP dataset. **(a)-(c)** Depict problem instances, where red edges have negative weights, and black edges have positive weights. **(d)-(f)** Depict optimal solutions. Gray edges are cut, and black edges retained.

contains 20 000 instances for training, and 1000 graphs each for validation and test splits. Three graphs with their respective optimal solutions are depicted in Figure 3.2.

RandomMP. To complement IrisMP, we generated a second dataset that contains sparse but larger problem instances in terms of the number of nodes, called RandomMP. To generate this dataset, we employed the following procedure. First, we sampled the number of nodes from a normal distribution with $\mu = 180$ and $\sigma = 30$. Then, for each node, we sampled its coordinates on a 2D plane uniformly in $[0, 1]$ for each coordinate. We connected nodes based on the k nearest neighbors, where k is drawn from a normal distribution with $\mu = 6$ and $\sigma = 2$. However, we constrained the minimum number of neighbors of each node to 1 so that the graph is connected. We computed edge weights based on the L_2 distance on the plane. Then, we subtracted the median from all edge weights to achieve an approximate distribution of 50% positive and 50% negative connections. Again, we generated a training dataset with 20 000 instances and splits of 1000 for validation and test. Examples are shown in Figure 3.3.

3.4.2 Test Datasets

We evaluate our models on three segmentation benchmarks. First, a graph-based image segmentation dataset [And+11] based on the Berkeley Segmentation Dataset (BSDS300) [Mar+01] consisting of 100 test instances. Second, a graph-based volume segmentation dataset (Knott3D) [And+12] containing 24 volumes. And last, 3 additional test instances based on the challenge on Circuit Reconstruction from Electron Microscopy Images (CREMI) [Bei+17] that contains volumes of electron microscopy images of fly brains. BSDS300 and Knott3D instances are available as part of a benchmark containing discrete energy minimization problems, called OpenGM [Kap+15a]. We provide statistics for each of these datasets in Section A.3 in Appendix A.

3.5 Experiments

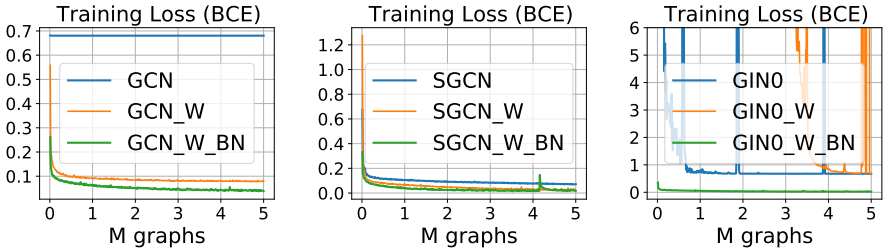
First, in Subsection 3.5.1 we ablate our proposed modifications to the message-passing update function of GCN as well as the update functions of models described in Table 3.1. Then, in Subsection 3.5.2 we ablate the impact of the number of convolutional layers, and in Subsection 3.5.3 we analyze the impact of our penalty term CCL and its hyperparameter α. Last, in Subsection 3.5.4 we evaluate all models trained on IrisMP and RandomMP on different test datasets and provide runtime as well as objective value evaluations, where we compare the proposed GCN to GIN and SGCN-based, edge-weight enabled models as well as to RGGCN [JLB19] and GTN [Shi+21]. Throughout our experiments (if not stated otherwise), we use the *optimal objective ratio* as the performance metric achieved by a model (higher is better), which we denote as follows:

$$\text{optimal objective ratio} = \max(0, c(\mathbf{y})/c(\tilde{\mathbf{y}}) \in [0, 1],$$

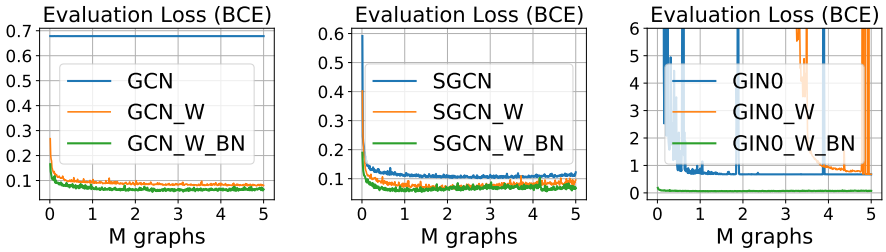
where $\tilde{\mathbf{y}}$ is the optimal solution of a given multicut problem. Here, we set model solutions to 0 if they would incur any positive cost, since such unfavorable edge configurations can always be avoided by not cutting any edge.

Table 3.2: Ablation study with GCN [KW17] trained on IrisMP without CCL. Additional comparison to vanilla versions of GIN [Xu+19], and MPNN [Gil+17]. We report performance on test data in terms of optimal objective ratio ↑.

Variant	IrisMP	RandMP	BSDS300	CREMI	Knott
GCN	Not applicable: Laplacian may not exist.				
- Laplacian	0.41	0.18	0.00	0.49	0.00
+ edge weights	0.95	0.18	0.40	0.57	0.19
+ signed norm.	**0.96**	**0.67**	**0.75**	**0.74**	**0.68**
= GCN_W	**0.96**	**0.67**	**0.75**	**0.74**	**0.68**
- edge weights	0.64	0.05	0.00	0.48	0.00
GIN0	0.41	0.04	0.07	0.48	0.00
MPNN	0.93	0.45	0.48	0.49	0.06



(a) GCN training loss. (b) SGCN training loss. (c) GIN training loss.



(d) GCN evaluation loss. (e) SGCN evaluation loss. (f) GIN evaluation loss.

Figure 3.4: Comparison of the training **(a)-(c)** and evaluation **(d)-(f)** losses of the networks with (indicated by *_W*) and without modified update functions, as well as with batch normalization (indicated by *_W_BN*).

3.5.1 Ablation Studies on Update Function Modifications

First, we determine the impact of each adjustment to the GCN update function. In Table 3.2 we show the results of this ablation study. While vanilla GCN is not applicable in the MP setting, simply removing the Laplacian from Equation 3.5 provides a first baseline. We observe that adding edge weights to Equation 3.5 improves the performance on the test split of the training data substantially. However, the model is not able to generalize to different graph statistics. By adding the signed normalization term we arrive at Equation 3.7, achieving improved generalizability. Removing edge weights from Equation 3.7 deteriorates performance and generalizability. Thus both changes are necessary to enable GCN in the MP setting.

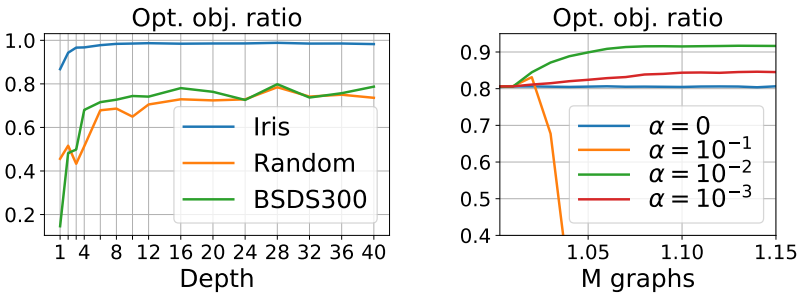
In Figure 3.4 we show the training and evaluation loss curves of training multicut networks with the modified update functions (see Table 3.1) in comparison to their vanilla versions, as well as to adding batch normalization to the modified versions. We trained on IrisMP with 12 message-passing iterations, set the dimensionality of node embeddings to 128, and performed edge classification with an MLP that consists of 2 hiddens layers with 256 neurons each. No CCL was applied. Optimization was performed with Adam [KB15] (0.001 learning rate, $5 \cdot 10^{-4}$ weight decay, $(0.9, 0.999)$ betas) and a batch size of 200. We see again that edge weighting is necessary for GCN to be able to learn meaningful edge representations in our setting. In fact, the original GCN is not able to provide any meaningful features for the edge classification network. Additionally, while vanilla SGCN is able to learn meaningful edge representations, we can still improve these by adding edge weights into their update function. For both networks, GCN as well as SGCN, adding batch normalization improves performance further. For GIN on the other hand, additional regularization via batch normalization is necessary to stably train the network. We conclude from this ablation study that our modifications to the update functions of these MPNNs are favorable in our MP setting. Additionally, regularizing these networks via batch normalization additionally improves performance and training stability.

3.5.2 Ablation Study on Network Size

Next, we evaluate the effect of depth of the GCN model when trained on the IrisMP dataset and evaluated on IrisMP, RandomMP, as well as BSDS300. Figure 3.5 (a) shows the results after varying the depth in increasing step sizes up to a depth of 40. The results suggest that increasing the depth improves the objective value up to a certain point. In the case of IrisMP graphs with diameter 1 and lengths of chordless cycles of at most 3, increasing the depth beyond 10 has no obvious effect. This is an important observation, because [LHW18] raise concerns that GCN models can suffer from over-smoothing such that learned representations might become indistinguishable. Our modifications to the update function of GCN seem to mitigate this issue, as additional depth has no deteriorating effect in our setting.

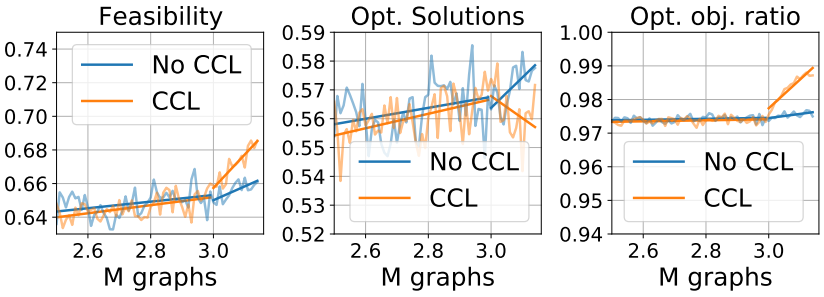
3.5.3 Ablation Study of the Cycle Consistency Penalty Term

Here, we evaluate the effect of applying the penalty term from Equation 3.9 by comparing models where CCL is applied after $3M$ instances to models solely trained without CCL. Figure 3.6 (a) and Figure 3.6 (b) show the progress of the ratio of feasible solutions and ratio of optimal solutions found during training. As soon as CCL is applied, the ratio of feasible solutions increases while the ratio of optimal solutions decreases. Hence, CCL induces a trade-off between finding feasible and optimal solutions, where the model is forced to find feasible solutions to avoid the penalty, and as a consequence, settles for suboptimal relaxated solutions. However, the objective



(a) Ablation of depth. (b) Ablation of hyperparameter α.

Figure 3.5: **(a)** Results in terms of optimal objective ratio on the evaluation data of IrisMP, RandomMP, and BSDS300 when training GCN_W_BN with varying depths on IrisMP. **(b)** Training progress of the optimal objective ratio for different values of α when training GCN_W_BN on RandomMP and applying CCL after $1M$ instances.



(a) Feasibility. (b) Optimality. (c) Objective.

Figure 3.6: **(a)** Ratio of feasible solutions before rounding, **(b)** ratio of optimal solutions, and **(c)** optimal objective ratio, for GCN_W_BN on IrisMP, applying CCL after $3M$ instances.

value after rounding improves (see Figure 3.6 (c)), which is most relevant because these values correspond to feasible solutions. This indicates that the model's upper bound on the optimal energy is higher while the relaxation is tighter when CCL is employed. Additionally, we ablate the effect of the hyperparameter α, tuning the strength of the penalty term (see Figure 3.5 (b)). Here, we set $\alpha \in \{0, 10^{-1}, 10^{-2}, 10^{-3}\}$. From this graph we derive two conclusions. First, adding our penalty term improves the objective value (after rounding) in comparison to not adding it (setting $\alpha = 0$). Second, α is a hyperparameter that has to be chosen carefully, since increasing it further a certain point deteriorates performance (in our case, $\alpha = 10^{-1}$ is too strong).

3.5.4 Evaluation on Test Data

We train the proposed MPNN-based solvers with (adapted) GCN, GIN, SGCN, RGGCN and GTN backbones in different settings, where we uniformly set the node representation dimensionality to 128. We set the depth of the MPNN to 12 for IrisMP and 20 for RandomMP. CCL is applied with $\alpha \in \{0, 10^{-2}, 10^{-3}\}$ and chordless cycle length up to 8. All of our experiments are conducted on MEGWARE Gigabyte G291-Z20 servers with NVIDIA Quadro RTX 8000 GPUs. Training curves of training runs on RandomMP are provided in Section A.4 in Appendix A.

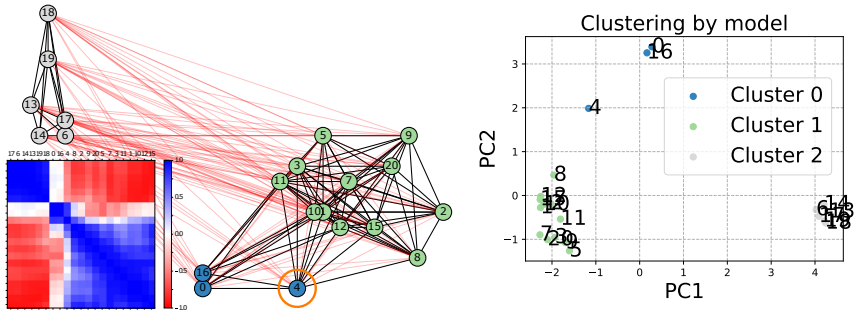
Results on Benchmarks. In Table 3.3, we show the results on all test datasets of the best models based on the evaluation objective value after rounding, and thereby compare models trained on IrisMP and models trained on RandomMP. In general, sparser problems (RandomMP and established test datasets) are harder for the solvers to generalize to. This is likely due to the longer chordless cycles that the model needs to consider to ensure feasibility. Overall, our GCN-based model provides the best generalizability over all test datasets both when trained on IrisMP and RandomMP. We compare the GNN-based solvers to different baselines. First, we train logistic regressions (LR) and MLPs as edge classifiers directly on the training data by concatenating node features and edge weights. All our learned models outperform these baselines substantially. This indicates that MPNNs provide meaningful topological information to the edge classifier that facilitates solving MP instances. Second, we compare against Branch & Cut Linear Program (LP) and ILP solvers as well as GAEC. In terms of objective value, MPNN-based solvers are on par with heuristics and LP solvers on complete graphs, even when trained on sparse graphs. On general graphs, ILP solvers and GAEC issue lower energies, and, as expected, training on complete graphs does not generalize well to sparse graphs. However, the wall-clock runtime comparison shows that MPNN-based solvers are faster than ILP and LP solvers by an order of 10^3. They are also substantially faster than the fast and greedy GAEC heuristic. We further compared to a time-constrained version of GAEC, where we set the available time budget to the runtime of the MPNN-based solver. The result shows that the trade-off between smaller energies and smaller runtime is in favor of the MPNN-based solver. We report additional experiments for our proposed MPNN-based model on domain specific training and show that task specific priors can be learned efficiently from only a few training samples in Section A.5 in Appendix A.

Table 3.3: Results on the test datasets. We compare different GNN variants, heuristics (GAEC) [Keu+15], LP-solver [Kap+15a], and ILP-solver [Kap+15a]. The performance is evaluated as optimal objective ratio ↑ and is averaged over all datasets via harmonic mean to account for generalizability. The last column shows the total runtime ↓ over all datasets in milliseconds. OOM indicates insufficient memory. OOT indicates no termination within 24hrs. Neither OOM nor OOT are considered in the runtime (marked with *).

		Solver				**Test Dataset**						**Runtime [s]**	
		Variant	Depth	α	l	IrisMP	RandomMP	BSDS300	CREMI	Knott	h.mean	forward	total
Proposed learned solvers	**IrisMP**	GCN_W_BN	12	0.001	3	0.9834	0.7188	**0.8912**	**0.7255**	**0.6902**	**0.7865**	0.5	4.4
		GINO_W_BN	12	0.01	3	**0.9905**	**0.7387**	0.8474	0.5464	0.0000	0.0000	0.0	4.0
		Signed_W_BN	12	0.01	3	0.9878	0.2526	0.6451	0.5154	0.3808	0.4510	1.3	5.3
		RGGCN_HE	12	0.01	3	0.7976	0.1449	0.4655	0.1544	0.1735	0.2218	0.1	4.1
		GT	12	0.001	3	0.7940	0.2964	0.6360	0.4037	0.6038	0.4836	0.1	4.0
		LR				0.6769	0.1118	0.6824	0.2689	0.0366	0.1164		N/A
		MLP				0.6626	0.3127	0.7139	0.2789	0.1493	0.3051		N/A
	RandomMP	GCN_W_BN	20	0.01	8	**0.9762**	**0.9041**	**0.9204**	0.8440	**0.7870**	**0.8815**	0.9	4.8
		GINO_W_BN	20	0.01	8	0.9528	0.8693	0.9109	0.4812	0.0000	0.0000	0.0	4.0
		Signed_W_BN	20	0.01	8	0.9709	0.8695	0.8825	0.4653	0.6408	0.7120	2.3	6.3
		RGGCN_HE	20	0.01	8	0.9703	0.8787	0.8352	0.5593	OOM	-	0.1	2.7*
		LR				0.8035	0.3938	0.7958	**0.9260**	0.7335	0.6681		N/A
		MLP				0.8985	0.3099	0.6804	0.4845	0.1517	0.3457		N/A
		GAEC				0.9836	0.9780	0.9997	0.9958	0.9968	0.9907		23.2
		Time-bounded GAEC				0.3642	0.0034	0.0000	0.1516	0.0000	0.0000		6.3
		LP solver				0.9882	0.9525	0.9979	0.9998	OOT	-		31 918.8*
		ILP solver				1.0000	1.0000	1.0000	1.0000	1.0000	1.0000		24 361.2

Table 3.4: Wall-clock runtime ↓ and objective values ↓ of MPNN-based solver vs. GAEC, LP and ILP on a growing, randomly-generated graph. OOT indicates no termination within 24hrs.

Nodes	**GAEC**		**LP**		**ILP**		**GCN_W_BN**	
$\lvert V\rvert$	[ms]	Objective	[ms]	Objective	[ms]	Objective	[ms]	Objective
10^1	**0**	−29	6	−24	11	−30	29	−29
10^2	**4**	−327	191	−246	273	−330	26	−276
10^3	**24**	−3051	6585	−2970	1299	−3093	29	−2643
10^4	228	−32 264	688 851	−31 531	18 604	−32 682	**78**	−27 552
10^5	2534	−323 189	OOT		2 171 134	−328 224	**557**	−269 122
10^6	35 181	−3 401 783	OOT		OOT		**8713**	−2 182 589



(a) Clustered nodes. (b) Node embeddings.

Figure 3.7: **(a)** Node clustering of the proposed GCN_W_BN on a complete graph ($c(\hat{\mathbf{y}}) = -220.6$) from IrisMP and the ordered cosine similarity between all learned node embeddings. **(b)** The first two principal components for each node embedding of (a). Node 4 (circled) is part of the green cluster in the optimal solution ($c(\tilde{\mathbf{y}}) = -222.9$). The closeness of both solutions is reflected in the embedding.

Scalability. Next, we conduct a scalability study on random graphs with an increasing number of nodes, generated according to the RandomMP dataset generation process. Results are shown in Table 3.4. While GAEC is fast for small graphs, the MPNN-based solver scales better and returns solutions substantially faster for larger graphs. LP and ILP solvers are not able to provide solutions within 24hrs for larger instances. It is noteworthy that MPNN-based solvers spend 75-99% of their runtime rounding the solutions. Hence, MPNN-based solvers are already more scalable and still have a large potential for improvement in this regard, while GAEC and LP/ILP solvers are already highly optimized for runtime.

Meaningful Embeddings. In Figure 3.7 we visualize the node embedding space given by our best performing model on an IrisMP instance. Plotting the cosine similarity between all nodes reflects the resulting clusters. This shows that the model is able to distinguish nodes based on their connectivity. We show further examples in Section A.6 in Appendix A.

3.6 Conclusion and Outlook

Summary. In this chapter, we addressed the minimum cost multicut problem using feed forward MPNN. To this end, we provided appropriate model and training loss modifications. Our experiments on two synthetic and two real datasets with various GCN architectures showed that the proposed approach provides highly efficient solutions even to large instances and scales better than highly optimized primal feasible heuristics (GAEC), while providing competitive energies. Another significant advantage of our learning-based approach is the ability to provide gradients for downstream tasks, which we assume will inherently improve inferred solutions.

Feasibility and Optimality Guarantees. The proposed regularization term (CCL) is a soft constraint encouraging feasibility. In its current design, it cannot offer guarantees in terms of feasibility or formal bounds in terms of an optimality gap. To ensure feasibility of solutions, proposed compositions of decision variables are rounded. Possible future research could focus on developing methods that provide guarantees for feasible solutions (without rounding) and solution quality. Several follow-up works proposed methods in this context. For example, Abbas and Swoboda [AS24] integrate representation learning with an MPNN into a provably correct solver. This creates an end-to-end trainable framework that maintains formal guarantees like dual feasibility and monotonic lower bounds. Another example is Li et al. [Li+24], who combine reinforcement learning with an MPNN to solve multicut instances. They ensure feasible solutions by contracting partial solutions in an autoregressive manner. Our method could be applied in a similar way, where edges that our model is most confident about are iteratively contracted. On the downside of this extension, autoregressive edge contractions require forward passes for each edge contraction iteration to recompute node and edge representations. This introduces an additional trade-off between feasibility and efficiency.

Computational Cost of Cycle Consistency Constraints. In general graphs, enumerating all chordless cycles can become intractable, as their number can grow exponentially with the number of nodes in the graph. Consequently, our proposed regularizer CCL shares the same limitation in terms of efficiency as other MP solvers that consider cycle consistency constraints. As a first remedy, we limited the length of considered cycles and precomputed cycles up to this length to facilitate training. Nevertheless, we observed an increase in training runtime by a factor of 100 when applying CCL. The large number of additional (and output-based) terms added to the loss function increases the computational graph substantially, and therefore the computational

time of backpropagating through this graph. As a second remedy, we only gradually introduced these penalty terms into the loss function toward the end of training. Future research could try to improve training efficiency by carefully selecting which cycle constraints to add to the loss function (instead of all of them). For example, prior works have shown that selecting top-k [Yan+19; Sin+20] of potential loss terms can improve performance. In a similar fashion, we could incorporate such a selection strategy based on the confidence of edge removals for violated constraints. For example, by only adding the top-k most confidently violated constraints, as well as the top-k least confidently non-violated constraints.

Tightness of Constraint Relaxations. While we already saw improvements in terms of feasibility and objective value of predicted solutions when applying CCL (see Figure 3.6), it can be speculated that making the LP relaxation of our approach tighter could improve these results further. Hence, future research could also investigate how to incorporate additional tightening constraints, for example wheel and bicycle inequalities [CR93], into the regularizer. Swoboda and Andres [SA17] observe that odd-wheel constraints can tighten the relaxation, albeit in their case the tightness decreases when the sparsity of the graph increases. We believe this is a promising direction, as it could also be combined with the selection strategy suggested above. These additional constraints could help keep the relaxations tight in the face of missing cycle constraints.

Chapter 4
Edge Detection with Discrete Constraints

Contents of This Chapter

Chapter Topic. This chapter is based on Jung et al. [Jun+22]. Similar as in Chapter 3, here we adapt cycle consistency constraints from the minimum cost multicut problem and incorporate them as penalty term into the training of convolutional neural networks for edge detection. In the context of images, graph cycles are created by connecting three neighboring pixels (nodes) forming a triangle. The resulting constraints penalize predicted image edges that pass through only one edge of these triangles, hence are discontinued. These penalties encourage the network to predict closed object contours. Experiments on an image segmentation benchmark as well as on electron microscopic recordings show that this approach yields more precise edge detection and segmentation results compared to unregularized baseline approaches.

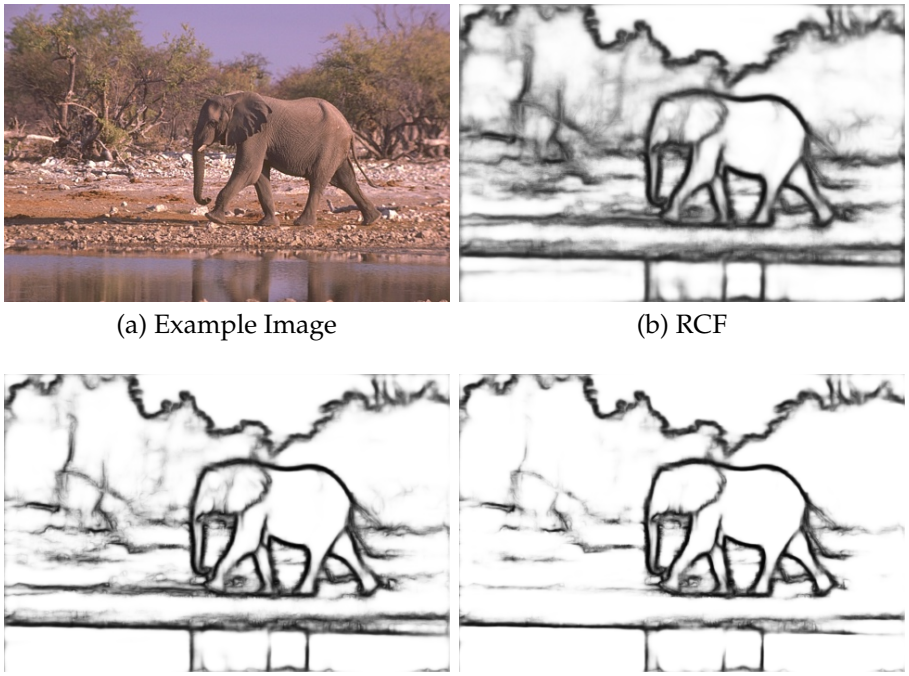
Chapter Outline. We introduce this chapter in Section 4.1. Then, we recap the minimum cost multicut problem in the context of edge detection and image segmentation in Section 4.2. In Section 4.3, we describe how to incorporate cycle consistency constraints as penalty terms in the training process of edge detection networks. We demonstrate the effectiveness of this formulation on segmentation benchmarks in Section 4.4, and conclude this chapter in Section 4.5.

4.1 Introduction

IMAGE segmentation is the task of partitioning an image into multiple disjoint components such that each component is a meaningful part of the image. While there are many different approaches for image segmentation [Arb+09], formulations based on the Minimum Cost Multicut Problem (MP) [CR93; DL97] were successful in the past [Kap+11; And+13; BHK15; Keu+15]. In this formulation, the number of components is unknown beforehand and no bias is assumed in terms of component sizes. The resulting segmentation is only determined by an input graph [Keu+15] built upon image pixels or superpixels, for which edge features can be generated by an edge detector such as Convolutional Neural Networks (CNNs) predicting edge probabilities.

A feasible solution to the MP decomposes the graph into disjoint subgraphs via a *binary* edge labeling. The decomposition is enforced by cycle consistency constraints in general graphs. If a path exists between two nodes where the direct edge between both is cut, then this constraint is violated. Song et al. [Son+19] introduced relaxed cycle constraints, i.e. constraints evaluated on non-binary network predictions, as higher-order potentials in a Conditional Random Field (CRF) to allow for end-to-end training of graph-based human pose estimation. By doing so, cycle constraint violations become a supervision signal for pose tracking. We transfer this approach to image segmentation, where we implement such constraints as penalization to encourage that object contours are closed. Yet, we observe that optimizing non-binary network predictions instead of binary edge labels only leads to few additionally closed contours. This is consistent with prior works, which show that relaxations of the cycle constraints to non-binary edge labels are too loose in practice [Kap+11]. In the CRF formulation, we propose to alleviate this issue by enforcing "more binary" (i.e. closer to 0 or 1) edge predictions that lead to less violated cycle constraints after discretization.

Our contributions are twofold. We are the first to address learnable, boundary-driven image segmentation using multicut constraints. To this end, we combine a CRF with the neural edge detection model Richer Convolutional Features (RCF) [Liu+19] to design an end-to-end trainable architecture that inputs the original image and produces a graph with learnable edge probabilities optimized by the CRF. Second, we propose an approach that progressively uses "closer to binary" boundary estimates in the optimization of the CRF and thus resolves progressively more boundary conflicts. In consequence, the end-to-end trained network yields more and more certain predictions throughout the training process while reducing the number of violated constraints. We show that this approach yields improved results for edge detection and segmentation on the BSDS500 benchmark [Arb+11] and on neuronal structures [Arg+15]. An example is given in Figure 4.1. Compared to the plain RCF model as well as to the RCF with the CRF by Song et al. [Son+19], our approach issues cleaner, less cluttered edge maps with closed contours.



(a) Example Image (b) RCF

(c) RCF-CRF (basic) (d) RCF-CRF (adaptive)

Figure 4.1: **(a)** Example BSDS500 [Arb+11] test image, **(b)** the edge map produced with the RCF edge detector [Liu+19], **(c)** the edge map by RCF-CRF using the basic CRF, and **(d)** the edge map from RCF-CRF using our adaptive CRF. The adaptive CRF promotes closed contours and removes trailing edges.

4.2 Related Work

Edge detection in the context of image segmentation is usually based on learning-driven approaches to learn to discriminate between object boundaries and other sources of brightness change such as textures. Structured random forests have been employed by Dollár and Zitnick [DZ13]. Xie and Tu [XT15] proposed a CNN-based approach called Holistically-Nested Edge Detection (HED) that leverages multiple edge map resolutions. Similarly, Kokkinos [Kok17] propose an end-to-end CNN for low-, mid- and high-level vision tasks such as boundary detection, semantic segmentation, region proposal generation, and object detection in a single network based on multiscale learning. Convolutional Oriented Boundaries (COB) [Man+16a] compute multiscale oriented contours and region hierarchies in a single forward pass. Such boundary orientations are needed as input along with the edge maps to compute hierarchical image segmentations in frameworks such as Multiscale Combinatorial Grouping (MCG) [Arb+14; Pon+17] or gpb-owt-ucm [Arb+11]. He et al. [He+19] propose a Bi-Directional Cascade Network (BDCN) for edge detection of objects in different scales, where individual layers of a CNN model are trained by labeled edges at a specific scale. Similarly, to address edge detection in multiple scales and aspect ratios, Liu et al. [Liu+19] provide an edge detector using RCF by exploiting multiscale and multilevel information of objects. Although BDCN provides slightly better edge detection accuracy on the BSDS dataset [Arb+11], we base our approach on the RCF edge detection framework because of its more generic training procedure.

The multicut approach has been extensively used for image and motion segmentation [Kap+15b; Kap+11; Keu+15; Arb+11; BHK15; And+11; KAB15; Keu17; Kar+20a; Lev+23; AS21]. Due to the NP-hardness of the MP, segmentation has often been addressed on pre-defined superpixels [And+13; Kap+11; Kap+16; BHK15]. While Kappes et al. [Kap+15b] utilize multicuts as a method for discretizing a grid graph defined on the image pixels, where the local connectivity of the edges define the join/cut decisions and the nodes represent the image pixels, Keuper et al. [Keu+15] proposed long-range terms in the objective function of the multicut problem defined on the pixel grid. An iterative fusion algorithm for the MP has been proposed by Beier, Hamprecht, and Kappes [BHK15] to decompose the graph. Andres et al. [And+11] propose a graphical model for probabilistic image segmentation and globally optimal inference on the objective function with higher orders. A similar higher order approach is also proposed by Kappes et al. [Kap+16] and Kim et al. [Kim+14] for image segmentation. Jung and Keuper [JK22] introduce a general solver for multicut problems based on graph convolutional neural networks and Kardoost and Keuper [KK21] extend the heuristic from Keuper et al. [Keu+15] to facilitate the estimation of uncertainties.

4.3 Penalizing Networks with Cycle Constraints

4.3.1 Cycle Constraints in the Multicut Problem

The MP is based on a graph $G = (V, E)$, where every pixel (or superpixel) is represented by an individual node or vertex $v \in V$. Edges $e \in E$ encode whether two pixels belong to the same component or not. The goal is to assign every node to a cluster by labeling the edges between all nodes as either "cut" or "join" in an optimal way based on real-valued edge costs w_e, where positive edge costs are attractive and negative edge costs are repulsive. One of the main advantages of this approach is that the number of components is not fixed beforehand, contrary to other clustering algorithms, and is determined by the input graph instead. Since the number of segments in an image cannot be foreseen, the MP is a well-suited approach. The MP can be formulated as an integer linear program [CR93; DL97] with objective function $c : E \to \mathbb{R}$ as follows:

$$\min_{y \in \{0,1\}^E} c(\mathbf{y}) = \mathbf{y}^\top \mathbf{w} = \sum_{e \in E} w_e y_e \tag{4.1}$$

subject to

$$\forall C \in \mathrm{cc}(G),\ \forall e \in C: \quad y_e \leq \sum_{e' \in C \setminus \{e\}} y'_e, \tag{4.2}$$

where y_e is the binary labeling of an edge e that can be either 0 (join) or 1 (cut), and $\mathrm{cc}(G)$ represents the set of all chordless cycles in the graph. If the cycle inequality constraint in Equation 4.2 is satisfied, the MP solution results in a decomposition of the graph and therefore in a segmentation of the image. Informally, cycle inequality constraints ensure that there cannot be exactly one cut edge in any chordless cycle. However, computing an exact solution is not tractable in many cases due to the NP-hardness of the problem. Relaxing the integrality constraints such that $y \in [0, 1]^E$ can improve tractability, however, valid edge label configurations are not guaranteed in this case. An example can be seen in Figure 4.2, where node B is supposed to be in the same component as A and C, however, A and C are considered to be in different components. Infeasible solutions have to be repaired in order to obtain a meaningful segmentation. This can be achieved by using heuristics [Bei+14; KK18; Keu+15; Pap+17].

4.3.2 Incorporating Cycle Constraints into a CRF

To improve the validity of relaxed solutions, Song et al. [Son+19] reformulate the MP as an end-to-end trainable CRF based on a formulation as Recurrent Neural Network (RNN) by Zheng et al. [Zhe+15]. By doing so, they are able to impose costs for violations of cycle inequality constraints during training. This is accomplished by first transforming the MP into a binary cubic problem, considering all triangles in the graph:

$$\min_{y \in \{0,1\}^E} \sum_{e \in E} c_e y_e + \gamma \cdot \sum_{\{u,v,w\} \in \binom{V}{3}} (y_{u,v}\overline{y}_{v,w}\overline{y}_{u,w} + \overline{y}_{u,v}y_{v,w}\overline{y}_{u,w} + \overline{y}_{u,v}\overline{y}_{v,w}y_{u,w}).$$

This formulation moves cycle inequalities into the objective function by incurring a large cost γ whenever there is an invalid edge label configuration like $(\text{cut}, \text{join}, \text{join})$ in a clique (as shown in Figure 4.2 – all other orders implied). The binary cubic problem is then transformed to a CRF by defining a random field over edge variables $\mathbf{y} = (y_1, y_2, ..., y_{|E|})$, which is conditioned on image $\mathbf{x}$. The cycle inequality constraints are incorporated in the form of higher-order potentials as they always consider three edge variables. Combining unary and third-order potentials yields the following energy function defining the CRF:

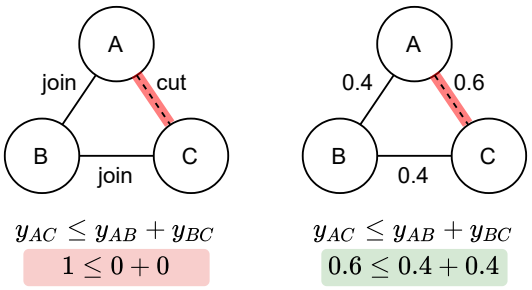
$$E(\mathbf{y} \mid \mathbf{x}) = \sum_i \psi_i^U(y_i) + \sum_c \psi_c^{Cycle}(y_c).$$

The energy function $E(\mathbf{y} \mid \mathbf{x})$ combines unary and higher-order (cycle-based) potentials. For the higher-order terms, Song et al. [Son+19] used pattern-based potentials proposed by Komodakis and Paragios [KP09]:

$$\psi_c^{Cycle}(y_c) = \begin{cases} \gamma_{y_c} & \text{if } y_c \in P_c \\ \gamma_{max} & \text{otherwise} \end{cases},$$

where P_c is the set of valid edge label configurations containing $(\text{join}, \text{join}, \text{join})$, $(\text{join}, \text{cut}, \text{cut})$, and $(\text{cut}, \text{cut}, \text{cut})$. The only invalid edge label configuration is $(\text{cut}, \text{join}, \text{join})$. The potential assigns a high penalty γ_{max} to invalid configurations, and a low cost γ_{y_c} to valid ones, depending on label pattern y_c.

Such CRFs can be made end-to-end trainable using mean-field inference, as proposed by Zheng et al. [Zhe+15]. This approach introduces an auxiliary distribution q over $\mathbf{y}$ that is



$y_{AC} \leq y_{AB} + y_{BC}$ $\quad$ $y_{AC} \leq y_{AB} + y_{BC}$

$1 \leq 0 + 0$ $\quad$ $0.6 \leq 0.4 + 0.4$

Figure 4.2: An example of an infeasible solution to the multicut problem depicting a violated cycle consistency constraint. **(left)** In the integer solution, the cycle inequality constraint covers the infeasibility. **(right)** Example of a relaxed solution that results in (left) when rounded and is feasible according to cycle inequality constraints.

optimized to minimize the Kullback-Leibler Divergence (KLD) [Csi75; Csi+07] between q and the true posterior distribution of $\mathbf{y}$. Optimizing $q(y_i)$ instead of the discrete variable y_i can be interpreted as a form of relaxation. Rather than enforcing hard constraints on binary edge variables (as in Equation 4.1), this method operates over soft assignments by optimizing the probabilities $q(y_i = l)$, which lie in the interval $[0, 1]$. These probabilities represent the belief of the model that node i takes label l.

Zheng et al. [Zhe+15] reformulate the update steps as individual CNN layers and repeat this stack multiple times to perform mean-field iterations. The repetition is treated as a recurrent process like in RNNs, and enables full trainability via backpropagation. Vineet, Warrell, and Torr [VWT14] extend this idea by incorporating higher-order potentials in the form of pattern-based potentials and co-occurrence potentials. For the CRF by Song et al. [Son+19], the corresponding update rule becomes:

$$q_i^t(y_i = l) = \frac{1}{Z_i} exp\Bigg\{ -\sum_{c \in C} \Bigg(\gamma_p \cdot \overbrace{\sum_{p \in P_{c|y_i=l}} \Bigg(\prod_{j \in c, j \neq i} q_j^{t-1}(y_j = p_j) \Bigg)}^{\text{valid labeling case gets low costs}} + \gamma_{max} \cdot \underbrace{\Bigg(1 - \Bigg(\sum_{p \in P_{c|y_i=l}} \Bigg(\prod_{j \in c, j \neq i} q_j^{t-1}(y_j = p_j) \Bigg) \Bigg) \Bigg)}_{\text{inverse of the valid labeling case gets high costs}} \Bigg) \Bigg\}, \quad (4.3)$$

where $q_i^t(y_i = l) \in [0, 1]$ denotes the probability that edge i takes label $l \in \{\text{cut}, \text{join}\}$ at iteration t. $P_{c|y_i=l}$ denotes the subset of valid edge label configurations in cycle c where the label of edge i is fixed to l, as defined by Equation 4.2. Looking at the case where $y_i = 1$ (i.e. the edge is cut), possible valid configurations are $(y_i, 0, 1)$, $(y_i, 1, 0)$, and $(y_i, 1, 1)$. For each valid labeling, the update rule considers the product of the marginal label probabilities $q_i^{t-1}(y_i = p_j)$ for the other two variables $y_{j \neq i}$ in the clique c. These products are summed over all valid configurations and then scaled by the valid labeling cost γ_p. The inverse of this aggregated probability mass is then scaled by the cost for invalid labelings γ_{max}. This formulation is equivalent to summing over all invalid configurations for the given fixed label of y_i. The final potential is accumulated over all cliques C that include the edge variable y_i. After computing the unnormalized log-probabilities for each label, a softmax function is applied to project the resulting values into the interval $[0, 1]$, yielding the updated marginal distribution $q_i^t(y_i = l)$. Costs γ_p and γ_{max} are considered as trainable parameters. The update rule is fully differentiable with respect to both the current marginals $q(y_i = l)$ and the cost parameters. Because the marginals are non-binary, the penalty for invalid configurations becomes smaller with uncertainty about the predicted label (i.e., when $q(y_i = l) \to 0.5$).

4.3.3 Cooling Mean-Field Updates

There have been various efforts to tighten the relaxation of the MP. For example, Swoboda and Andres [SA17] incorporate odd-wheel inequalities, while Kappes et al. [Kap+16] propose additional terminal cycle constraints. Although such constraints yield tighter solution bounds, they involve a large number of edges. A formulation of such constraints to higher-order CRFs, particularly those of order four or higher, becomes computationally intractable. Therefore, we choose a more straight forward and tractable alternative by interpreting the network predictions $q(\mathbf{y})$ as relaxed edge label assignments and progressively pushing them closer toward binary

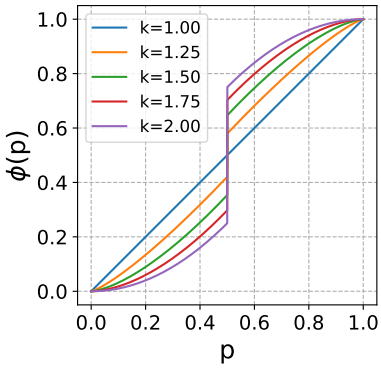


Figure 4.3: Function ϕ plotted for different values of exponent k.

values (i.e., 0 and 1) during mean-field updates. For this, we modify the mean-field update in Equation 4.3 by replacing the term $q_i^{t-1}(y_i = p_j)$ with a transformed value $\phi\left(q_j^{t-1}(y_j = p_j), k\right)$, where $\phi(\cdot)$ is a newly introduced sharpening function defined as follows:

$$\phi(q,k) = \begin{cases} 1-(1-q)^k & \text{if } q >= 0.5 \\ q^k & \text{otherwise.} \end{cases}$$

Here, $q \in [0,1]$ represents the current edge probability, and k is an exponent that controls the strength of the sharpening and can be adapted during training. The function ϕ pushes probabilities greater than or equal to 0.5 closer to 1, and probabilities below 0.5 closer to 0. In doing so, it narrows the distribution around the two binary extremes and thus reduces the integrality gap. Consequently, the model is encouraged towards more confident predictions and violations of the multicut cycle constraints are penalized mores strictly, providing a stronger training signal. This effect is amplified the larger the exponent k becomes. Figure 4.3 depicts $\phi(\cdot)$ and the resulting cooling effect for different values of k.

Choosing a large value for exponent k at the beginning of finetuning can be problematic, as the network would need to adjust abruptly to produce near-binary outputs. We thus introduce a dynamic schedule for k throughout the training process that gradually sharpens predictions. The aim is to focus more on confidently violated constraints in the beginning, and progressively strengthen constraints on less confident predictions over time. We therefore propose a cooling schedule that incrementally increases k based on a constraint satisfaction criterion:

$$k = \begin{cases} k + 0.05 & \text{if } N(C_{inv}) < a \\ k & \text{otherwise,} \end{cases} \tag{4.4}$$

where $N(C_{inv})$ is the average number of invalid cycles (that do not adhere to the cycle inequality constraints) across all images, and the hyperparameter a specifies the maximum number of allowed cycle constraint violations before increasing the exponent k. With this cooling schedule, k is increased after every epoch in which the number of invalid (relaxed) cycles falls below the threshold a. This strategy progressively tightens the relaxation and encourages the model to produce increasingly binary edge predictions, while maintaining training stability in the early stages.

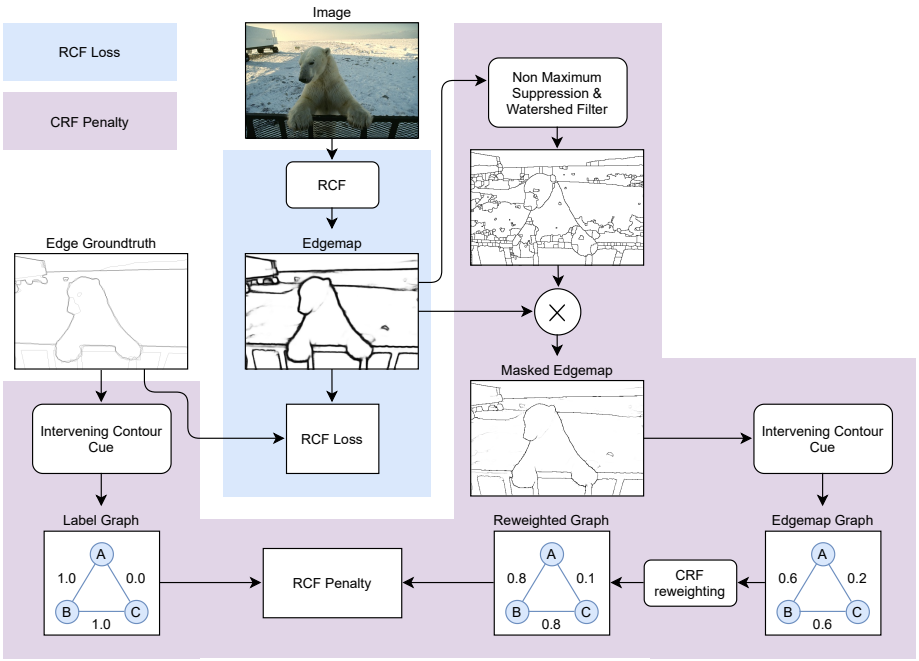


Figure 4.4: The RCF-CRF training process is depicted, where blue highlights the RCF loss and purple highlights the additional CRF penalty.

To leverage the mean-field update for image segmentation, we require an edge detection network that produces prediction values for the CRF potentials. A suitable learning-based approach that provides high quality edge estimates is the RCF [Liu+19] architecture. In practice, the edge detection network is first pretrained until convergence independently of the CRF, and then finetuned with the CRF. The sharpening exponent k is initialized to 1 and progressively updated during training according to the schedule defined in Equation 4.4.

4.3.4 Penalizing Image Segmentation Networks

The RCF architecture for edge detection was introduced by Liu et al. [Liu+19]. Their main idea is based on the HED framework [XT15], which adapts an image classification architecture like VGG16 [SZ15] by dividing them into five stages and removing fully-connected layers. Each stage produces a side output, which is supervised by an individual loss function. These side outputs are subsequently fused by a learnable weighted fusion layer. In contrast to HED, which only considers the last convolutional layer from each stage, the RCF architecture aggregates features from all convolutional layers within each stage (hence the term *richer convolutional features*). Each side output is transformed via sigmoid activation to produce a probability map.

RCF-CRF Architecture. Figure 4.4 depicts the complete proposed architecture, where the CRF is combined with RCF to encourage consistent boundary predictions. In the first stage, edge probability maps Pb are generated from the input image using the RCF, and the corresponding RCF loss is applied during training. To apply the CRF penalty, we construct a pixel graph in which edges represent potential boundaries, with associated weights derived from edge

probabilities. For each pixel, edges of the 8-connectivity are added for distances ranging from 2 to 8 pixels, yielding a large number of cycles. To efficiently compute edge weights for the graph, we apply non-maximum suppression to the output of the pretrained RCF and then generate Watershed boundaries. This preprocessing step is performed only once and is used to mask subsequent RCF predictions, enabling efficient retrieval of edge weights as they are updated during training. We use the Intervening Contour Cue (ICC) [LM98] to compute edge weights. Specifically, the weight of an edge between pixels i and j is computed as $w_{i,j} = \max(Pb(\rho) \mid \rho \in L_{i,j})$, where $Pb(\rho)$ is the edge probability at pixel coordinate ρ, and $L_{i,j}$ denotes the set of all pixel coordinates along the line connecting i and j (including endpoints). Potential locations of these maximum values can be precomputed using the Watershed masks to improve efficiency.

Edge ground truth labels are generated on-the-fly similar to ICC by checking if an edge exists along the line connecting two pixels in the ground truth edge map to determine their cut/join label. The RCF loss follows the original formulation in Liu et al. [Liu+19], which is based on cross-entropy. It excludes ambiguous edge pixels that have been annotated by fewer than half of the annotators. The total loss for training, referred to as RCF-CRF loss, is computed as the sum of the RCF loss and the CRF penalty. After training, the model can produce segmentations by applying either hierarchical approaches such as MCG [Arb+14], or multicut solvers [Keu+15].

4.4 Experiments

We evaluate our approach in two different image segmentation applications. First, we show experiments and results on the BSDS500 [Arb+11] dataset for edge detection and image segmentation. Second, we consider the segmentation of electron microscopic recordings of neuronal structures [Arg+15].

4.4.1 Berkeley Segmentation Dataset and Benchmark

BSDS500 [Arb+11] contains 200 train, 100 validation, and 200 test images that show colored natural photography often depicting animals, landscapes, buildings, or humans. Due to its large variety in objects with different surfaces and different lighting conditions, it is generally considered a difficult task for edge detection as well as image segmentation. Several human annotations are given per image. The RCF is pretrained on the augmented BSDS500 data used by Liu et al. [Liu+19] and Xie and Tu [XT15]. After convergence, it is finetuned with our CRF using only the non-augmented BSDS images to reduce training time. We evaluate the impact of finetuning the CRF in different training settings:

- **Baseline RCF**: Further training of the RCF without CRF.
- **Baseline CRF**: Finetuning the RCF network without cooling scheme.
- **Adaptive CRF**: Finetuning with cooling scheme.

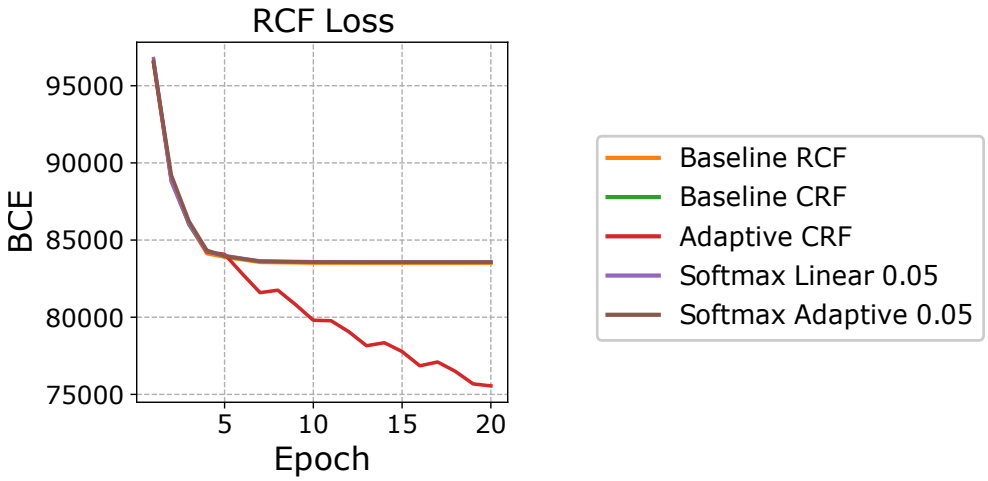
Additionally, we consider two settings where we enforce more binary solutions by introducing temperature decay in the softmax [HVD15] that is applied after each mean field iteration. Here, we consider two cases:

- **Softmax Linear**: Decaying the temperature by 0.05 after each epoch.
- **Softmax Adaptive**: Updating temperature according to the same adaption process as in Equation 4.4.

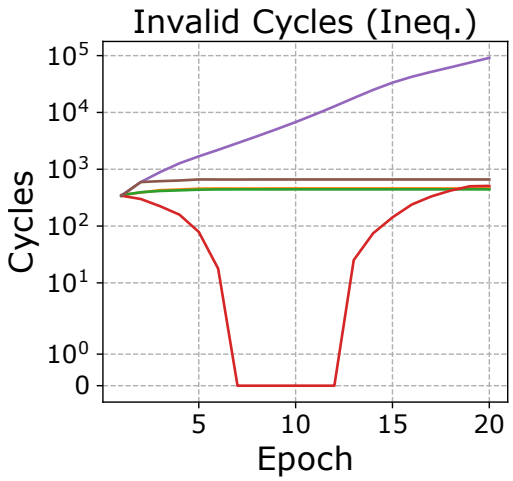
We provide more training implementation details in Section B.1 in Appendix B.

Invalid Cycles. Figure 4.5 (b) depicts the number of violated cycle inequalities during fine-tuning of the RCF network in different settings. There, it can be seen that Baseline CRF is able to reduce the number of violations rapidly to around 500 invalid cycles per image on average. During training, it constantly stays at this level and does not substantially change anymore. Due to this observation, we set parameter a in the cooling scheme of Adaptive CRF and softmax temperature decay to 500. When training Adaptive CRF with this parameter setting, the number of violations further decreases to zero. The impact of reduced uncertainty can furthermore be seen when looking at the number of invalid cycles after rounding edge probabilities to binary edge labels. As Figure 4.5 (c) shows, only the settings Adaptive CRF and Softmax Linear are able to reduce the number of invalid cycles after rounding. However, Softmax Linear is not able to reduce the number of violations before rounding in contrast to Adaptive CRF. This indicates that the cooling scheme of Adaptive CRF provides a better training signal for the RCF, which is also confirmed considering RCF training loss depicted in Figure 4.5 (a). Only Adaptive CRF is able to provide sufficiently strong training signals such that the RCF network can further decrease its training loss. Interestingly, the number of invalid rounded cycles as well as constraint violations start increasing again after some training iterations for the Adaptive CRF. This shows the trade-off between the RCF loss and the penalization by the CRF.

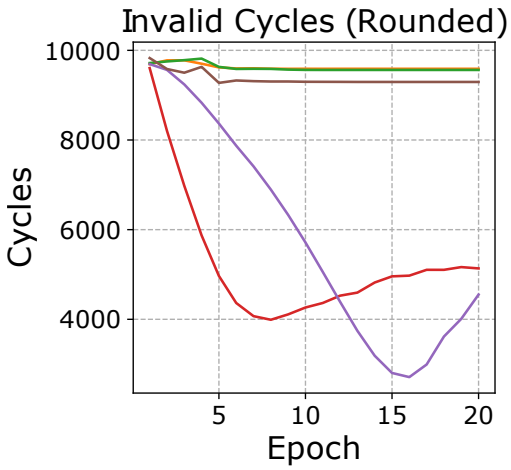
Edge Detection. Table 4.1 shows evaluation scores on the BSDS500 test set. The F-measure at the Optimal Dataset Scale (ODS) and the Optimal Image Scale (OIS) as well as the Average Precision (AP) are reported. The Multiscale Version (MS) is computed similar to Liu et al. [Liu+19] with scales 0.5, 1.0 and 1.5. Adaptive CRF achieves substantial improvements in terms of ODS and OIS to all other training settings. Figure 4.6 shows the precision recall curves for edge detection on the BSDS500 test set. The proposed Adaptive CRF yields the highest ODS score and operates at a higher precision level than other models. AP decreases slightly with the CRF models, which is expected since the CRF removes uncertain edges that do not form closed components and therefore affects the high recall regime. Qualitative examples are shown in Figure 4.7.



(a) RCF Loss



(b) Invalid Cycles



(c) Invalid Cycles (rounded)

Figure 4.5: **(a)** Training progress in terms of RCF loss when training in different settings. Only Adaptive CRF is able to reduce the RCF loss further, while all other settings provide insufficient training signal to further improve on the basic RCF model. **(b)** Number of invalid cycle inequalities during training. Adaptive CRF substantially outperforms all other training settings. **(c)** Number of invalid cycle inequalities after rounding the solution during training. Adaptive CRF and Softmax CRF are able to improve on baselines substantially.

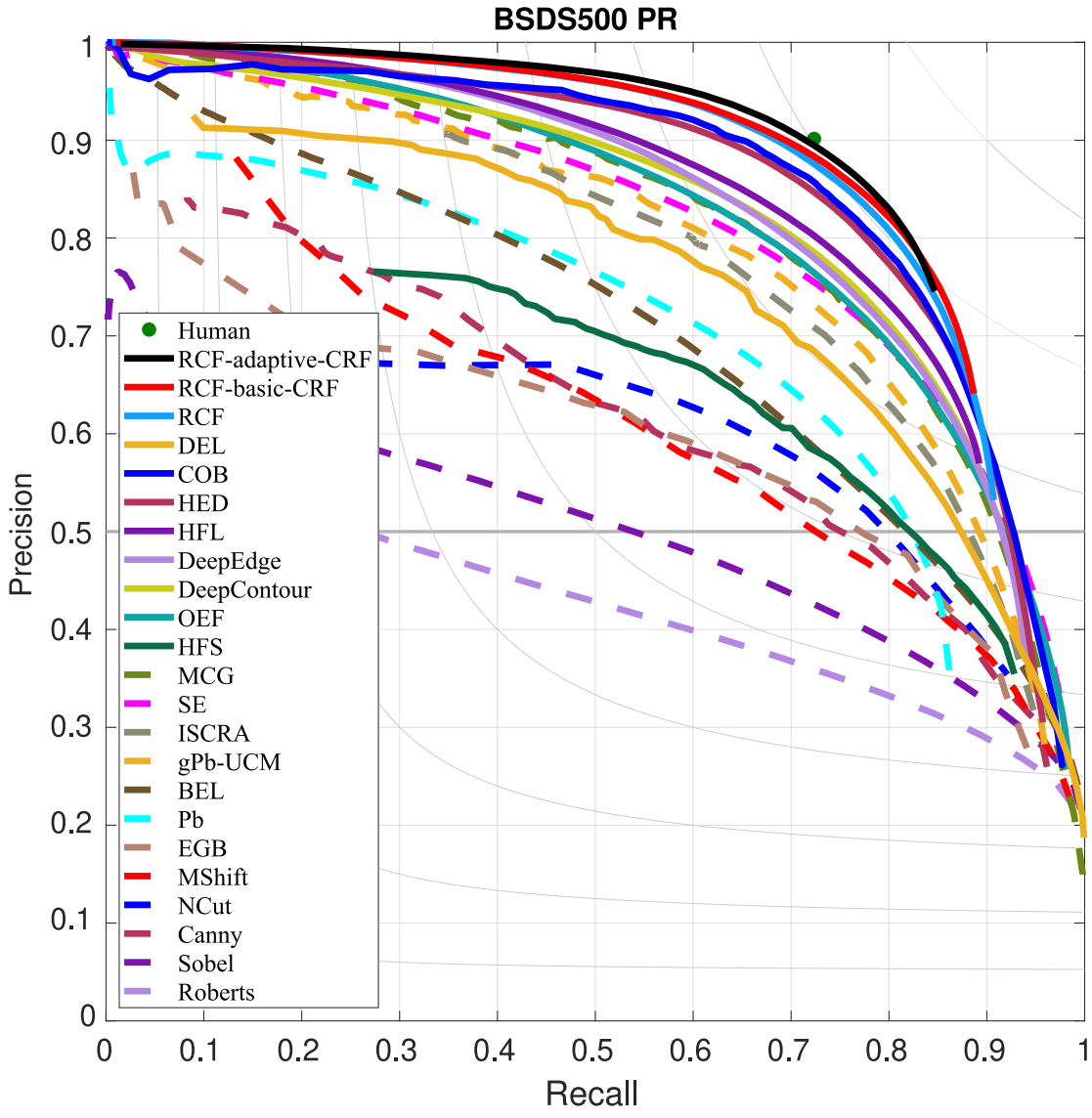


Figure 4.6: Boundary precision recall curves for edge detection on the BSDS500 test set. RCF models all have the VGG backbone. Models that were optimized with the CRF yield steeper curves.

Figure 4.7: Example BSDS500 [Arb+11] test images and resulting edgemaps, UCMs, and segmentations. The edge map optimized with Adaptive CRF is less cluttered and accurately localizes contours compared to Baseline RCF. More examples are shown in Section B.2 in Appendix B.

Table 4.1: Edge Detection and Segmentation results on the BSDS500 test set. All RCF models are based on VGG16 [SZ15]. Results reported for Baseline RCF are computed for a model trained by us and are slightly worse than the scores reported by Liu et al. [Liu+19].

	Edge Detection			Segmentation		
Models	**ODS**	**OIS**	**AP**	**ODS**	**OIS**	**AP**
Baseline RCF	0.811	0.827	0.815	0.803	0.829	0.832
Baseline RCF (MS)	0.812	0.830	**0.836**	0.808	0.832	**0.849**
Baseline CRF (ours)	0.810	0.827	0.815	0.804	0.828	0.831
Baseline CRF (MS) (ours)	0.812	0.831	**0.836**	0.808	0.831	**0.849**
Softmax Linear (ours)	0.810	0.826	0.815	0.803	0.829	0.831
Softmax Linear (MS) (ours)	0.812	0.831	**0.836**	0.808	0.831	**0.849**
Adaptive CRF (ours)	0.815	0.830	0.812	0.808	0.830	0.828
Adaptive CRF (MS) (ours)	**0.817**	**0.835**	0.833	**0.813**	**0.834**	0.847

Image Segmentation. To obtain a hierarchical segmentation using the predicted edge maps, we compute MCG [Arb+14] based Ultrametric Contour Map (UCM) [Arb+11] that generate hierarchical segmentations based on different edge probability thresholds. Edge orientations needed for MCG were computed using the standard filter operations. In contrast to Liu et al. [Liu+19] we do not use the COB framework but use pure MCG segmentations to allow for a more direct assessment of the proposed approach. Results for all training settings are reported in Table 4.1. Again, Adaptive CRF models outperform all other models in ODS, while AP is only slightly affected. Multiscale information additionally improves results.

Figure 4.8 depicts the segmentation precision recall curves comparing the RCF-based methods to other standard models. Similar to the edge map evaluation, the curves are steeper for the CRF-based model compared with the plain RCF, thereby following the bias of human annotations (i.e., approaching the green marker). Depending on when the curves start to tilt, the corresponding F-measure can be slightly lower as it is the case for the basic CRF. Adaptive CRF, however, also yields a considerably higher F-score improving over the baseline from 0.808 to 0.813. This result shows that employing cycle information is generally beneficial to estimate closed boundaries.

4.4.2 Neuronal Structure Segmentation

Next, we conduct experiments on the segmentation of neuronal structures [Arg+15]. The data was obtained from a serial section Transmission Electron Microscopy dataset of the Drosophila first instar larva ventral nerve cord [Car+10; Car+12]. This technique captures images of the Drosophila brain with a volume of $2 \times 2 \times 1.5\mu$ and a resolution of $4 \times 4 \times 50nm/pixel$. The volumes are anisotropic, i.e. while the x- and y-directions have a high resolution, the z-direction has a rather low resolution [Arg+15]. Both, training and test set consist of a stack of 30 consecutive grayscale images. The goal of the challenge is to produce a binary map that corresponds to the membranes of cells in the image.

Beier et al. [Bei+17] have applied a multicut approach to this application. Their pipeline first produced an edge map from the original images by using either a cascaded random forest or a CNN. In order to reduce complexity, they aggregated individual pixels to superpixels by using the distance transform watershed technique [Ach+13]. Based on these superpixels they solve the multicut and the lifted multicut problem using the fusion moves algorithm [BHK15].

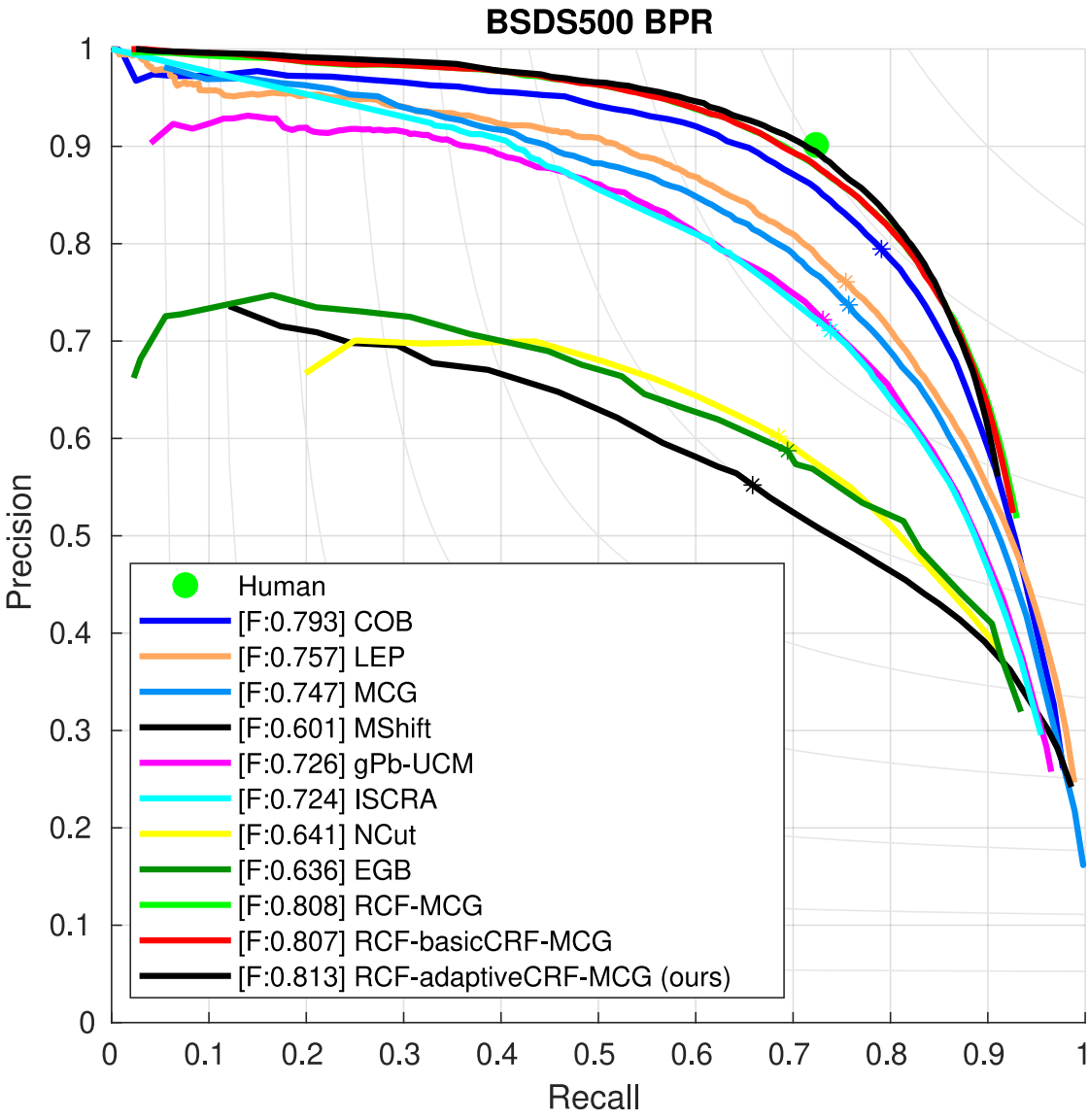


Figure 4.8: Precision recall curves for segmentation on the BSDS500 test set. The proposed RCF model with Adaptive CRF yields the highest ODS score and operates at a higher precision level than other models.

Table 4.2: Results of the ISBI challenge on the test set.

Models	$\mathbf{V}^{\text{Rand}}$	$\mathbf{V}^{\text{Info}}$
Baseline [Bei+17]	0.9753	0.9874
Basic-CRF-optimized (ours)	0.9784	**0.9889**
Adaptive-CRF-optimized (ours)	**0.9808**	0.9887

In this context, we apply Adaptive CRF as a post-processing method to an existing graph without training the underlying edge detection. Edge weights for the test set are computed using a random forest in the simple (not lifted) multicut pipeline from [Bei+17] and define a graph. We optimize these graph weights with the proposed approach and subsequently decompose the graph using the fusion move algorithm as in Beier et al. [Bei+17]. The number of mean-field iterations was set to 20 and for Adaptive CRF the update threshold a was set to 100. Since the CRF is not trained but used only for optimizing the graph once, the update threshold is evaluated after every mean-field iteration rather than every epoch.

The graph obtained from Beier et al. [Bei+17] for the test set contained 74 485 cycles in total. Before applying the CRF, 61 099 of them violated the cycle inequality constraints and 38 283 cycles were invalid after rounding. Afterwards, both cycle counts were close to zero. Table 4.2 contains the results obtained for the not-modified edge weights (Baseline), the edge weights optimized with Baseline CRF and the edge weights optimized with Adaptive CRF. For evaluation, we also refer to the ISBI challenge [Arg+15] that indicates two measures: (i) The foreground-restricted Rand Scoring after border thinning V^{Rand}, and (ii) the foreground-restricted Information Theoretic Scoring after border thinning V^{Info}. Both CRF models were able to improve the segmentation result in both evaluation metrics. While the differences in V^{Info} score are rather small, Adaptive CRF increased the V^{Rand} score by 0.005 compared to the baseline. Taking into account that the baseline is already very close to human performance, this is a very good result. Comparing the two CRF models it can be seen that the V^{Info} score is almost the same for both approaches. In terms of V^{Rand} score, Adaptive CRF improves stronger over the baseline than Baseline CRF. Overall, this experiment shows that applying the CRF is beneficial for image segmentation even without training for edge extraction. Accordingly, our approach can be applied even as a post-processing step without an increase in training time.

4.5 Conclusion and Outlook

Summary. We introduced an adaptive higher-order CRF that can be applied as penalty for an edge detection network. This penalty encodes a preference for closed contours in the edge maps by encouraging compliance with the cycle constraints defined by the MP. Combining this CRF with the RCF model [Liu+19] for edge detection yields sharper edge maps and promotes closed contours on BSDS500. Precision recall curves show that CRF penalized models yield steeper curves having a higher precision level. Similarly, the resulting segmentations show that the approach is able to generate more accurate and valid solutions. Moreover, the CRF can be used as post-processing to optimize a graph for cycle constraints as shown on the electron microscopy data. This has shown considerable improvement in the evaluation metrics without increasing training time.

Solution Quality Guarantees. Similar to Chapter 3, we incorporate MP constraints as regularizer into the training of the model. These constraints are again encoded softly and encourage the model to learn closed and thin object contours, but do not enforce them. Consequently, this regularizer cannot guarantee the absence of disconnected contours. However, the approach of encoding disconnected contours as violated constraints allows us to count them and to provide an approximation of the solution quality in this context. Future research could facilitate this aspect and provide confidences about possible contours and their quality in addition to edge probabilities.

Computational Cost of Cycle Consistency Constraints. In the proposed regularization approach, we treat every pixel in the image as a node of the resulting MP instance. Since it is sufficient to enumerate all triangles for cycle consistency constraints [GW90] when MP instances are defined as complete graphs, it would make sense to connect every pixel with each other pixel in the image. However, this yields an intractable amount of potentials for the proposed CRF. Hence, designing a mapping from the prediction map provided by the RCF model to an MP instance is not trivial. We decided to connect each pixel with its 8-connectivity (for a range of pixel distances from 2 up to 8) to create a sparse graph. For the CRF penalty, we then consider triangles for cycle consistency constraints. As a result, this decision leaves (longer) chordless cycles of the graph unaccounted for. This is similar in fashion to our Graph Convolutional Network (GCN)-based approach in Chapter 3, where we only considered paths up to a certain length. Future research could investigate the influence of this approximation, and similarly, find ways to strategically select and incorporate cycles into the penalty to improve computational efficiency further.

Flexibility to Train from Scratch. The current approach precomputes a mask for the ICC step based on initial predictions by the pretrained RCF (once) to improve training compute time. This design choice limits the proposed RCF-CRF model in two ways. First, it can only be applied to provide a training signal for finetuning (or to refine edge weights during inference). Second, when edge predictions deviate from the ground truth, the penalty term could promote incorrect edges, potentially working against the classification loss. Another reasonable approach would be to use the ground truth as a mask for ICC instead, which enables the model to be trained from scratch. Future work could investigate this alternative design choice further. Combined with strategic selection of computed constraints for improve training efficiency, this could allow training a model from scratch instead of only finetuning existing ones.

This page intentionally left blank.

Part II

Match: Regularization via Feature Matching

This part of the thesis focuses on regularizing the training of Generative Adversarial Networks (GANs) by penalizing discrepancies between features of generated samples and those of the training data. In contrast to Part I, were we considered penalty terms based on prediction outputs, here we penalize based on (learned) feature distributions comparing synthesized images with their corresponding training data. First, in Chapter 5, pretrained image representations are leveraged by incorporating the image quality metric Fréchet Inception Distance (FID) as a training regularizer. This reveals flaws of the widely adopted metric that we discuss further. Second, Chapter 6 shows that GANs overfit in the spatial domain and benefit from regularization in the frequency domain. For this, we add a discriminator network that extracts spectral features from images and learns to distinguish synthesized images from training data in the frequency domain. Applied as an additional loss alongside the spatial discriminator during generator training, this formulation encourages the generator to align synthesized images with training images across both spectral and spatial domains, thereby promoting better generalization.

This page intentionally left blank.

Chapter 5

Learned Representations to Penalize Image Synthesis

Contents of This Chapter

Chapter Topic. This chapter is based on Jung and Keuper [JK21a]. It explores how learned representations from pretrained image classification networks can regularize the training of generative adversarial networks for image synthesis. For this, we incorporate a penalty term based on Fréchet Inception distance, a metric commonly used to evaluate image quality in image synthesis tasks. This penalty compares the distribution of learned features between the training and a batch of synthesized images. Based on experiments with this learned penalty approach, the chapter identifies inherent limitations using Fréchet Inception distance as a metric for image quality. We demonstrate that rankings on this metric are not necessarily aligned with human judgement.

Chapter Outline. We introduce this chapter in Section 5.1. We start by summarizing recent works on generative adversarial networks and their evaluation metrics in Section 5.2. Then, in Section 5.3 we introduce different training settings that incorporate Fréchet Inception distance as additional penalty term and discuss their results. We discuss further considerations about the reliability of Fréchet Inception distance as image quality metric in Section 5.4 and conclude the chapter in Section 5.5.

5.1 Introduction

THE generation of photo-realistic, unseen images has made substantial progress with the introduction of Generative Adversarial Networks (GANs) [Goo+14]. Since then, many architectures emerged competing with each other to provide the best performance [BDS19; Kar+20b; Kar+17]. With it arose the question how their performance should be measured to provide a ranking between different approaches. Metrics like Inception Score (IS) and Fréchet Inception Distance (FID) were proposed to evaluate image distributions automatically. These metrics are based on extracting features from images that are provided by the Inception v3 image classification network [Sze+16], and hence inspired their names. While IS was proven to be not useful to compare different models [BS18], its successor FID is now widely adopted within the GAN community. FID compares the distribution of features between two image datasets that are estimated from training data and samples from the generator network.

Well-performing generators are supposed to produce images that match the feature distribution of the underlying training data. Some theoretical and practical shortcomings of FID are already discussed in the literature. For example, a bias due to different sample sizes [CF20], and inconsistent downsampling implementations between different image processing libraries [PZZ22]. However, none of these works discuss the shortcomings that come with its underlying feature extractor. If FID is a well-defined metric in terms of image quality, incorporating it as a training signal seems to be a reasonable approach. To this end, we incorporate FID as a learned penalty term into the training of different GAN architectures. However, by examining the image quality of resulting generators networks, we show that FID is not aligned with human judgement (see Figure 5.1 for an example).

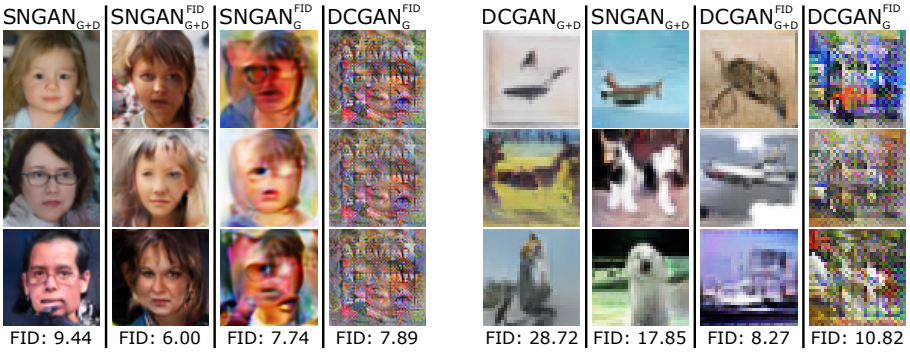


Figure 5.1: FID [Heu+17] is commonly used to decide if one model is superior to another in terms of producing images that are close to the training distribution. Here, we show images generated by different models trained on FFHQ [KLA19] **(left)** and CIFAR10 [Kri09] **(right)** with their respective FID (lower is better).

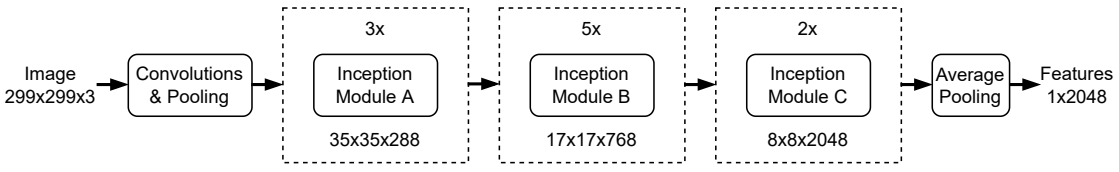


Figure 5.2: Simplified depiction of the Inception v3 architecture (grid size reductions omitted).

5.2 Related Work

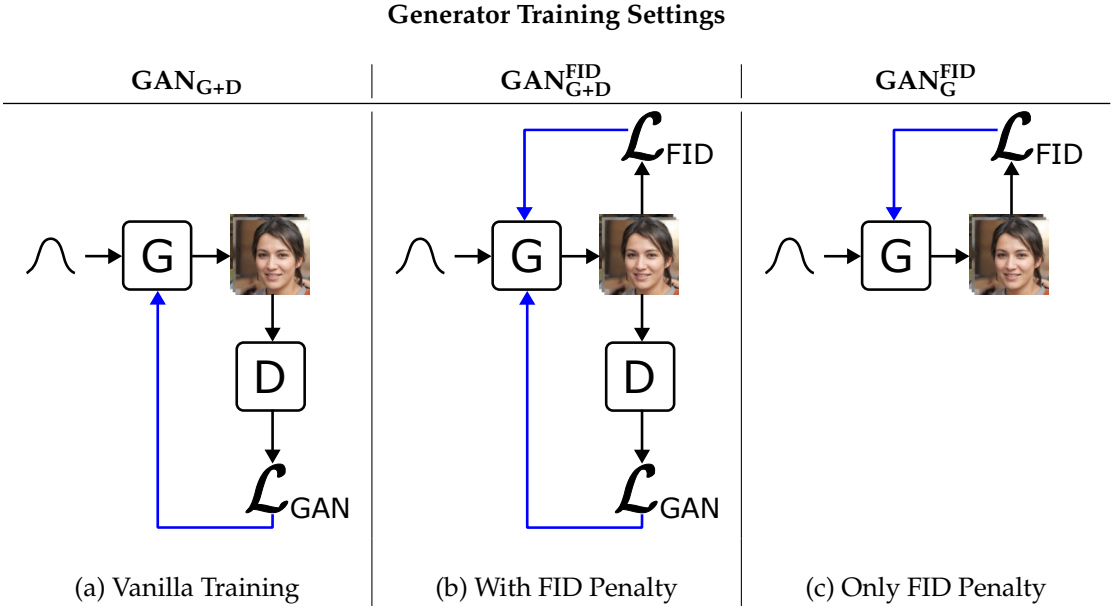
Generative Adversarial Networks. GANs were originally formulated by Goodfellow et al. [Goo+14] and defined the state-of-the-art in realistic image generation before the introduction of diffusion networks [Rom+22] (see also introduction in Subsection 2.2.3). The training progress of GANs is an optimization problem, where the goal of the generative model is to approximate a real data distribution p_{data} with a surrogate data distribution p_θ. To this end, Goodfellow et al. [Goo+14] propose to minimize the Jensen-Shannon divergence between both data distributions. Limitations with respect to the training stability and distribution metrics have for example been discussed by Biau et al. [Bia+20], Li et al. [Li+18a], and Mescheder, Geiger, and Nowozin [MGN18]. Recent works towards improving GANs propose improved loss functions, regularization techniques, and latent space constraints for better training stability [Gul+17a; Mao+17; Gul+17a; Miy+18; MO14; DKD17; GSV17; DKK20; BDS19; Kod+17] and high image resolutions [Kar+17; Kar+20b; Kar+20b]. Such models can generate appealing images that are often indistinguishable from real images.

Evaluation Metrics. Measuring the quality of generative models is a long-standing problem [TOB16]. Due to the popularity of GANs, a significant amount of research efforts is spent to provide metrics for the case of image data. An extensive overview of available metrics is provided by Borji [Bor19; Bor22].

Driven by the urge to evaluate the performance of GANs automatically, Salimans et al. [Sal+16] proposed a metric called IS that measures the variety of image feature vectors provided by the Inception v3 image classification network [Sze+16] trained on ImageNet [Den+09]. This score ranges in $[1, 1000]$ [BS18], where generator networks that produce images resulting in larger scores are considered more desirable. However, Barratt and Sharma [BS18] criticize IS for its sensitivity to small perturbations of the parameters of the underlying Inception network as well as its careless usage in practice. They show that the assumptions of IS are not met when the score is used to compare models that are trained on different datasets than the feature extractor network Inception.

FID [Heu+17] has established itself as standard metric to compare GAN architectures. This metric compares features statistics of generated images to training images extracted by the Inception v3 image classification network. The mean as well as the covariance of generated image features are compared to the real data distribution by computing the Wasserstein-2 distance. When computing FID, a certain number of images, N, is drawn from the generator on which its performance evaluation is based. Different works employ different sample sizes, for example Gulrajani et al. [Gul+17a] compute FID_{10k} generating $10k$ samples, whereas Karras et al. [Kar+20b; Kar+17] compute FID_{50k} generating $50k$ samples. Unfortunately, these differences in evaluation protocols make it hard to compare different models, since Grover et al. [Gro+19] show that FID has a bias related to the number of samples. They advise that to be able to compare different generators in an unbiased fashion, the asymptotic FID_∞ should be estimated. Therefore, they provide a protocol to regress FID_∞ based on different sample sizes.

Since Inception v3 is used in FID as feature extractor, all images are resized to 299^2 (see Figure 5.2) before being fed through the network. However, Parmar, Zhang, and Zhu [PZZ22] show that this introduces practical problems when comparing FID between papers due to differences in the used image libraries that resize input images. They compared different implementations of resizing operations and showed that commonly used deep learning libraries, like PyTorch and Tensorflow, are introducing aliasing artifacts during downsampling.



(a) Vanilla Training (b) With FID Penalty (c) Only FID Penalty

Figure 5.3: Different settings to train generator networks. **(a)** Vanilla generator training is denoted by $\text{GAN}_{\text{G+D}}$. Here, the training signal is provided by a discriminator network. **(b)** In the setting of $\text{GAN}^{\text{FID}}_{\text{G+D}}$, we add an additional penalty term based on FID (see Equation 5.1). **(c)** In $\text{GAN}^{\text{FID}}_{\text{G}}$, we drop the discriminator network and train the generator solely by minimizing the penalty.

5.3 Training with Fréchet Inception Distance

5.3.1 Fréchet Inception Distance

To compute FID between two image datasets, image features are extracted by sampling images from those datasets and feeding them into the pretrained Inception v3 [Sze+16] image classification network. In the following, we denote Inception v3 by function f_h that returns feature column vectors given input images and feature matrices with samples on rows given input batches of images. FID is then computed via the Wasserstein-2 distance

$$\text{FID} = ||\boldsymbol{\mu}_{\mathcal{D}_1} - \boldsymbol{\mu}_{\mathcal{D}_2}||_2^2 + tr(\boldsymbol{\Sigma}_{\mathcal{D}_1}) + tr(\boldsymbol{\Sigma}_{\mathcal{D}_2}) - 2 \cdot tr(\sqrt{\boldsymbol{\Sigma}_{\mathcal{D}_1}\boldsymbol{\Sigma}_{\mathcal{D}_2}}) \tag{5.1}$$

between two image datasets $\mathcal{D}_1$ and $\mathcal{D}_2$, where $tr(\cdot)$ is the trace operator, and

$$\boldsymbol{\mu}_{\mathcal{D}} = \frac{1}{|\mathcal{D}|}\sum_{\mathbf{x}\in\mathcal{D}} f_h(\mathbf{x}) \qquad \text{and} \qquad \boldsymbol{\Sigma}_{\mathcal{D}} = \sum_{\mathbf{x}\in\mathcal{D}} \frac{(f_h(\mathbf{x}) - \boldsymbol{\mu}_{\mathcal{D}})(f_h(\mathbf{x}) - \boldsymbol{\mu}_{\mathcal{D}})^\top}{|\mathcal{D}| - 1} \tag{5.2}$$

are the mean vector and covariance matrix of feature vectors of dataset $\mathcal{D}$. The motivation of FID is that, given a sufficiently large number of samples, the first two moments of the feature distributions should match if the images are sampled from the same dataset. Hence, the closer the generator network gets to reproducing the Inception v3 feature distribution of the training data, the smaller FID becomes.

5.3.2 Minimizing Fréchet Inception Distance

Settings. If FID is a metric that aligns with human judgement, we can assume that the visual appearance of images generated by any GAN model should improve if we optimize the generator by minimizing FID [MH21]. But can this assumption hold? To verify, we train two GAN architectures, Deep Convolutional Generative Adversarial Network (DCGAN) [RMC15] and Spectral Normalization Generative Adversarial Network (SNGAN) [Miy+18], on FFHQ [KLA19] downscaled to 64^2 image resolution (further called FFHQ64) and CIFAR10 [Kri09]. We consider three training procedures that we depict in Figure 5.3. These are: (a) $\text{GAN}_{\text{G+D}}$ where we train the model in its common setting in which a discriminator network provides the training signal for the generator network (baseline), (b) $\text{GAN}^{\text{FID}}_{\text{G+D}}$ where we extend (a) by adding an additional penalty term for the generator network given by Equation 5.1 (further called FID penalty), and (c) $\text{GAN}^{\text{FID}}_{\text{G}}$ where we drop the discriminator network and train the generator solely by minimizing the FID penalty.

To improve training efficiency, we adopt the method of calculating the matrix square root when computing FID (see Equation 5.1) as proposed by Mathiasen and Hvilshøj [MH21]. Specifically, we consider that $tr(\sqrt{\mathbf{\Sigma}_{\mathcal{D}_1}\mathbf{\Sigma}_{\mathcal{D}_2}})$ can be reformulated as:

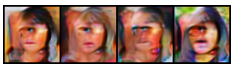
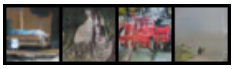
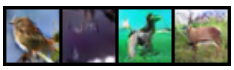
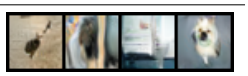
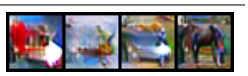
$$tr(\sqrt{\mathbf{\Sigma}_{\mathcal{D}_1}\mathbf{\Sigma}_{\mathcal{D}_2}}) = \sum_{i=1}^{n} |\sqrt{\lambda_i \left[\mathbf{C}_{\mathcal{D}_1}\mathbf{\Sigma}_{\mathcal{D}_2}\mathbf{C}_{\mathcal{D}_1}^{\top}\right]}|, \tag{5.3}$$

where $\mathbf{C}_{\mathcal{D}_1} = \left(f_h\left(\mathcal{D}_1\right) - \boldsymbol{\mu}_{\mathcal{D}_1}\right)$ is the centered feature matrix of dataset $\mathcal{D}_1$ and $\lambda_i[\cdot]$ denotes the ith eigenvalue of the given matrix. Originally, computing the matrix square root in Equation 5.1 involves an eigendecomposition of the matrix product $\mathbf{\Sigma}_{\mathcal{D}_1}\mathbf{\Sigma}_{\mathcal{D}_2} \in \mathbb{R}^{2048\times 2048}$, where the dimensionality corresponds to the feature dimensionality of Inception v3. In contrast, the reformulation in Equation 5.3 applies the eigendecomposition to matrix $\mathbf{C}_{\mathcal{D}_1}\mathbf{\Sigma}_{\mathcal{D}_2}\mathbf{C}_{\mathcal{D}_1}^{T} \in \mathbb{R}^{n\times n}$, where n is the number of data samples. Hence, as long as we sample less than 2048 images to approximate the FID penalty, Equation 5.3 is more efficient.

In each iteration that we minimize FID directly, we generate 400 images for DCGAN and 360 images for SNGAN to approximate the current FID. During training we measure FID to track the training progress after each epoch by sampling 10 000 images from the generator. We provide the complete details to our implementation in the Appendix in Section C.1.

Training Results. The progress for each training setting is shown in Figure 5.4. Given the training curves provided by evaluating FID after each epoch, one could assume that applying FID penalty stabilizes the training. We could go even further and assume that the discriminator is not even necessary for GAN training, since the results measured by FID differ only slightly. However, the story changes after examining the images each of these models generate (we provide an overview of all results with example images in Table 5.1 and more example images in Appendix C). By inspecting image samples, we observe no visual improvement in image quality between $\text{GAN}_{\text{G+D}}$ and $\text{GAN}^{\text{FID}}_{\text{G+D}}$ despite the large gaps in terms of FID. We even argue that images produced by $\text{GAN}^{\text{FID}}_{\text{G+D}}$ contain more artifacts and are less visually pleasing, which we see as a sign of fitting to the FID penalty. We hypothesize that the generator learns to produce features to match the training data distribution on average. This observation becomes more severe in the case of $\text{GAN}^{\text{FID}}_{\text{G}}$. Here, we notice that the missing discriminator leads to spatially incoherent feature distributions. For example $\text{SNGAN}^{\text{FID}}_{\text{G}}$ adds mostly single eyes and aligns facial characteristics in a daunting manner. We assume that this is the result of the choice of layer that features are extracted from Inception v3. Features are spatially pooled and therefore lose

Table 5.1: Combined results of trained models from Subsection 5.3.2. More generated images are provided in the Appendix in Sections C.2-C.5.

Data	Resolution	Model	FID↓	Images
FFHQ	64^2	$\text{DCGAN}_{\text{G+D}}$	14.86	
		$\text{DCGAN}^{\text{FID}}_{\text{G+D}}$	5.38	
		$\text{DCGAN}^{\text{FID}}_{\text{G}}$	7.89	
		$\text{SNGAN}_{\text{G+D}}$	9.44	
		$\text{SNGAN}^{\text{FID}}_{\text{G+D}}$	6.00	
		$\text{SNGAN}^{\text{FID}}_{\text{G}}$	7.74	
CIFAR10	32^2	$\text{DCGAN}_{\text{G+D}}$	28.72	
		$\text{DCGAN}^{\text{FID}}_{\text{G+D}}$	8.27	
		$\text{DCGAN}^{\text{FID}}_{\text{G}}$	10.82	
		$\text{SNGAN}_{\text{G+D}}$	17.85	
		$\text{SNGAN}^{\text{FID}}_{\text{G+D}}$	8.07	
		$\text{SNGAN}^{\text{FID}}_{\text{G}}$	11.66	

all spatial information before they are extracted. While human annotators would surely prefer images produced by $\text{SNGAN}_{\text{D+G}}$ over $\text{SNGAN}_{\text{G}}^{\text{FID}}$ (in cases where data fidelity is preferred over art), we see that this is not reflected by FID. Hence, FID is not aligned with human perception. We argue that discriminative features provided by image classification networks are not sufficient to provide the basis of a meaningful metric. To further emphasize on this point, we analyze the role of the choice of feature-extracting backbone for FID in the next section. There, we substitute Inception v3 by an extensive choice of different image classification networks. Additionally, we evaluate FID on ImageNet-C at different severities for each backbone network. Here we see that biases present in Inception v3 are also widely present in other classification networks. Additionally, we see that different networks would produce different rankings in-between corruption types.

5.4 Further Analysis of Fréchet Inception Distance

FID and Model Robustness. While pretrained feature extractors can facilitate certain tasks in meaningful ways, like object recognition or detection, we argue that caution needs to be taken in the context of comparing image distributions. The Inception v3 network is trained on the ImageNet object recognition challenge [Den+09], where the task is to classify images into 1000 distinct classes. Hence, the network learns to extract features from images that discriminate classes found in ImageNet. For this task, it is beneficial for the network to become robust against several distribution shifts, like color or intensity changes and diverse spatial transformations. A common way to equip image classification networks with robustness against before-mentioned shifts is to augment the training data. For example, Inception v3 was trained with the following augmentation pipeline [Goo21a]: (a) Cropping the input image with a random scale (8%-100%) and aspect ratio ($3/4$ to $4/3$), (b) randomly flipping the input image horizontally, and (c) randomly introducing color distortions in terms of hue, saturation, brightness, and contrast. Consequentially, we assume that the network is at least robust to some degree of corruption. In the following, we investigate (i) how robust Inception v3 is against certain corruption types, and (ii) how these findings impact the applicability of FID as a metric.

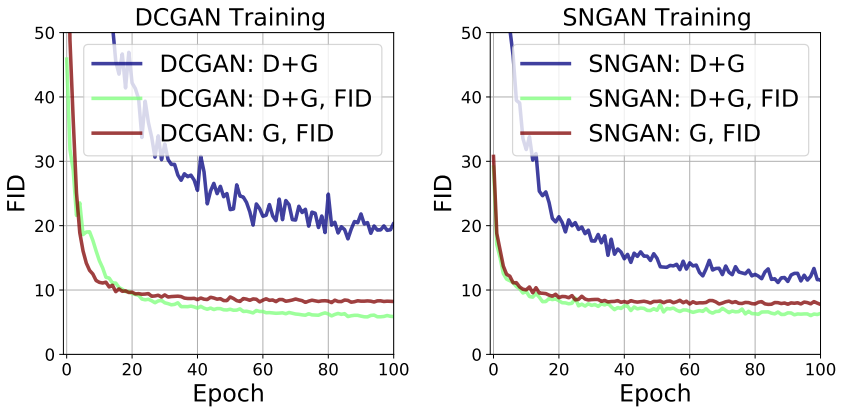


Figure 5.4: Results of training DCGAN and SNGAN with different settings. FID penalty stabilizes training and achieves better performances (i.e. smaller FID), whereas the discriminator only marginally improves generator performance.

We first investigate how different kinds of corruptions influence FID. Research in the area of robustness has produced several benchmark dataset collections, one of which is ImageNet-C [HD19]. This collection contains the ImageNet [Den+09] validation images with 19 corruption types at 5 severity levels, hence, a total of 95 image datasets. On this benchmark, we can uncover which types of corruptions influence the feature distribution provided by Inception v3 (and therefore FID) to which degree. In Figure 5.5 we depict the corresponding FID between the original ImageNet validation images and all datasets in ImageNet-C. We interpret larger FID as indicator that the distribution of features extracted by Inception v3 is influenced more by certain corruptions, and hence that the model is more sensitive to these types of corruptions. We can make the following observations from Figure 5.5: First, Inception v3 is mostly robust to deviations in brightness, saturation, and contrast up to a certain degree. While information about colors and intensities is lost in these cases, edges are mostly preserved. In the frequency domain, we can see these corruptions acting on low frequencies [Yin+19a], while high frequencies are mostly untouched. Second, Inception v3 is sensitive to noise that acts on the high frequency spectrum, either by adding high frequency artifacts (impulse noise, shot noise, speckle noise), or by removing high frequency information (glass blur, Gaussian blur). These observations lead us to the assumption that Inception v3 has a bias towards extracting features based on edges and textures rather than color and intensity information. This aligns with its augmentation pipeline that introduces color distortions, but keeps high frequency information intact. Consequently, FID inherits this bias. When used as ranking metric, generative models reproducing textures well might be preferred over models that reproduce colors well.

Sensitivity to Translation. Since ImageNet-C contains no corruptions that test robustness to translations, we further investigate this aspect by introducing additional corruptions (see Figure 5.6). We are interested in knowing how FID is influenced by (a) flipping images either horizontally or vertically, and (b) by translating the images in certain directions. Our results indicate that Inception v3 is mostly robust towards horizontal flips and translations, while being more sensitive in the vertical direction. Again, this fits well with the training data augmentation of Inception v3 that reinforces robustness via random horizontal flips [Goo21a].

Substituting the Backbone. We provide an overview similar to Figure 5.5 with an extensive list of substitute networks evaluated on ImageNet-C at severity 1 in Figure 5.7. Here we see that biases present in Inception are also widely present when we use different classification networks instead. However, we also see that different networks would produce different rankings in-between corruption types. We additionally provide this overview for all other severities of ImageNet-C in the Appendix in Section C.6.

FID is related to Deep Fake Detection. Should features extracted in the context of comparing image distributions be robust after all? To answer this question, we need to discuss the premise under which two image datasets should be considered equal. We argue that a meaningful metric should be at least visually aligned with human perception. Hence, if a human can be fooled by a generator network, than this generator should be considered superior to one that is not able to do so. In this sense, we believe that measuring the performance of GANs and deep fake detection is interconnected. In Table 5.2 we show results of training logistic regressions to distinguish images from FFHQ and images generated by StyleGAN2 without latent space truncation ($\psi = 1.0$) and with truncation ($\psi = 0.5$), solely based on features provided by Inception v3. Training and testing are performed on different splits (we provide more implementation details in the Appendix in Section C.7). We see that truncation decreases FID substantially, and consequently improves the ability of detecting StyleGAN2 generated images as fake. In contrast, we show exemplary images produced by StylegAN2 without and with truncation in Figure 5.8. By inspecting these images,

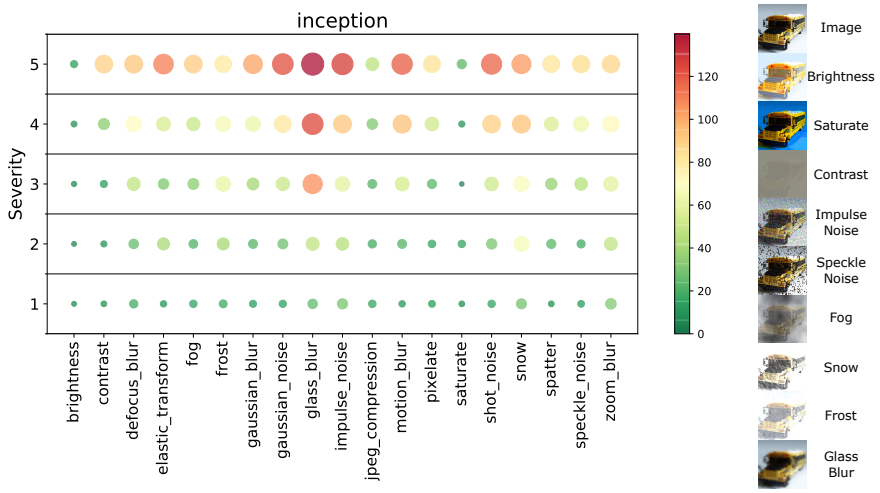


Figure 5.5: **(left)** Color-coded FID between 19 corrupted ImageNet validation datasets with 5 severity levels [HD19] to their originals. Colors and circle sizes are normalized over all corruptions and severities. **(right)** Examples of different corruptions at severity 5.

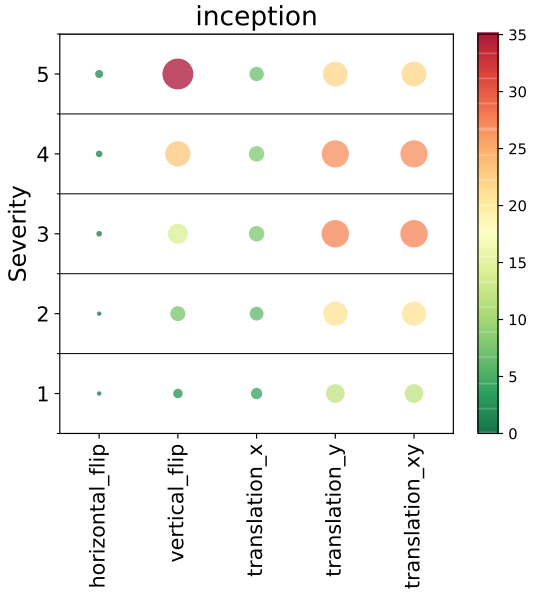


Figure 5.6: Corruptions introduced by transforming CIFAR10 [Kri09] test images a) with increasing likelihood of horizontal or vertical flips and b) by moving the image in x, y, or both directions up to an increasing distance (reflection padding). The figure depicts the FID between a corruption at a certain level with the original CIFAR10 test dataset.

Figure 5.7: Color- and size-coded FID between ImageNet validation images and 19 corrupted versions thereof provided by ImageNet-C [HD19]. Inception v3 is substituted by different classification networks [SZ15; He+16; Zha+17c; Zop+18; Sze+17; Hua+17; IS15; Sze+16] to investigate whether the ranking is affected by the feature extractor. All corruptions are at severity 1. Colors and circle sizes depend on the largest observed FID per network. Additionally, PCA features are shown, which provide descriptive features with different sensitivity to corruptions compared to image classification networks. We can see that rankings are inconsistent in-between different feature extractors.

Table 5.2: Results of fake detection of two logistic regressions trained on features provided by Inception v3 [Sze+17], either on images generated with the truncation trick ($\Psi = 0.5$) or without ($\Psi = 1.0$). Although applying the truncation trick improves image quality (see Figure 5.8), FID deteriorates and fake detection is more accurate in discriminating those images.

Model	Data Resolution	FID↓	Accuracy↑	Precision↑	Recall↑	F1↑
StyleGAN2 $\psi = 1.0$	FFHQ 1024^2	2.65	.71	.70	.72	.71
StyleGAN2 $\psi = 0.5$	FFHQ 1024^2	57.77	.98	.98	.98	.98

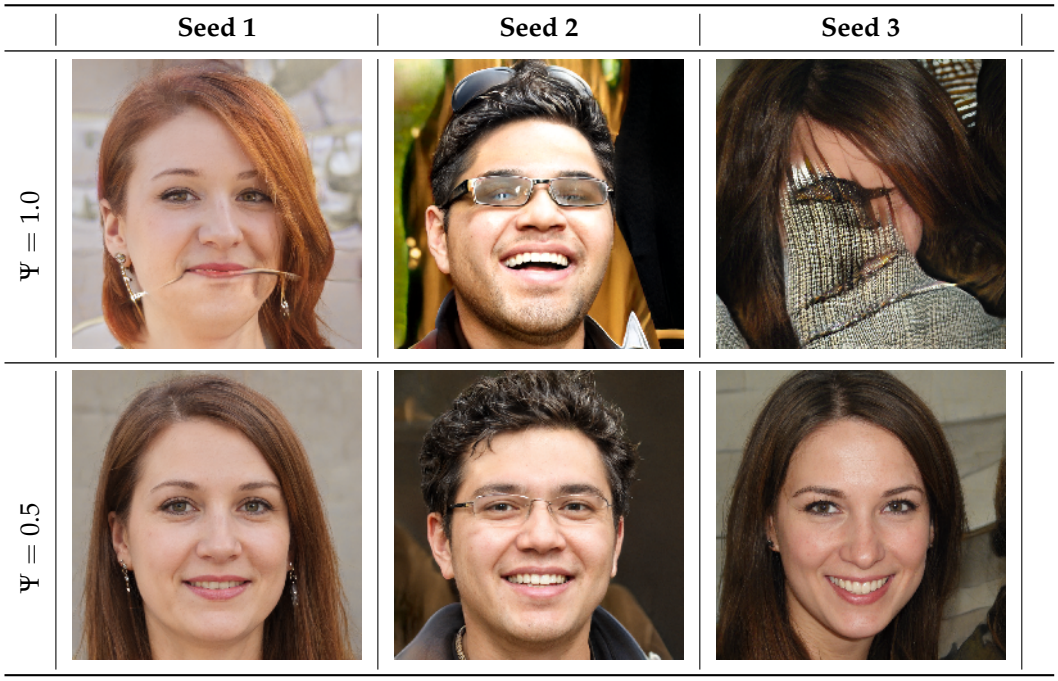


Figure 5.8: Images generated by StyleGAN2 for different seeds. For the same seed the truncation trick [KLA19] is either applied ($\Psi = 0.5$) or not ($\Psi = 1.0$). The truncation trick pulls style vectors toward an average latent vector, resulting in higher-quality but less diverse outputs.

we observe that truncation removes textures (and also artifacts). We hypothesize that its bias towards textures facilitates Inception to extract features that allow almost perfect detectability (98% when truncation is applied). However, we expect humans to be easier fooled by truncated images than untruncated ones. Hence, we argue that this is a hint towards that FID is not aligned with human perception.

5.5 Conclusion and Outlook

Summary. We provided an overview of inherent biases in FID that are present due to the design choice to use Inception v3 as feature extractor. We showed that the way Inception v3 was trained encouraged it to become robust to corruptions related to color, intensity, saturation, and horizontal translations, while being sensitive to corruptions affecting textures and to vertical translations. These biases influence the ranking when different image-synthesizing models are compared and need to be considered. Additionally, we showed that FID as a metric is not aligned with human perception by minimizing FID as a penalty term. Here, we showed that generators trained with FID penalty produce images with substantially improved FID, but worse visual appearance. The plotted training curves showing the development of validation FID are misleading in those cases. Finally, we showed that substituting Inception v3 with another image classification network would simply interchange different biases. Hence, we hope to inspire further research to close the gap towards a humanly aligned and unbiased metric that enables to fairly rank image-generative models.

Parameter-free vs. Parameterized Discriminators. In this chapter we considered FID as parameter-free discriminator that adds a batch-wise penalty comparing the distance of image features statistics between training images and generated images during training. Learned representations by Inception v3 are transferred for this task. While we argued before that computing feature statistics to compare them directly becomes computationally expensive, another possibility is to train another model to compare those image features. This corresponds to adding a discriminator network to the adversarial training of GANs that facilitates pretrained feature extractors. While former work explored this idea in the context of image manipulation [Sun+18], some succeeding works exist now that explored this idea further for image synthesis [Sau+21; Kum+22]. In this context, Sauer et al. [Sau+21] employ a discriminator that projects images into multiple learned feature spaces at different scales, providing feedback to the generator and improving training stability and image quality. Kumari et al. [Kum+22] extend this idea by ensembling representations from multiple feature extractors. Beyond GANs, diffusion models [HJA20] have emerged as competitors for generative modeling. Here, Dhariwal and Nichol [DN21] incorporate pretrained classification networks to class-condition the denoising process.

Approximation of FID and its Efficiency. Even though we improve efficiency of computing FID by reformulating the matrix square root computation, applying FID as regularization during training remains expensive. It requires not only passing large batches of generated images through the backbone feature extractor, but also computing the covariance matrix of the extracted features at each iteration, which itself is a costly operation. As a consequence, we were only able to approximate FID as a training signal on at most 400 images each iteration, since GPU memory restricts the batch size. This limits the scalability of FID as a regularizer, particularly for higher-resolution image synthesis where large batch sizes are prohibitive. At the same time, prior work has shown that FID estimates improve with larger sample sizes [CF20]. One workaround could be to offload parts of the computation, such as covariance estimation, to the

CPU. This, however, comes at the cost of significantly slower training. Future research could investigate possibilities to improve computational efficiency of FID-based regularizer. Examples could be online algorithms [Wel62] to compute necessary feature statistics, as well as gradient accumulation [Lam21] to achieve larger effective batch sizes.

Benchmarking FID Alternatives for Human Perception. Since the publication of this chapter, a number of alternative approaches to FID have been proposed [Alf+22; Kyn+23; Jaj+24; Luz+24; Jay+24]. The main critique points covered by these approaches are either the backbone of FID providing feature representations, or the assumption that representations are Gaussian distributed made by the Wasserstein-2 distance underlying FID. Especially feature extractors like CLIP [Rad+21] or adversarially [GSS15] trained networks seem promising as replacements for Inception v3. In comparison, CLIP is trained on a substantially larger dataset and with a different objective, namely the alignment of images with their corresponding text descriptions (created by humans). It is argued that CLIP features are therefore more aligned with human judgment [Zal+25]. Adversarially trained networks are also argued to learn features that are more aligned with human perception [Tsi+19]. Unfortunately, none of the proposed alternatives adequately address the aspect of human perception in a holistic way. While Jayasumana et al. [Jay+24] use CLIP features and provide a human evaluation study, it is limited to comparing a single model to an earlier checkpoint of itself.

Future research should take rankings provided by human annotators into account when proposing an alternative to FID. Such considerations are already reflected in related metrics comparing two images (rather than two image distributions). For example, Zhang et al. [Zha+18c] train a perceptual similarity metric based on a dataset of human judgments. To collect this dataset, they distort image patches twice and ask annotators which distorted version is closer to the original patch. Similarly, Fu et al. [Fu+23] collect a dataset of human judgment about semantic similarity of two images. For this, annotators had to choose between two images, deciding which of the two is more similar to a reference image. We argue that the next most important necessary step towards future work on an FID successor is to create a benchmark on which possible candidates can demonstrate in which aspects they improve.

This page intentionally left blank.

Chapter 6

Spectral Distribution-Aware Image Synthesis

Contents of This Chapter

Chapter Topic. This chapter is based on Jung and Keuper [JK21b]. It investigates spectral distribution fidelity of generated images as an essential aspect of generalization in image synthesis. For this, we extend learned regularization into the spectral domain by introducing an additional discriminator network that extracts spectral features from images. With these features, the network learns to distinguish synthesized images from their corresponding training data. By providing an additional training signal for the generator network, it acts as a learned regularizer in the spectral domain. Experiments show that this approach substantially improves the alignment of images in the spectral domain, while keeping similar image quality in the spatial domain compared with unregularized models.

Chapter Outline. We introduce this chapter in Section 6.1. Then, we summarize related works on image synthesis in Section 6.2, focusing on flaws in the generation process that materialize in the spectral domain. In Section 6.3, we discuss properties of image generation in the spectral domain by discussing the role of upsampling, analyzing spectral training data distributions, and proposing a method to assess spectral fidelity of synthesized images. In Section 6.4 we introduce our approach which learns to regularize in the spectral domain directly via an additional discriminator network. We show experimental results comparing this approach to baselines in Section 6.5, and conclude this chapter in Section 6.6.

6.1 Introduction

IMAGE generation using Generative Adversarial Networks (GANs) has made substantial progress in recent years. Especially the generation of photo-realistic images at high resolution has arrived at a level where it becomes hard for humans to distinguish between real and generated images. While the training data distribution appears to be well learned in the model's latent space, it is surprising how reliably real and generated images can be distinguished even when various cloaking techniques such as blurring or compression are applied [YDF19]. In recent works on the detection of generated images [Fra+20; DKK20] it was argued that this effect, at least partially, is due to artifacts introduced during the generation process itself. In fact, commonly used upsampling schemes seem to cause these artifacts which are mostly in the high frequency domain and cannot be corrected by the network itself [Fra+20; DKK20; Bai+20]. Such artifacts are undesired, not only because generated images can easily be identified as such, but also because they might be perceivable, for example, as grid patterns on the generated images (see Section D.1 in Appendix D).

Durall, Keuper, and Keuper [DKK20] propose a GAN regularization approach as a remedy. They argue that a generator network, given a sufficient amount of convolutional layers, can generate images with realistic frequency spectra if they are penalized for deviating from the average frequency spectrum of the real images during training. Their proposed regularization allowed to produce more realistic frequency spectra, however, the resulting spectra were still not able to match the distribution of the real data.

In this chapter, we address this problem in a different way. Instead of introducing a regularization term, we propose to use a second discriminator which directly acts on the power spectra of real and generated images. This way, the generator network is not forced to produce images with average power spectra as in Durall, Keuper, and Keuper [DKK20], but is enabled to learn the distribution of frequency spectra from training data. Since the training of GANs is computationally expensive, we argue that an additional discriminator should be lightweight so that diverse architectures can easily adopt it. We therefore base our discriminator on one-dimensional (1D) projections of the frequency spectra instead of acting on the full two-dimensional data. We show that the resulting model can be trained with different commonly used GAN losses and evaluate its ability to fit the real images' frequency spectra in terms of a proposed cloaking score as well as in terms of the performance of frequency-based detection methods of generated images [DKK20]. We make the following contributions:

- We propose to learn to generate images with a higher fidelity to the real images' frequency distribution by employing a discriminator that acts on the frequency spectra.
- The proposed discriminator is efficient and modular and can be trained stably with different GAN losses.
- We propose a measure for the spectral distribution fidelity which allows to assess how well generated images can be distinguished from real ones by their frequency spectra.
- We show in various experiments that the proposed approach enables generating images with realistic frequency spectra and therein outperforms the recent method from Durall, Keuper, and Keuper [DKK20] without sacrificing image quality in terms of Fréchet Inception Distance (FID).

6.2 Related Work

Generative Adversarial Networks. Generative networks have recently been successful in a wide range of applications, such as the generation of photo-realistic images at high resolution [Kar+17; BDS19; KLA19; Kar+20b] to style transfer [Iso+17; Zhu+17a; Zhu+17b; Hua+18], and more generally in image-to-image translation [Pat+16; ISI17; Zhu+17b; Cho+18; MCS19; KLA19] and text-to-image translation [Ree+16; Dai+17a; Zha+17b; Zha+18b]. GANs [Goo+14] play a crucial role in this context. They aim to approximate a latent-space model of the underlying data distributions from training images in a zero-sum game between a generator and a discriminator network. From this latent data distribution model, new samples can be generated by sampling. Recent works towards improving GANs proposed different loss functions, regularizations, or latent space constraints [Gul+17a; Mao+17; Gul+17a; Miy+18; MO14; DKD17; GSV17; DKK20; BDS19; Kod+17] to improve training stability and aim at high image resolutions [Kar+20b].

Image Synthesis in the Frequency Domain. Generated images are hard to distinguish from real images by the human eye. On the one end, automatically detecting generated images helps to protect content authenticity in the context of deep fakes. On the other hand, it can help to improve the generation process itself as it allows to find systematic mistakes currently made by image generation networks. One such systematic mistake seems to be especially apparent in the frequency spectra of generated images [Dur+20; Wan+20; Fra+20; Bai+20]. Surprisingly high detection rates can be achieved by feeding the Fourier transform (or the discrete cosine transform) of generated images into a deep network [ZKC19] (or simpler learning models such as support vector machines [Dur+20] or ridge regression [Fra+20]). As analyzed, for example, in Durall et al. [Dur+20] and Frank et al. [Fra+20], these systematic artifacts in the frequency domain are an effect of the generation process itself, more precisely in the up-convolutions. In Durall, Keuper, and Keuper [DKK20], a regularization approach acting on the frequency spectra of generated images has been proposed, which supports the training process by penalizing whenever a generated image's spectrum deviates from the average spectrum of the real data. Our approach is related to Durall, Keuper, and Keuper [DKK20]. However, we argue that a pointwise regularization of all generated spectra w.r.t. the average spectrum of real images is suboptimal since it does not properly allow to learn the data distribution. Instead, we propose to use a discriminator on the power spectra in order to learn the generation of images according to both, the distribution of the real data in spatial as well as in frequency domain.

6.3 Spectral Properties of Image Generation

Generative neural network architectures, like GANs, create high-dimensional outputs (such as high-resolution images) from low-dimensional latent space samples. Therefore, they rely on stepwise up-scaling mechanisms which successively increase the output resolution, followed by convolutional layers. Such upsampling can be done for example using "bed of nails" [DV18], nearest neighbor or bilinear interpolation, all of which have different effects on the properties of the resulting upsampled feature map or image. The spectral properties of an image $\mathbf{x}$ can be analyzed by its discrete Fourier transform as follows:

$$\mathbf{x}^{\mathcal{F}}_{[k,\ell]} = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} e^{-2\pi i \cdot \frac{m \cdot k}{M}} e^{-2\pi i \cdot \frac{n \cdot \ell}{N}} \cdot \mathbf{x}_{[m,n]}, \tag{6.1}$$
$$\text{for} \quad k = 0, \ldots, M-1, \quad \ell = 0, \ldots, N-1,$$

which transforms a 2D image $\mathbf{x}$ into a 2D array $\mathbf{x}^{\mathcal{F}}$ of its spatial frequency components. During upsampling, the frequency spectrum of an image is altered depending on the upsampling method. While bilinear interpolation results in smooth images with few high frequency components, the more commonly used "bed of nails" interpolation upsampling, which fills the missing values with zeros, initializes the upsampled image (or feature map) with many high frequency components. Durall, Keuper, and Keuper [DKK20] provide a theoretic analysis of the effect of upsampling using bed of nails interpolation, which we summarize below.

6.3.1 Spectral Effects of Upsampling

For simplicity, Durall, Keuper, and Keuper [DKK20] consider in their analysis the case of one-dimensional signals, which generalizes to higher dimensions. For a signal $\mathbf{a}$ the discrete Fourier transform $\mathbf{a}^{\mathcal{F}}$ is given by

$$\mathbf{a}_k^{\mathcal{F}} = \sum_{j=0}^{N-1} e^{-2\pi i \cdot \frac{jk}{N}} \cdot \mathbf{a}_j, \quad \text{for} \quad k = 0, \dots, N-1. \tag{6.2}$$

Let $\acute{\mathbf{a}}$ be the version of $\mathbf{a}$ that is upsampled by a factor of 2, then:

$$\begin{aligned} \acute{\mathbf{a}}_{\bar{k}}^{\mathcal{F}} &= \sum_{j=0}^{2\cdot N-1} e^{-2\pi i \cdot \frac{j\bar{k}}{2\cdot N}} \cdot \acute{\mathbf{a}}_j \\ &= \sum_{j=0}^{N-1} e^{-2\pi i \cdot \frac{2\cdot j\bar{k}}{2\cdot N}} \cdot \mathbf{a}_j + \sum_{j=0}^{N-1} e^{-2\pi i \cdot \frac{2\cdot (j+1)\bar{k}}{2\cdot N}} \cdot \mathbf{b}_j, \end{aligned} \tag{6.3}$$

for $\bar{k} = 0, \dots, 2N-1$, where $\mathbf{b}_j = 0$ for "bed of nails" interpolation (and $\mathbf{b}_j = \mathbf{a}_j$ for nearest neighbor interpolation). The first term in Equation 6.3 is similar to the original Fourier Transform while the second term is zero for $\mathbf{b}_j = 0$. It can be seen that when the spatial resolution is increased by a factor of 2, the frequency axes are scaled by $1/2$. From sampling theoretical considerations [DKK20], it is

$$(6.3) = \sum_{j=0}^{2\cdot N-1} e^{-2\pi i \cdot \frac{j\bar{k}}{2\cdot N}} \cdot \sum_{t=-\infty}^{\infty} \acute{\mathbf{a}}_j \cdot \delta(j - 2t). \tag{6.4}$$

The point-wise multiplication with the Dirac impulse comb removes exactly the values for which $\acute{\mathbf{a}} = 0$. In order to apply the convolution theorem [Kat04], one has to assume $\mathbf{a}$ being a periodic signal. Then, it is

$$\begin{aligned} \acute{\mathbf{a}}_{\bar{k}}^{\mathcal{F}} &= \frac{1}{2} \cdot \sum_{t=-\infty}^{\infty} \left(\sum_{j=-\infty}^{\infty} e^{-2\pi i \cdot \frac{j\bar{k}}{2\cdot N}} \acute{\mathbf{a}}_j \right) \left(\bar{k} - \frac{t}{2} \right) \\ &\overset{(6.3)}{=} \frac{1}{2} \cdot \sum_{t=-\infty}^{\infty} \left(\sum_{j=-\infty}^{\infty} e^{-2\pi i \cdot \frac{j\bar{k}}{N}} \cdot \mathbf{a}_j \right) \left(\bar{k} - \frac{t}{2} \right). \end{aligned} \tag{6.5}$$

Thus, the frequency spectrum $\acute{\mathbf{a}}^{\mathcal{F}}$ will contain replica of the frequency spectrum of $\mathbf{a}$. More precisely, all frequencies beyond $N/2$ are upsampling artifacts which can only be removed if the upsampled signal is smoothed appropriately.

In addition to these theoretic considerations, Durall, Keuper, and Keuper [DKK20] also show practically that the correction of the resulting spectra is not possible with the commonly used 3×3 convolutional filters.

6.3.2 Analysis of Real Data Distribution

In order to analyze the distributions of real and generated images' frequency spectra, we consider an aggregate representation as in Durall, Keuper, and Keuper [DKK20]. Assuming square images, we compute the magnitude of the 2D spectral frequencies and integrate over circles of constant radius in the frequency domain to obtain a 1D profile of the power spectrum, commonly referred to as the *azimuthal integral*:

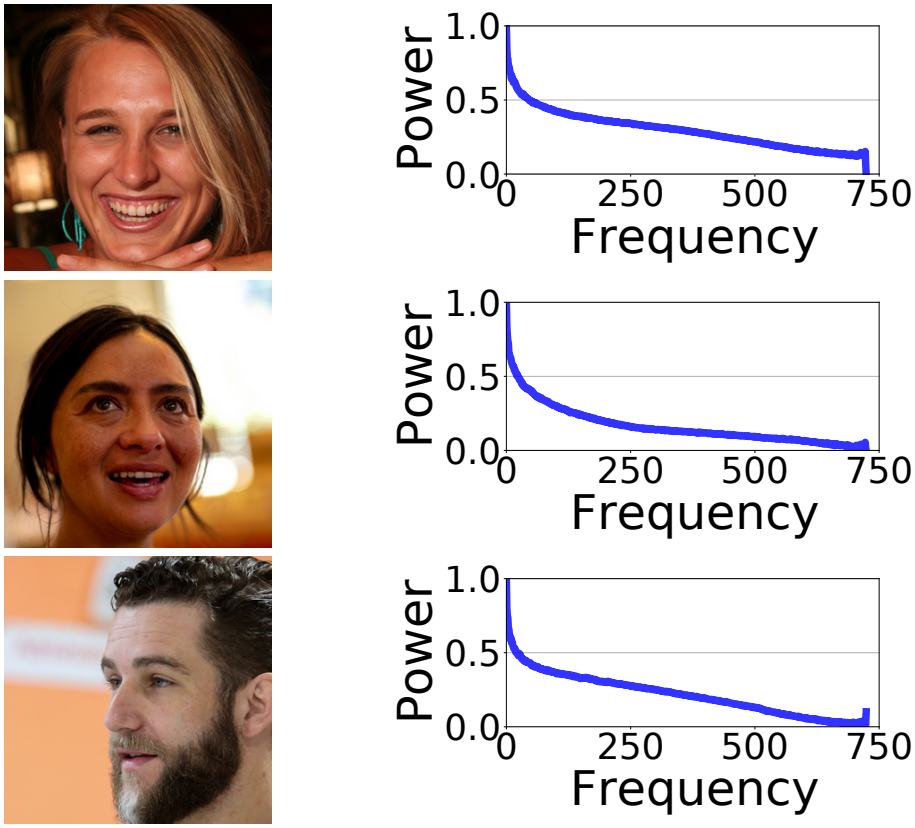
$$AI(\omega_k) = \int_0^{2\pi} \| \mathbf{x}^{\mathcal{F}} \left(\omega_k \cdot \cos\left(\phi\right), \omega_k \cdot \sin(\phi) \right) \|^2 \mathrm{d}\phi$$
$$\text{for} \quad k = 0, \ldots, M/2 - 1\,, \tag{6.6}$$

where ω_k denotes the radial spatial frequency. As pointed out in Durall, Keuper, and Keuper [DKK20], this notation is abusive for discrete $\mathbf{x}^{\mathcal{F}}$. In practice, the integral is implemented as sum over interpolated values. Figure 6.1 (a) shows examples of real images from the Flickr-Faces-HQ (FFHQ) [KLA19] dataset and their corresponding frequency profiles computed using Equation 6.6.

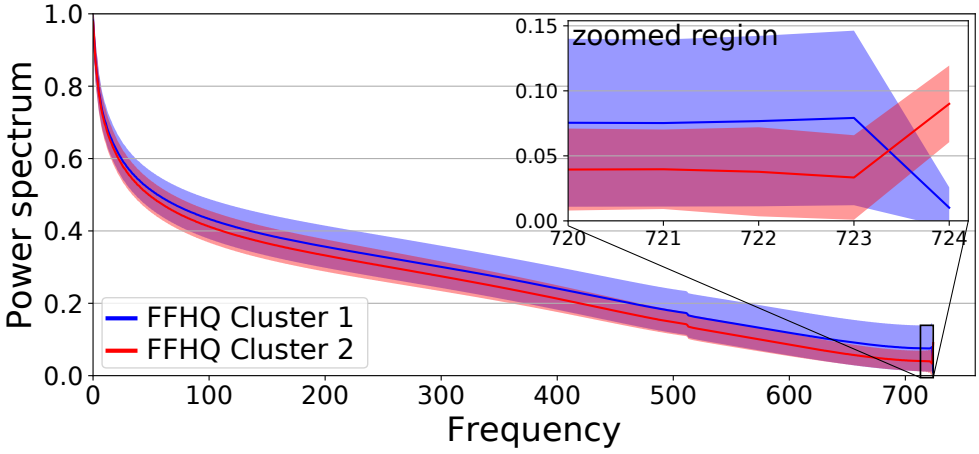
From these examples, one can see that the frequency profiles of such images are diverse and can vary substantially, especially in the high frequency regime. Figure 6.1 (b) shows the average frequency profiles after clustering the images of FFHQ by the magnitude of their highest frequency. The profiles can be well clustered into two groups just by looking at the highest frequency, which indicates that the frequency distribution of the real data cannot be well approximated by a uni-modal Gaussian distribution as done by Durall, Keuper, and Keuper [DKK20]. Since the true distribution is unknown, we argue that a suitable way of generating images with a higher spectral fidelity is to use a learned regularization approach. Our model therefore contains an additional discriminator, taking as input the frequency profile of real and generated images.

6.3.3 Evaluation in the Frequency Domain

In general, the evaluation of the quality of generated images by GANs is highly subjective. Therefore, Heusel et al. [Heu+17] introduced FID to provide a method of comparing different image generating models. Since its proposal, this quality measure is widely adopted as one of the key indicators to proof the qualitative performance of GANs. At the time of its publication, StyleGAN2 [Kar+20b] was the best performing model trained on the FFHQ dataset according to FID. Subjectively, generated images by StyleGAN2 are hard to distinguish from real images. This is reflected by a low FID score of 2.84 ± 0.03. However, recent advances in deep fake detection [Dur+20; DKK20; Fra+20] showed that it is possible to recognize such images using frequency domain representations. Figure 6.2 shows the averaged profiles of a real data distribution, represented by FFHQ, and the corresponding images drawn from the learned distribution by StyleGAN2. It is visible that the power spectrum is misaligned throughout most frequencies. Images produced by StyleGAN2 contain high frequency components, which can be observed by a rapid increase in the power spectrum of the last frequencies. Such behavior indicates the presence of grid artifacts and high frequency noise. Thus, we argue that generated images should not only be assessed by FID but also by their spectral distribution fidelity and propose a supplementary metric to evaluate this alignment in the frequency domain.



(a) Example images from FFHQ with their respective frequency profiles.



(b) Clustering of FFHQ spectral profiles.

Figure 6.1: **(a)** The frequency profiles from the FFHQ images are diverse, suggesting that their distribution is not uni-modal. **(b)** Average spectral profiles of the real data (FFHQ) after clustering with k-means (k=2). In the highest frequencies, the profiles can be well separated in two clusters.

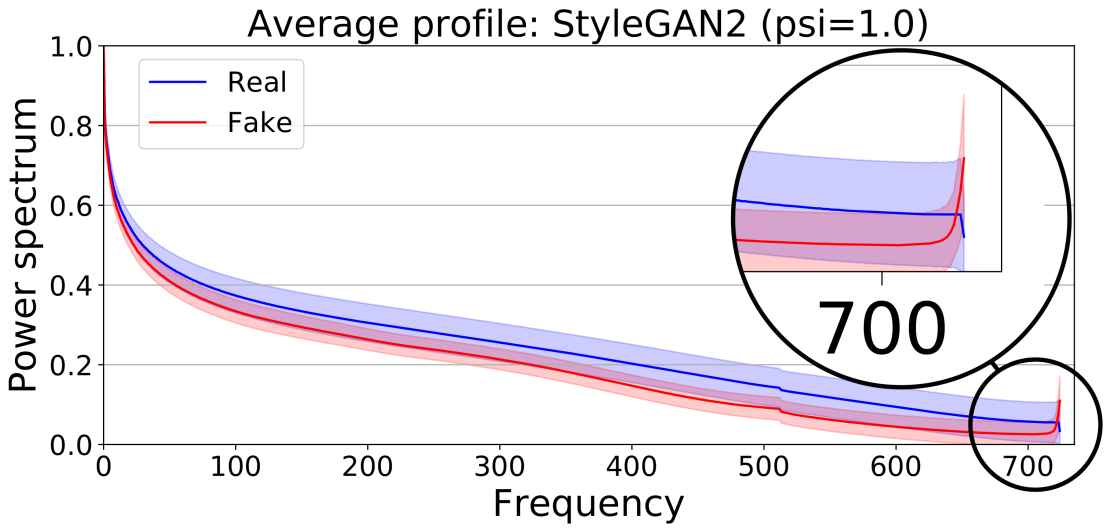


Figure 6.2: Average spectral profiles of real data (FFHQ) and data generated by StyleGAN2 [Kar+20b]. High frequency components in the generated images indicate grid artifacts or noise, which was not removed by the discriminator.

For this, we transform images into spectral profiles (Equation 6.6), and then train a logistic regression on profiles given by all images taken from the real distribution, and a corresponding number of generated images given by a model. We measure how close the model is to random guessing via:

$$\text{Cloaking Score} = 1 - 2 \cdot |\text{Accuracy} - 0.5|. \tag{6.7}$$

The Cloaking Score (CS) ranges in $[0, 1]$, where a score of 1.0 indicates perfect spectral alignment, and a score of 0.0 indicates that generated images can be linearly separated from real images in the spectral domain. For the example in Figure 6.2, the CS is 0.042 after 1000 epochs, 0.022 after 10 000 epochs, and 0.018 after 40 000 training epochs. Since this evaluation method gives rise to a trade-off between preciseness and runtime, we settled for 1000 epochs. In our experiments the runtime for $140k$ 64^2-sized images is around 3 minutes when all images are read from disk. We provide training details for the involved logistic regression in Section D.3 in Appendix D.

6.4 Learning to Regularize Spectral Distributions

Generative adversarial networks are trained in a minimax game between generator and discriminator networks, where the discriminator wants to recognize generated images and the generator wants the discriminator to perform poorly, i.e. to generate images which the discriminator can not tell apart from real ones, leading to objectives such as [Goo+14]:

$$\min_G \max_D V(D, G) = \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}(\mathbf{x})}[\log(D(\mathbf{x}))] + \mathbb{E}_{\mathbf{h} \sim p_{\mathbf{h}}(\mathbf{h})}[\log(1 - D(G(\mathbf{h})))].$$

Durall, Keuper, and Keuper [DKK20] propose to add a penalty term, a *Spectral Regularization*, to the generator to reduce discrepancies between the real and generated spectral distributions. This regularization is implemented as a cross entropy loss on the Equation 6.6 profiles of the generator

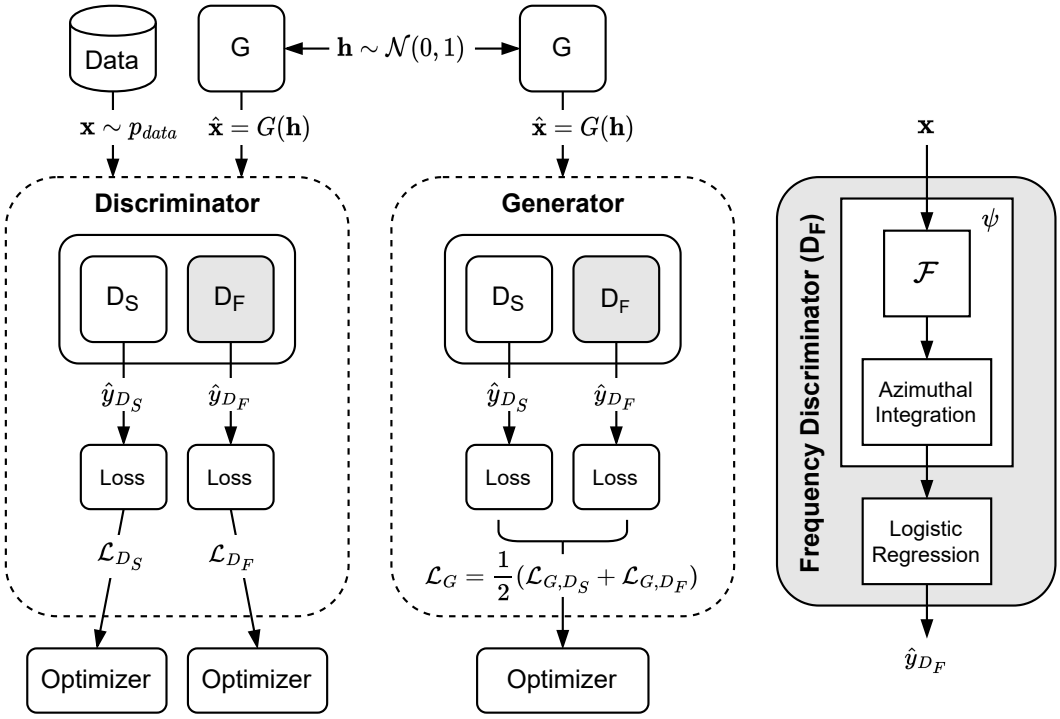


Figure 6.3: Training process of the proposed model. Losses are computed based on predictions representing the realness score of the given image in the spatial domain ($\hat{y}_{D_S}$) as well as the frequency domain ($\hat{y}_{D_F}$). Both spatial (D_S) and frequency (D_F) discriminators are trained separately by individual optimizers. When training generator G, both resulting losses are averaged.

output, to minimize the difference of the generated images' spectra to the mean spectrum of real images. Hence, the generator tries to minimize this penalty by forcing each image to reflect the same average profile. In our experiments, this leads to an unstable training progress resulting in mode collapse (we discuss this later). As we have shown above in the example for FFHQ, the real spectral distribution of a dataset is not necessarily a uni-modal Gaussian distribution around the average profile. Instead, there are certain characteristics that the generator needs to be able to learn. The regularizer by Durall, Keuper, and Keuper [DKK20] seems suboptimal in this respect. Therefore, we argue that instead of learning one average profile, the generator should be taught to generate images according to the real data distribution in spatial as well as in spectral domain.

We propose to use a second discriminator network (D_F) for this purpose (see Figure 6.3). While one could obviously consider to use full 2D power spectra and a convolutional architecture as in the original discriminator, we argue that an additional discriminator should be as lightweight and modular as possible. The proposed frequency spectrum discriminator D_F takes as input real or generated images which then go through a spectral transformation layer ψ that computes the magnitude of their Fourier transform. Afterwards, the azimuthal integral (Equation 6.6) is computed, projecting the 2D spectrum onto a 1D vector. Then, we add a fully connected layer with Sigmoid activation function, which is trained to discriminate between real and generated images.

Such a simple discriminator can in principle be trained with any of the commonly used GAN loss functions. To investigate the dependency of the discriminator performance on the employed

loss, we consider models with the commonly used loss functions in Table 6.1 and focus on simple architectures (i.e. Deep Convolutional Generative Adversarial Network (DCGAN) [RMC15], Least Squares Generative Adversarial Network (LSGAN) [Mao+17], Wasserstein Generative Adversarial Network (WGAN) [ACB17], and Wasserstein Generative Adversarial Network with Gradient Penalty (WGAN-GP) [Gul+17a]).

We train both the spatial (D_S) and the frequency (D_F) discriminators separately, applying the same loss function (e.g. BCE in case of DCGAN). For training generator G, both resulting losses are combined by averaging (see Figure 6.3). Additionally, we keep the proposed architectural suggestions by Durall, Keuper, and Keuper [DKK20], namely increasing the kernel size of the last upconvolutional layer to 8×8, and adding an additional block of three 5×5 convolutional layers to the end of the generator network. This, as they argue empirically, is necessary to provide the network with the capacity to "repair" the spectral artifacts.

Implementation Details. The memory consumption of the proposed discriminator depends on the image width and height and, assuming square images where $W = H$, adds only $\lceil W/\sqrt{2} \rceil + 1$ parameters. We evaluate it with different architectures. DCGAN, LSGAN and WGAN-GP were trained using Adam optimizer, for WGAN, we used RMSprop, with a learning rate of 0.0002 and a batch size of 128 for 500 epochs. In all cases, we apply the same loss function on both discriminators.

Table 6.1: Investigated discriminator losses.

$\mathcal{L}_{\text{DCGAN}}$	$-\mathbb{E}_{\mathbf{x}}[\log(D(\mathbf{x}))] - \mathbb{E}_{\hat{\mathbf{x}}}[\log(1 - D(\hat{\mathbf{x}}))]$
$\mathcal{L}_{\text{LSGAN}}$	$-\mathbb{E}_{\mathbf{x}}[(D(\mathbf{x}) - 1)^2] + \mathbb{E}_{\hat{\mathbf{x}}}[D(\hat{\mathbf{x}})^2]$
$\mathcal{L}_{\text{WGAN}}$	$-\mathbb{E}_{\mathbf{x}}[D(\mathbf{x})] + \mathbb{E}_{\hat{\mathbf{x}}}[D(\hat{\mathbf{x}})]$
$\mathcal{L}_{\text{WGAN-GP}}$	$\mathcal{L}_{\text{WGAN}} + \lambda \mathbb{E}_{\hat{\mathbf{x}}}\left[(\|\nabla D(\alpha \mathbf{x} + (1-\alpha)\hat{\mathbf{x}})\|_2 - 1)^2\right]$

6.5 Experiments

Experiments are conducted on the FFHQ dataset, which contains 70 000 high-quality face images at 1024×1024 resolution showing large variations in terms of age, ethnicity, and backgrounds. From this dataset, we downsample three versions in resolution 64×64, 128×128, and 256×256. We evaluate all experiments on $10k$ examples in terms of FID and the proposed CS. If the distribution of the real data is matched in both spatial and frequency domain, the resulting FID should be low while CS should be high. Additionally, we report the sum of absolute differences between the average profile of $10k$ generated images with the average profile of all real images, denoted as Spectral Difference (SD). This measure is similar to the regularization in Durall, Keuper, and Keuper [DKK20].

Architectural Adjustment. First, we validate the impact of adding additional convolutional layers with filter sizes of 5×5 as suggested by Durall, Keuper, and Keuper [DKK20] on DCGAN and LSGAN (see Table 6.2). In both cases, the modification leads to an improved FID and a lower spectral difference. Yet, the cloaking score is even lower than the one of the baseline approach, indicating that the generated distributions are still linearly separable from their integrated frequency spectra (Equation 6.6). This leads to two observations: First, increasing the amount

of convolutions at the highest resolution can increase image quality of simple GAN methods. Second, the conventional spatial discriminator does not provide the necessary signal to bring the spectral distributions closer together. From these observations, we continue with the modified networks to investigate whether the proposed spectral discriminator can provide the necessary training signal, and is able to use the network capacity to assimilate the generated images' frequency spectra to the real distribution.

Effect on Training Stability. We compare the training stability of the proposed method with the one from Durall, Keuper, and Keuper [DKK20]. We show in Figure 6.4 (top) the FID over 500 training epochs for five training runs the model from Durall, Keuper, and Keuper [DKK20]. Two out of five of these runs collapsed before reaching epoch 200. On the other hand, in Figure 6.4 (bottom), we show the same experiment for the proposed model, where not only the training runs stably but also the resulting FID is lower.

Comparison of Loss Functions. To evaluate the proposed spectral discriminator D_F in the context of diverse losses, we train variants of DCGAN [RMC15], LSGAN [Mao+17], WGAN [ACB17], and WGAN-GP [Gul+17a] with and without D_F. While the modified DCGAN and LSGAN models can generate images at up to 256^2 pixel resolution, the training behavior of the Wasserstein GANs becomes easily unstable. Thus, we only consider them until 128^2 pixels resolution. Additionally, to evaluate scalability to high resolutions, we finetune a pretrained state-of-the-art model, StyleGAN2 [Kar+20b], with additional D_F.

The resulting FID over the training epochs as well as the spectral differences are displayed in Figure 6.5 (a) and Figure 6.5 (b), respectively. It can be seen that the proposed discriminator has a relatively small impact on the FID during training while the spectral difference is substantially reduced by our method. The effect on the spectrum can be assessed in Table 6.3 (left block) and in Table 6.4. For all models and resolutions, the FID is on par with the plain version without additional discriminator while the spectral difference is considerably decreased and the cloaking score increased when D_F is added. The spectral regularization proposed by Durall, Keuper, and Keuper [DKK20] yields higher FID and lower cloaking score than the proposed approach.

Effect on Deep Fake Detection. In Table 6.3 (right block), we further evaluate the effect of the proposed discriminator on the detectability of the generated images using recent methods from Durall, Keuper, and Keuper [DKK20] and Wang et al. [Wan+20]. While Durall, Keuper, and Keuper [DKK20] leverage the 1D spectra in a simple Support Vector Machine (SVM) or Logistic Regression (LR) classifier and require retraining in every setting, Wang et al. [Wan+20] train a CNN on generated images as a "universal" detector. We evaluate the method from Durall, Keuper, and Keuper [DKK20] in two scenarios: First, in Table 6.3 (right block), indicated with the asterix, we train on the data without spectral discriminator and evaluate in the transfer setting. Second, we train separately for every setting. As can be seen, the detection is almost random in the transfer setting for the approach by Durall, Keuper, and Keuper [DKK20] as well as for our approach in all tested GAN settings and resolutions, while the generated images from [DKK20] can still be detected with about 80% accuracy when a model is trained specifically on this data. With respect to the universal detector by Wang et al. [Wan+20], both methods decrease the detection accuracy substantially. This confirms that the removed frequency artifacts can be perceived in the image domain and their removal is to be desired.

Table 6.2: Evaluation of the architectural change by Durall, Keuper, and Keuper [DKK20] on the image generation quality for DCGAN and LSGAN at 64^2 pixel resolution. In both cases, the changed upsampling and additional convolutional layers lead to an improved FID and Spectral Difference (SD), while the Cloaking Score (CS) is low.

Model	**FID** ↓	**SD** ↓	**CS** ↑
DCGAN_{orig}	17.749	1.510	**0.35**
DCGAN	**15.257**	**1.293**	0.16
LSGAN_{orig}	18.423	1.602	**0.28**
LSGAN	**15.518**	**0.468**	0.04

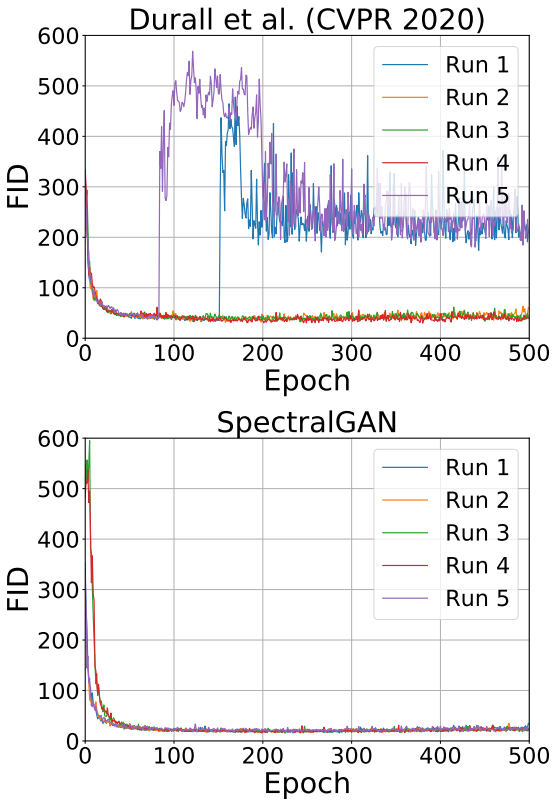
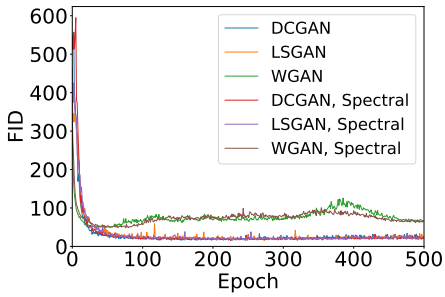
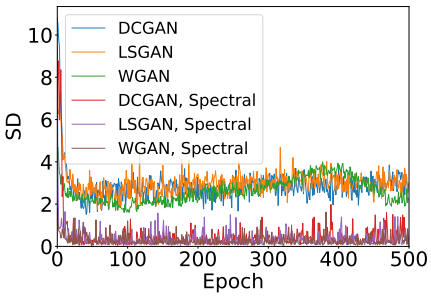


Figure 6.4: Comparison of the training stability of Durall et al. **(top)** and our proposed method **(bottom)**. The training was run for 500 epochs on 64^2 resolutions using the DCGAN loss. Our approach not only achieves lower FID values, but also results in more stable training runs.



(a) Resulting FID scores are similar, and the training is stable.



(b) Spectral differences are considerably reduced with the proposed discriminator D_F.

Figure 6.5: FID and SD with our models trained on FFHQ64. *Spectral* indicates that D_F was used, and omitted otherwise.

Table 6.3: Evaluation of the proposed discriminator in terms of GAN quality measures and generated image detection scores.

	GAN Quality			**Detection Accuracy**				
				Durall et al.*		Durall et al.		Wang et al.
Model	FID ↓	SD ↓	CS ↑	SVM ↓	LR ↓	SVM ↓	LR ↓	ACC ↓
				64^2				
DCGAN	**15.3**	1.3	0.16	0.89	0.89	0.89	0.89	0.81
DCGAN (D)	29.9	0.3	0.25	0.51	**0.49**	0.79	0.82	0.67
DCGAN (S)	15.6	**0.0**	**0.84**	**0.50**	0.50	**0.59**	**0.58**	**0.66**
LSGAN	15.5	0.5	0.04	0.94	0.94	0.94	0.94	0.67
LSGAN (S)	**15.5**	**0.0**	**0.86**	**0.51**	**0.50**	**0.55**	**0.56**	**0.64**
WGAN	**47.7**	1.3	0.01	0.99	0.99	0.99	0.99	0.79
WGAN (S)	47.9	**0.0**	**0.85**	**0.50**	**0.50**	**0.62**	**0.65**	0.79
WGAN-GP	**39.4**	0.6	0.18	0.92	0.94	0.92	0.94	0.76
WGAN-GP (S)	39.4	**0.3**	**0.54**	**0.51**	**0.51**	**0.75**	**0.76**	**0.65**
				128^2				
DCGAN	19.9	1.6	0.00	0.99	0.99	0.99	0.99	0.82
DCGAN (D)	41.9	0.4	0.08	**0.51**	0.51	0.98	0.98	0.87
DCGAN (S)	**19.9**	**0.1**	**0.72**	**0.51**	**0.50**	**0.66**	**0.69**	**0.81**
LSGAN	20.9	3.8	0.01	0.99	0.99	0.99	0.99	0.81
LSGAN (S)	**19.0**	**0.1**	**0.76**	**0.50**	**0.50**	**0.82**	**0.83**	**0.80**
WGAN	**20.8**	2.0	0.01	0.99	0.99	0.99	0.99	0.83
WGAN (S)	24.6	**0.1**	**0.66**	**0.52**	**0.53**	**0.61**	**0.60**	**0.81**
WGAN-GP	47.6	2.3	0.02	0.99	0.99	0.99	0.99	0.96
WGAN-GP (S)	**42.7**	**0.3**	**0.38**	**0.50**	**0.50**	**0.82**	**0.83**	**0.62**
				256^2				
DCGAN	**20.1**	9.8	0.00	1.00	1.00	1.00	1.00	0.96
DCGAN (D)	29.8	0.8	0.01	0.50	0.50	1.00	0.99	0.99
DCGAN (S)	20.9	**0.2**	**0.50**	**0.50**	**0.50**	**0.89**	**0.93**	**0.85**
LSGAN	19.9	6.0	0.00	1.00	1.00	1.00	1.00	0.85
LSGAN (S)	**19.9**	**0.2**	**0.46**	**0.50**	**0.50**	**0.77**	**0.82**	**0.72**

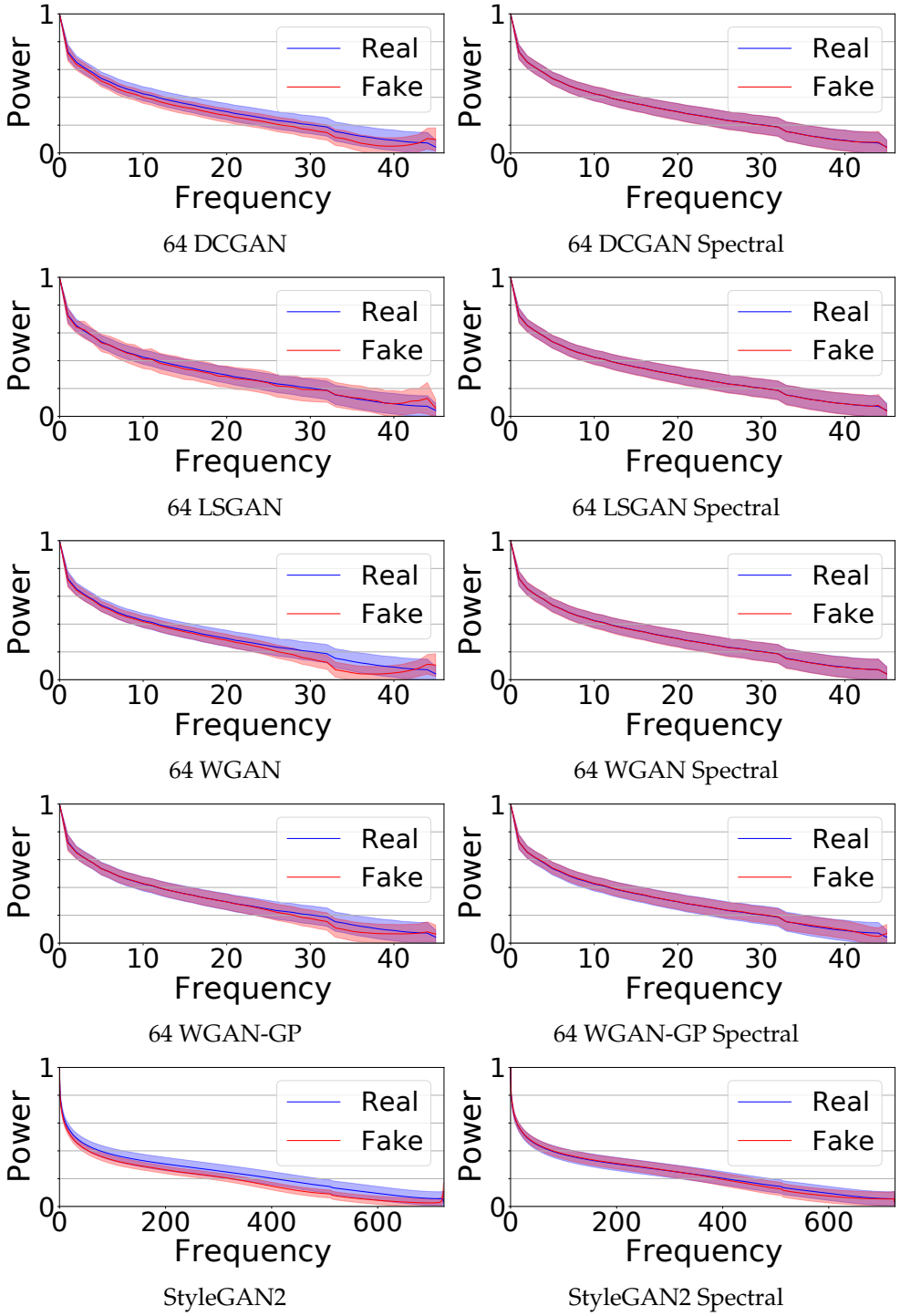


Figure 6.6: Resulting spectral profiles of experiments with our models trained on FFHQ64 and StyleGAN2. *Spectral* indicates that D_F was applied. With the spectral discriminator, the mean and standard deviation of the spectral profiles fit almost perfectly.

Table 6.4: Evaluation of the finetuned StyleGAN2.

Size	Model	GAN Quality FID ↓	SD ↓	CS ↑
1024^2	StyleGAN2	**2.73**	32.25	0.09
	StyleGAN2, Spectral	3.33	**5.98**	**0.24**

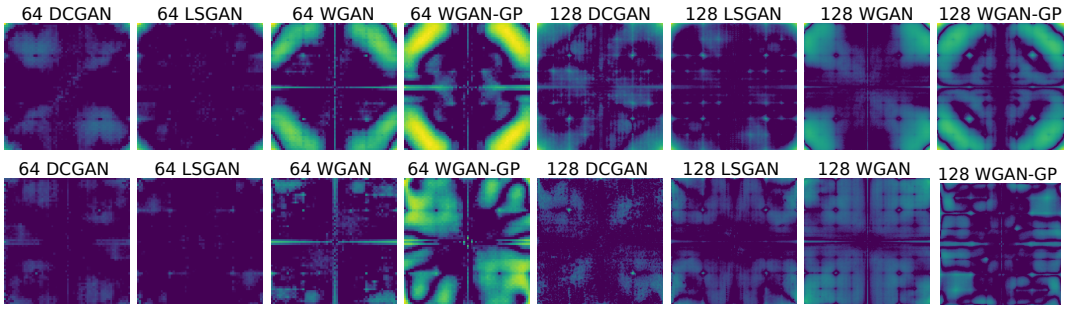


Figure 6.7: Mean absolute differences of 2D power spectra of real and generated images. (**top row**) Differences for the original, unregularized models, and (**bottom row**) differences for the models with spectral discriminator D_F. While the differences of the original models follow a characteristic pattern, our modified models show less specific differences and lower magnitudes.

Power Spectrum Visualizations. Figure 6.6 depicts a visualization of the resulting average frequency profiles with our approach as an overlay on the average frequency of real images. With the proposed discriminator, the average frequencies fit almost perfectly. Figure 6.7 shows the average differences of real and generated 2D power spectra in log-space (compare the visualization in Wang et al. [Wan+20]). It can be seen that all tested GANs show specific differences. When the proposed discriminator, acting on the 1D projection of the spectra, is employed, the differences become more diffuse and have lower magnitudes. This indicates that, on average, the generated images have less perceivable sampling artifacts. In Section D.2 of Appendix D we show a comparison to the method by Durall, Keuper, and Keuper [DKK20] in terms of 2D differences, as well as 1D differences for higher resolutions. Additionally, we show example images generated by our approach in Section D.4.

6.6 Conclusion and Outlook

Summary. In this chapter, we proposed an adversarial image generation approach that enables to generate images with a substantially reduced amount of spectral artifacts. The proposed method employs a simple discriminator on the 1D projections of frequency spectra of real and generated images. Thus, the generator aims to match the real data distribution not only in the image domain but also in the frequency domain. Our approach is very lightweight and allows for a stable training, as we experimentally show for different loss functions and resolutions. It generates images that can not easily be distinguished from real ones by frequency artifacts and improves, in this respect, over the recent method by Durall, Keuper, and Keuper [DKK20].

Effect of Architectural Design Choices. Durall et al. [Dur+20] suggest modifying the generator network architecture by adding additional convolutional layers and using larger kernel sizes to enable the network to better compensate for high-frequency components. We adopt these architectural changes and focus on improving the training signal by incorporating input from the frequency domain via a spectral discriminator. Schwarz, Liao, and Geiger [SLG21] show in their follow-up work that different upsampling operations inherently bias the generator towards specific spectral properties. In particular, nearest neighbor and bilinear upsampling tend to favor the generation of limited high-frequency content, which can actually be advantageous when working with natural images due to their typically exponential spectral decay. Conversely, zero insertion and reshaping methods introduce a tendency towards checkerboard artifacts. They also conclude that while the discriminator generally struggles with frequencies of low magnitude, high frequencies are not inherently more difficult to detect. However, they find that none of the convolutional-based discriminator architectures are able to provide artifact-free supervision in their experiments. This suggests that further research improving spatial discriminators may be necessary to eliminate the need for additional frequency-domain discriminators. Future research could investigate this aspect, for example by employing aliasing-free downsampling operations [Gra+22] in discriminator networks.

Spectral Properties of Diffusion Models. The suggested regularization in the spectral domain facilitates the fidelity of generated images in the spectral domain, as it was also confirmed by follow-up works [SLG21]. However, while providing a step into the right direction, it does not solve the problem completely. This can be seen in Figure 6.1 and was also confirmed by Schwarz, Liao, and Geiger [SLG21]. As argued earlier, one reason could be architectural biases. Since GANs were mostly succeeded by diffusion models [HJA20; DN21; Rom+22] for image synthesis in recent years, the question arises whether this model family inherently is more capable to provide spectral fidelity. Works in the realm of detection of diffusion-generated images [Bam24; Ric+24; Chu+25; Kar+25] hint towards the existence of spectral artifacts, whereas the specific type of artifacts seem to depend on the model in question [Cor+23]. Additionally, Ricker et al. [Ric+24] show that universal detectors, for example Wang et al. [Wan+20], trained on diffusion-generated images can still detect GAN-generated images, but not the other way around. Hence, albeit diffusion networks still contain artifacts, they are less prone than in GANs. However, generating artifact-free images seems to be still an open problem and requires more research to provide fidelity in the spectral domain.

1D vs. 2D Power Spectra. Our approach is focused on providing a lightweight method to improve spectral fidelity. As such, reducing the power spectra to 1D profiles offers the advantage of lowering computational overhead. However, this simplification likely leads to a loss of information that may limit the effectiveness of the spectral discriminator. Hence, the next natural step would be to extend our approach to the full 2D power spectrum to capture more detailed spectral characteristics. Follow up work [SLG21] indicates that using the full Fourier transform can indeed improve image fidelity, but scaling such methods to real-world scenarios remains challenging, particularly due to the increased complexity and computational demands associated with processing full 2D spectral information. Luo et al. [Luo+23] build on this idea by introducing a ViT-based architecture that transforms image patches to their 2D spectrum instead of the full image. This design enables the method to scale effectively to high-resolution images. They show that this approach improves over the 1D approach in the context of super-resolution tasks. However, this improvement comes at the cost of substantial increased computational complexity and an increase in the number of model parameters.

This page intentionally left blank.

Part III

Judge: Learning to Weight Data to Regularize Generative Models

In former parts of this thesis, we regularize the output of representation learning-based approaches either directly via penalty terms in Part I, or by matching extracted features between synthesized and training data in Part II. In this final part, we investigate data-based regularization approaches for generative models. For this, we augment the training data for a model with its own output and retrain it in such a way that its behavior aligns with preferences that we specify about the data. For this, each training data instance is judged by its importance, represented by a score that we provide. In Chapter 7, we apply this regularization method to optimize the representation space of discrete variational autoencoders. We demonstrate the effectiveness in guiding image synthesis by providing scores for the smiling degree of face images. Chapter 8 presents a generative neural architecture search approach that optimizes image classification architectures for multiple target attributes, like accuracy, hardware latency, and robustness to adversarial and common perturbations. This is supported by a newly introduced robustness dataset in Chapter 9.

This page intentionally left blank.

Chapter 7

Biasing Discrete Representations for Image Synthesis

Contents of This Chapter

Chapter Topic. This chapter is based on Jung et al. [Jun+23]. Here, we present a novel framework to regularize discrete variational autoencoders by weighting their training data. This method allows us to regularize the model by directly encoding preferences in terms of a score that we assign to training data instances. We specifically investigate regularizing the discrete representation space to synthesize images that conform to particular preferences. In particular, we choose the degree of smiling in face images as a proxy task. By utilizing tree-based models that are trained to predict the smiling degree, we can synthesize globally optimal training images. Augmenting the training dataset with these samples and retraining the model optimizes the representation space towards this attribute. We show that this data-driven regularization method improves the smiling degree substantially over the course of retraining the model.

Chapter Outline. We introduce this chapter in Section 7.1. In Section 7.2, we detail our optimization approach for discrete latent spaces. For this, we first summarize former work in discrete latent models. Then, we describe how we make use of tree-based models for global optimal data samples, and the weighted retraining strategy for latent space optimization. Experiments in Section 7.3 demonstrate the effectiveness of this method, particularly in the image synthesis task of generating face images with emphasis on their smiling degree. The chapter concludes with a concise summary in Section 7.4.

7.1 Introduction

MANY problems in science and engineering can be formulated as optimization of a costly-to-evaluate black-box function over high-dimensional or structured input domains. Notable examples are drug design, which typically requires expensive, time-consuming laboratory experiments for evaluation, neural architecture search, where every potential network solution with variable complexity must be trained and tested, or image synthesis, which can be framed as a black-box optimization problem whose objective function is a human judgment.

In the last few years, Latent Space Optimization (LSO) has been established [Góm+18], which tackles black-box optimization problems in a two-step procedure: First, a deep generative model is trained on the input data, and second, standard optimization methods such as Bayesian Optimization (BO) [Sha+16; BCF10] are used in the low-dimensional and continuous latent space learned by the model. Despite great successes in application fields such as chemical design [Góm+18; JBJ20] and automatic machine learning [Lu+18], LSO lacks in performance if the training data of the model mainly consists of low-scoring points and the true global optimum lies far away from this data [TDH20]. To address these weaknesses, Tripp, Daxberger, and Hernández-Lobato [TDH20] have proposed a method to boost the efficiency and performance of LSO by iteratively retraining an encoder-decoder-based model (e.g., a Variational Autoencoder (VAE)) [KW14]) on data points queried along the optimization trajectory and weighting those data points according to their objective value. This can be understood as an induced domain shift of a generative model.

By building on Tripp, Daxberger, and Hernández-Lobato [TDH20], this chapter demonstrates the applicability of such distribution shifts induced by weighted retraining on black-box optimization problems involving unstructured image data. In particular, we implement Tripp, Daxberger, and Hernández-Lobato [TDH20] on a generative model involving a discrete latent space, namely Vector Quantized Variational Autoencoder (VQ-VAE) [OVK17] which constitute very powerful yet simple to train alternatives to VAEs. In contrast to VAEs, VQ-VAEs can generate images at high quality without overly smooth details. Classical black-box optimization approaches such as BO are typically based on Gaussian processes [RW05] or neural networks [Sno+15] and thus assume fully continuous input domains. To allow for targeted optimization within the discrete latent space, this work transfers the LSO process from continuous to categorical input domains by utilizing tree-based ensembles as surrogate objective model in the latent space, encoding their predictions as mathematical optimization programs and solving those programs deterministically using state-of-the-art global solvers, see Figure 7.1. As we show in Section 7.3, the proposed framework improves results compared to continuous LSO via standard VAEs, and generates high-quality images that have substantially higher objective function values than the training data.

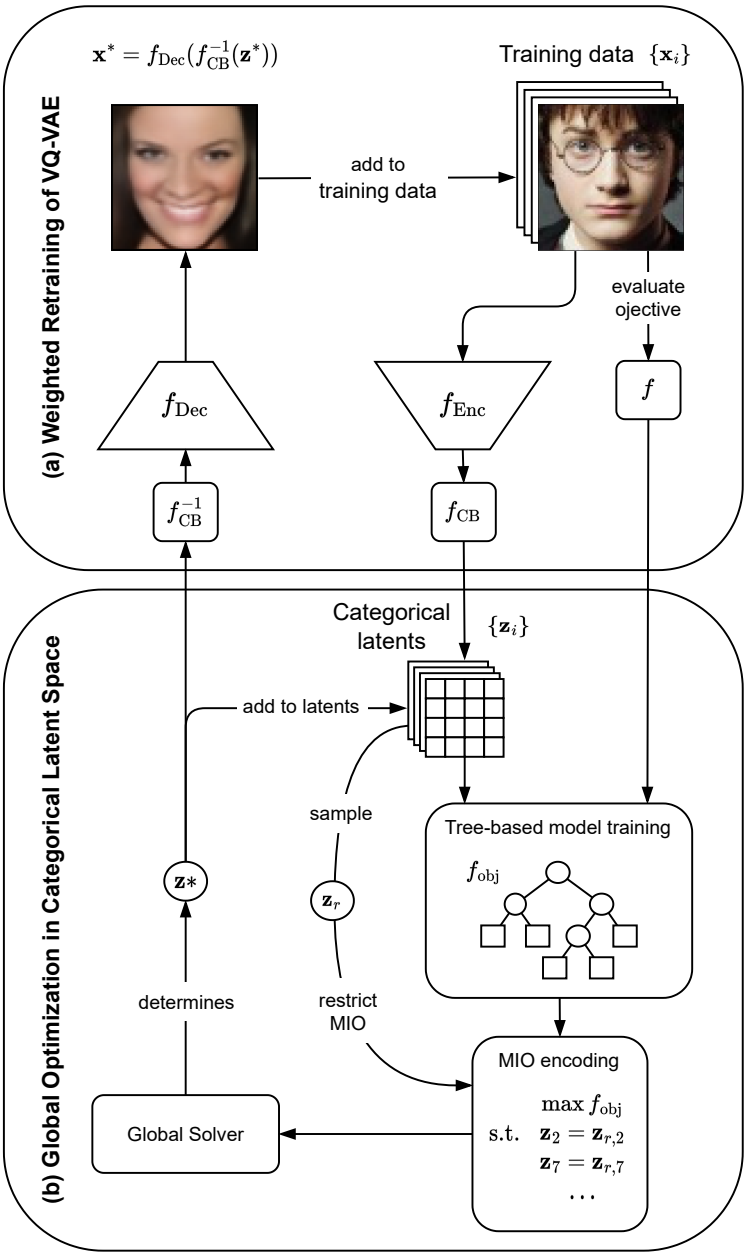


Figure 7.1: Proposed Framework. **(a)** The encoder and decoder of a VQ-VAE are initially pretrained and periodically finetuned on weighted training data. Images are encoded in a discrete latent space. **(b)** Optimization is performed in the categorical latent space: a tree-based ensemble model is learned from the categorical latent training data representations $\{\mathbf{z}_i\}$, encoded as a constrained mixed integer optimization problem, and globally solved to determine the next (optimal) query point $\mathbf{z}^*$ according to the black-box evaluation function f. This procedure is repeated r times before the VQ-VAE is finetuned on the queried data points. For this, the corresponding decoded images $\mathbf{x}^* = f_{\text{Dec}}(f_{\text{CB}}^{-1}(\mathbf{z}^*))$ are added to the training data and the weight for each training image is updated.

7.2 Discrete Latent Space Optimization

Although learning representations with continuous variables has been the focus of many previous works [Che+16a; DGF16; Vin+10], Oord, Vinyals, and Kavukcuoglu [OVK17] has demonstrated that discrete representations learned by VQ-VAEs capture important features of the data with a more natural fit for images. Thus, we provide a solution to transfer the LSO from continuous to discrete latent spaces in order to leverage the expressiveness of VQ-VAEs to generate high-scoring images with good quality. Then, we apply weighted retraining [TDH20] to induce a distribution shift in the generator distribution. These two steps can be alternated until convergence to train generators with desired target distributions.

7.2.1 Discrete Latent Variables

Here, we first recap the VQ-VAE model [OVK17] introduced in Subsection 2.2.3. The full VQ-VAE model consists of a discrete latent codebook $\{\mathbf{e}_k\}_{k=1}^K$, an encoder f_{Enc}, and a decoder f_{Dec}, where each codebook representation vector $\mathbf{e}_k \in \mathbb{R}^d$ is a learnable embedding and K is the total number of vectors. The shared codebook and encoder enable each input image $\mathbf{x} \in \mathbb{R}^{H \times W \times D}$ to be represented as a grid $\mathbf{z} \in \{1, \dots, K\}^{h \times w}$ of discrete latent variables. These are obtained via vector quantization from the encoder output $\hat{\mathbf{h}} = f_{\text{Enc}}(\mathbf{x}) \in \mathbb{R}^{h \times w \times d}$:

$$\mathbf{z}_{[i,j]} = \underset{k \in \{1,\dots,K\}}{\arg\min} \|\hat{\mathbf{h}}_{[i,j]} - \mathbf{e}_k\|_2^2,$$

where $[i, j]$ refers to the spatial location in the latent feature map. Here, $H \times W$ and $h \times w$ denote the spatial dimensions of the input image and the latent feature map, respectively. For the decoder, each latent variable is replaced with the latent representation from the codebook given the corresponding codebook entry. The decoder then reconstructs an image via:

$$\hat{\mathbf{x}} = f_{\text{Dec}}(\mathbf{h}) \quad \text{with} \quad \mathbf{h}_{[i,j]} = \mathbf{e}_{\mathbf{z}_{[i,j]}}.$$

7.2.2 Global Optimization in Discrete Latent Spaces

Our method performs optimization in the categorical latent space of a VQ-VAE, which is pretrained on some input dataset $\mathcal{D}$. To this end, a latent objective model $f_{\text{obj}}(\cdot)$ is constructed to approximate the black-box function $f(\cdot)$ at the output of the decoder $f_{\text{Dec}}(\cdot)$, i.e. $f_{\text{obj}}(\mathbf{z}) \approx f(f_{\text{Dec}}(\mathbf{z}))$ for all $\mathbf{z} \in \{1, \dots, K\}^{h \times w}$. Objective model f_{obj} can be trained by using the encoder f_{Enc} and the learned codebook f_{CB}, which together map every f-evaluated input image $\mathbf{x}$ to a corresponding categorical latent representation $\mathbf{z} = f_{\text{CB}}(f_{\text{Enc}}(\mathbf{x}))$.

When dealing with categorical feature spaces, tree-based ensemble models like random forests [Bre01] or gradient-boosting trees [Fri01] are popular choices for f_{obj}, since they naturally support various data types. Thus, we train a decision tree ensemble model to predict the objective value of a given image sample from its discrete latent representation. Interestingly, Thebelt et al. [The+21] propose an intricate approach that allows to encode the trained tree-based model with a Mixed-Integer Optimization (MIO) formulation, which allows the optimization of the discrete latent code w.r.t. the objective value, the tree-based model has been trained to predict. Please refer to Thebelt et al. [The+21] for the theoretical proof. By solving the resulting MIO program deterministically using a global solver [Kra88], f_{obj} can be optimized to determine the next latent query point $\mathbf{z}^*$.

During latent optimization, the solution $\mathbf{z}^*$ has to be restricted to stay sufficiently close to latent representations of the training data. Otherwise, without any constraints on the latent variables, the decoded version $\mathbf{x}^* = f_{\text{Dec}}(\mathbf{z}^*)$ of the query point $\mathbf{z}^*$ most likely has bad quality. This happens because values of those latent variables not having an impact on image regions that determine the objective function value can be chosen arbitrarily in the optimization process, which may lead to highly distorted image features. We address this problem as follows: First, a single training sample $\mathbf{x}_r$ is randomly drawn and mapped to its latent representation $\mathbf{z}_r$. Second, only those $t \in \mathbb{N}$ latent variables having the highest feature importance under the trained tree-based model are free to be globally optimized, while the remaining $h \cdot w - t$ variables are fixed to the respective entries of $\mathbf{z}_r$. Taking into account the optimization result $\mathbf{z}^*$ and function $f_{\text{CB}}^{-1}(\cdot)$ mapping from categorical variables to codebook representations, we can collect the corresponding objective function evaluation $f(\mathbf{x}^*)$ of its decoded version $\mathbf{x}^* = f_{\text{Dec}}(f_{\text{CB}}^{-1}(\mathbf{z}^*))$. Then, f_{obj} is refit after every iteration. We provide an ablation study on the number of free latent variables in Section 7.3.

7.2.3 Weighted Retraining

Subsection 7.2.2 introduces a technique to perform LSO in discrete latent spaces. However, as identified in Tripp, Daxberger, and Hernández-Lobato [TDH20], the underlying model does not necessarily learn a latent space that is amenable to efficient optimization of the objective function, especially in cases where the global optimum is far away from the training data. To resolve this issue, we follow Tripp, Daxberger, and Hernández-Lobato [TDH20] and weight the training data points $\{\mathbf{x}_i\}_{i=1}^N$ according to their objective values $\{f(\mathbf{x}_i)\}_{i=1}^N$: The higher a value $f(\mathbf{x})$ is, the more probability mass the training distribution should place on the corresponding input point $\mathbf{x}$. This weighting scheme requires assigning a weight $\omega_i \in \mathbb{R}$ with $\sum_{i=1}^N \omega_i = 1$. In this work, we adopt the rank-based function from Tripp, Daxberger, and Hernández-Lobato [TDH20].

To propagate information on new points acquired during the iterative LSO process to the VQ-VAE, where it could potentially help to uncover new promising regions that an optimization algorithm can exploit, the VQ-VAE is periodically finetuned after every $r \in \mathbb{N}$ LSO iterations.

7.3 Experiments

Here, we provide an empirical evaluation of the proposed discrete LSO with weighted retraining, as introduced in Section 7.2. First, we define a challenging optimization task using the face dataset CelebA [Liu+15]. Based on this image synthesis task, we evaluate the ability of our model to compete with continuous LSO via standard VAEs [TDH20].

Optimization Task. We employ 64×64 resolution face images from CelebA (see Section E.1 in Appendix E for further details) for an image synthesis task which can be viewed as black-box optimization problem. Our goal is to generate smiling faces by optimizing for the respective attribute degree in the space of colored 64×64 face images. To represent the smiling attribute on a continuous scale, we make use of the extended CelebA-Dialog dataset [Jia+21], which includes fine-grained labels for five selected CelebA attributes that cannot be accurately described by binary labels. In particular, the smiling attribute is divided into six levels $(0 - 5)$ that describe the degree in ascending order. For unseen face images, we estimate the degree of smiling by using a probability-weighted average $\hat{f}_{\text{smile}} \colon \mathbb{R}^{64 \times 64 \times 3} \to [0, 5]$ of class predictions coming from the ResNet-50 [He+16] classifier used in the official implementation of Jiang et al. [Jia+21], which is pretrained on CelebA-Dialog.

Table 7.1: Ablation study results. Top10 and Top50 scores are reported as mean final score $\pm$ one standard deviation after reaching the query budget of 500.

t	**FID**	**Top**10	**Top**50
4	42.69	2.72 ± 0.07	2.42 ± 0.04
8	41.69	2.85 ± 0.07	2.50 ± 0.05
16	43.70	2.88 ± 0.06	2.58 ± 0.02

We discard points with high objective values from the training data to make the problem more challenging and to represent the situation where the optimum (degree 5) lies far outside the training distribution. Specifically, *only images with a smiling degree* ≤ 2 *are kept in the training set.*

Implementation Details. Throughout this work, we assume a query budget of 500 and a retrain frequency of $r = 5$. For the underlying VQ-VAE, we use the Convolutional Neural Network (CNN) encoder-decoder architecture as in Oord, Vinyals, and Kavukcuoglu [OVK17] (see Section E.2 for further details). We assume a discrete latent space of size 8×8 (64 latent variables in total). Moreover, we consider $K = 256$ embedding vectors with a dimensionality of $D = 64$ each. The tree-based model f_{obj} is chosen to be an ensemble of 800 gradient-boosted regression tree and an interaction depth of 2, following Thebelt et al. [The+21]. The minimum number of samples in one leaf equals 20, and the maximum number of leaves per tree is set to 5.

Continuous LSO with VAEs is implemented by employing a neural network as latent objective model, as done by Snoek et al. [Sno+15] (see Section E.3 for further details). Motivated by Ghosh et al. [Gho+20], local optimization in the continuous latent space is constrained using ex-post density estimation via Gaussian Mixture Models, and new query points are restricted to stay sufficiently close to the estimated distribution. Optimization is performed using the SLSQP algorithm [Kra88].

Evaluation Metrics. Following common practice for BO [Sha+16; TDH20], we show the worst of the 10 and 50 best novel smiling degree predictions obtained up until query $m = 1, \ldots, 500$, which is denoted as Top10 and Top50 score function, respectively. To ensure statistical significance, the mean $\pm$ one standard deviation across 20 runs with different random seeds is reported. Furthermore, we use Fréchet Inception Distance (FID) scores [Heu+17] as quantitative assessment of the quality of generated images. Since our goal is to find faces that have a higher smiling degree than the best point in the training data, we compute FID scores between all 10 000 generated images from the 20 runs and the subset of CelebA faces having a smiling degree between 3 and 5.

Ablation Study on the Number of Free Variables. First, we measure the impact of the number t of free LSO variables on the performance of our proposed method. For this purpose, we test 4, 8, and 16 free variables for 20 optimization runs each. FID scores as well as final Top10 and Top50 scores after 500 optimization iterations are reported in Table 7.1. As we can observe, selecting $t = 8$ appears to offer an optimal trade-off: While generated faces have substantially lower quality if t is increased (comparing FID values of 41.69 vs. 43.70), a reduction of t leads to lower smiling degrees (according to Top10: 2.85 vs. 2.72). Hence, we set $t = 8$ for all further experiments.

Results on Smiling Face Synthesis. Figure 7.2 presents quantitative comparison between our approach and continuous LSO via standard VAEs. Corresponding visual results are shown in Figure 7.3. Even if weighted retraining is applied, the Top50 scores resulting from LSO with

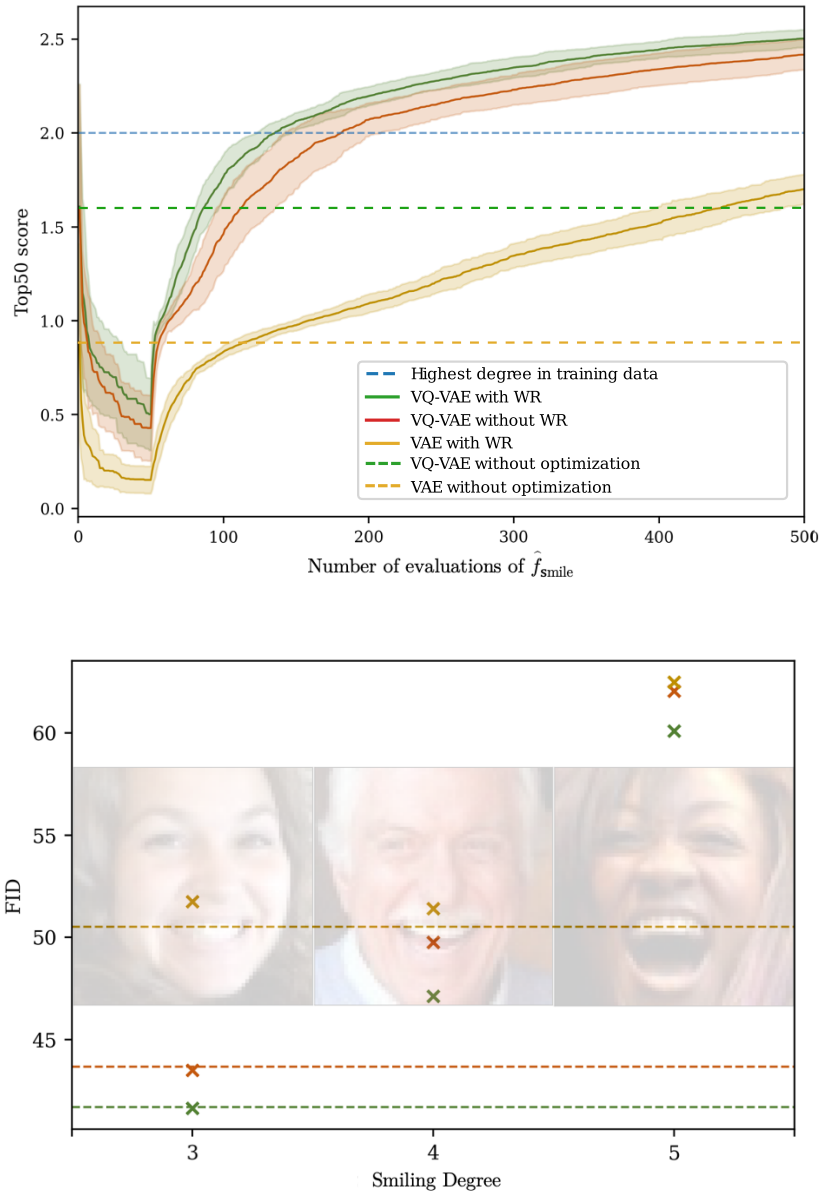


Figure 7.2: Quantitative comparison between discrete and continuous LSO variants: VQ-VAE with weighted retraining (green), VQ-VAE without weighted retraining (red), and VAE with weighted retraining (yellow). Evaluation metrics: **(top)** Top50 score function, where the blue dashed line depicts the best smiling degree in the training data. VQ-VAE as well as weighted retraining improve the degree of smiling substantially. **(bottom)** Fréchet Inception Distance (FID) scores, where the dashed lines show the respective scores of testing against the subset of CelebA images having an unseen smiling degree ($3-5$), and the markers show testing against each training data subset with the corresponding unseen smiling degree (3, 4, and 5; background images depict training data samples for these smiling degrees). In all cases, VQ-VAE with weighted retraining achieves the best result.

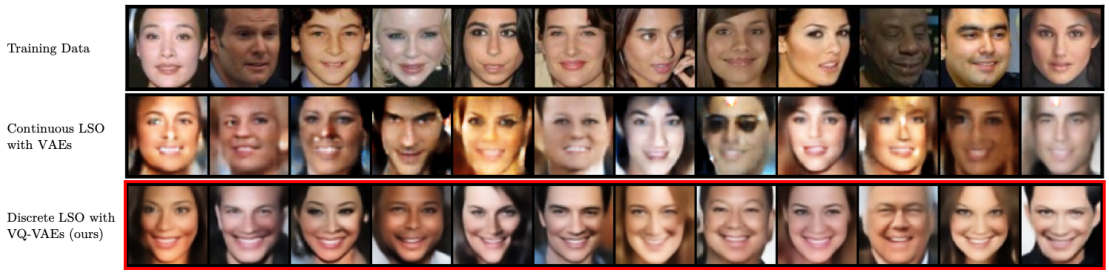


Figure 7.3: In contrast to previous LSO approaches (**middle**), our proposed method is able to synthesize smiling faces with high quality and less artifacts (**bottom**). Furthermore, the generated images have a substantially higher degree of smiling compared to the best points in the restricted training dataset (**top**, restricted to smiling degree ≤ 2) that was used to train the models. LSO is performed to maximize smiling degree, and FID to the target distribution (unseen smiling degrees $3-5$) improves from 50.51 to 41.69.

VAEs are strictly below the value 2.0 that corresponds to the best training images (mean final score: 1.70). Even without weighted retraining of the VQ-VAE, our method outperforms VAEs by a large margin (mean final score: 2.42). Weighted retraining further improves the final Top50 score from 2.42 to 2.50 on average.

Moreover, the FID results show that our method successfully leverages the expressiveness of VQ-VAEs to generate images with higher quality compared to standard VAEs. The FID score can be decreased by 21%, from 50.51 to 41.69. Again, weighted retraining hereby has a positive effect and leads to a substantial improvement of 5%. In addition, we tested against three separate subsets containing all images having a smiling degree of 3, 4, and 5, respectively, to detect potential biases among the generated faces. However, for all considered LSO variants, the results are as expected: higher smiling degrees in the target distribution lead to higher FID scores. Overall, the level of FID scores is relatively high, which can likely be attributed to the following reason: the target distribution (degrees $3-5$) is different from the distribution the VQ-VAE is pretrained on (degrees $0-2$), which prevents direct comparison to other generative models that are explicitly designed and trained to generate samples that resemble the given training data.

7.4 Conclusion and Outlook

We propose a method for efficient black-box optimization in the discrete latent space of VQ-VAEs, which combines (i) choosing a tree-based ensemble as latent objective model, (ii) encoding its predictions as an MIO problem that is solved globally to determine the next query point and (iii) iteratively finetuning the underlying VQ-VAE on weighted data. With the challenging task of generating smiling faces that are not contained in the training distribution, we demonstrate that our method notably outperforms continuous LSO with VAEs in terms of both image quality and optimization of the objective function. To the best of our knowledge, this is the first work that successfully applies LSO in discrete latent spaces on image synthesis tasks.

Scalability. Our experiments are performed on 64^2 resolution images. This has the welcome property that the discrete latent space of the VQ-VAE trained on this resolution is quite small having 64 latent variables (see Section E.2). Likewise, the continuous latent space optimization

with the VAE was also performed on 64 variables (see Section E.3). In comparison, stable diffusion-based networks [Rom+22] have $4k$ variables, modern successors even $16k$ [Ess+24]. We can rightfully assume that scaling the global optimization methods of selecting the next best latent point up to these large number of variables becomes infeasible. Future research could investigate how to mitigate this problem, for example by compressing the latent space further. Another idea could be to facilitate the learned predictor to identify which variables are most informative for the property to be optimized for. This could be done by fitting decision trees or linear models, or via interpretability methods in the case of neural networks.

Predictor Performance. One requirement for the method in this chapter to work is a well-performing predictor. For this we either need a pretrained predictor, or ground truth data to train a predictor on. In either case, the whole method relies (i) on the fact that the predictor has learned which attribute we want to optimize for, and (ii) that the predictors performs well. If this is not the case, because we cannot know how well the predictor performs (if we have no ground truth data), or the ground truth data is noisy, then our assumption about the property we want to encode and the property the predictor infers may not be aligned. This is especially pronounced in tasks where the property is hardly measurable and subjective (as for example with smiling degrees). This can have unexpected consequences, as the differences in results might be subtle. In essence, we have no guarantee that the model is optimized for what we expect. Future research could elaborate on this limitation, for example by considering calibrated methods [Guo+17] as predictors that provide confidence levels with their predictions. With these methods it would be possible to also incorporate the predictor's confidence into the weighting of training data, and thus mitigate some of the risk.

This page intentionally left blank.

Chapter 8
Biasing Generative Neural Architecture Search

Contents of This Chapter

Chapter Topic. This chapter is based on Lukasik*, Jung*, and Keuper [LJK22]. Here, we develop a generative approach for neural architecture search. For this, we utilize a graph neural network-based generator that learns representations over possible neural architectures. In a similar way as in Chapter 7, we regularize this representation space towards desired target properties of neural architectures via retraining of the generator on its own output during the search process. Architectures that are proposed by the model are trained and evaluated on a target task. The model is then regularized by assigning importance scores to training instances based on these results. Accuracy on image classification is the common optimization goal in existing neural architecture search benchmarks, and we outperform several other methods on multiple benchmarks when optimizing for only this property. Additionally, our approach is able to optimize for multiple desired properties by combining multiple scores. In particular, we combine scores for accuracy with hardware efficiency. In this multiobjective setting, we show the effectiveness of regularizing architecture representations in the proposed data-driven way.

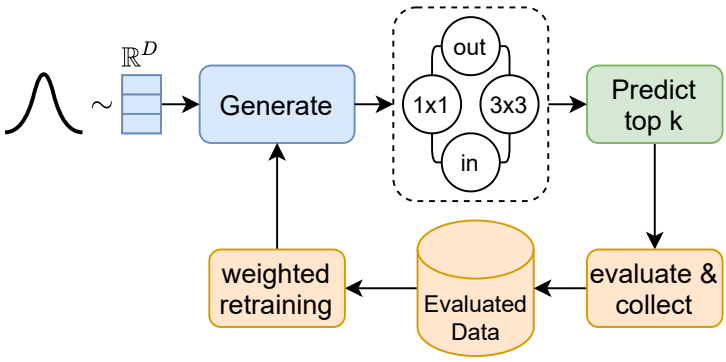
Chapter Outline. We introduce this chapter in Section 8.1, and summarize related work about neural architecture search and generative graph neural networks in Section 8.2. Then, we describe our proposed graph generative search process and the regularization method in Section 8.3. Experimental results on several neural architecture search benchmarks are presented in Section 8.4. Lastly, we conclude this chapter in Section 8.5.

8.1 Introduction

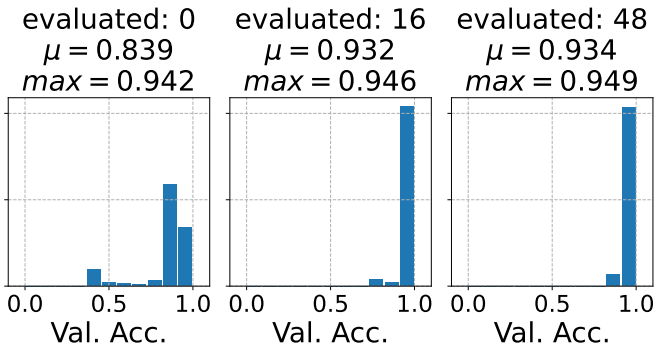
THE first image classification network [KSH12] applied to the large-scale visual recognition challenge ImageNet [Den+09] achieved unprecedented results. Since then, the main driver of improvement on this challenge are new architecture designs [SZ15; Sze+15; Sze+16; He+16] that ultimately lead to architectures surpassing human performance [He+15]. Since manual architecture design requires good intuition and a huge amount of trial-and-error, the automated approach of Neural Architecture Search (NAS) receives growing interest [Rea+17; Zop+18; Yin+19b; DY20; Kly+22; Li+21]. Well-performing architectures can be found by applying common search practices like random search [BB12], evolutionary search [Rea+17; Rea+19], Bayesian Optimization (BO) [Kan+18; Ru+21; WNS21a], or local search [WNS21b] on discrete architecture search spaces, such as NAS-Bench-101, NAS-Bench-201, DARTS and NAS-Bench-NLP [Yin+19b; DY20; LSY19; Kly+22]. However, these methods are inefficient because they require to evaluate thousands of architectures, resulting in impracticable search times. Recent approaches avoid this problem by training surrogate models to approximate the performance of an architecture [LSY19; CZH19] or by generating architectures based on learned architecture representation spaces [Zha+19; Luk+21]. Improving query efficiency is crucial in NAS, since every query implies a full training and evaluation of the neural architecture on the target dataset.

This trade-off between query efficiency and resulting high-scoring architectures is an active research field. Yet, no attempts were made so far to leverage the advantages of both search paradigms. Therefore, we propose a model that incorporates the focus of promising architectures already in the architecture generation process by optimizing the latent space *directly*: We let the generator learn in which areas of the data distribution to look for promising architectures. This way, we reduce the query amount even further, resulting in a query efficient and very effective NAS method. Our proposed method is inspired by a Latent Space Optimization (LSO) technique [TDH20], originally used in the context of variational autoencoders [KW14] to optimize generated images or arithmetic expressions using BO. We adapt this concept to NAS and pair it with an architecture performance predictor in an end-to-end learning setting, so that it allows us to iteratively reshape the architecture representation space. Thereby, we promote desired properties of generated architectures in a highly query-efficient way, i.e. by learning expert generators for promising architectures. Since we couple the generation process with a surrogate model to predict desired properties (such as high accuracy or low latency), there is no need in our method for BO in the generator's latent space, making our method more efficient.

In practice, we pretrain, on a target space of neural architectures, a Graph Neural Network (GNN)-based generator network, which does not rely on any architecture evaluation and is therefore fast and query-free. The generator is trained in a novel generative setting that directly compares generated architectures to randomly sampled architectures using a reconstruction loss without the need of a discriminator network as in Generative Adversarial Networks (GANs) [Goo+14] or an encoder as in Variational Autoencoders (VAEs) [KW14]. We use an MLP as a surrogate to rank performances and hardware properties of generated architectures. In contrast, previous generative methods either rely on training and evaluating supernets [HC21], which are expensive to train and dataset specific, or pretrain a latent space and search within this space directly using BO [Zha+19; Yan+20; Luk+21], Reinforcement Learning (RL) [Rez+21] or gradient based methods [Luo+18]. These methods incorporate either GANs, which can be hard to train or VAEs, which are biased by the regularization, whereas our plain generative model is easy to train. In addition we enable backpropagation from the performance predictor to the generator. Thereby, the generator can efficiently learn which part of the architecture search space is promising with only few evaluated architectures.



(a) Proposed search method.



(b) Architecture accuracy covering the representation space.

Figure 8.1: (**a**) Our search method generates architectures from points in an architecture representation space that is iteratively optimized. (**b**) The architecture representation space is biased towards better-performing architectures with each search iteration. After only 48 evaluated architectures, our generator produces state-of-the-art performing architectures on NAS-Bench-101.

By extensive experiments on common NAS benchmarks [Yin+19b; DY20; Zel+22; Kly+22; Li+21] as well as ImageNet [Den+09], we show that our method is effective and sample-efficient. It reinforces the generator network to produce architectures with improving validation accuracy (see Figure 8.1), as well as in improving on hardware-dependent latency constraints (see Figure 8.7) while keeping the number of architecture evaluations small. In summary, we make the following contributions:

- We propose a simple model that learns to focus on promising regions of the architecture space. It can thus learn to generate high-scoring architectures from only a few queries.
- We learn architecture representation spaces via a *novel generative design* that is able to generate architectures stochastically while being trained with a simple reconstruction loss. Unlike VAEs [KW14] or GANs [Goo+14], no encoder network nor discriminator network is necessary.
- Our model allows sample-efficient search and achieves state-of-the-art results on several NAS benchmarks as well as on ImageNet. It allows joint optimization w.r.t. hardware properties in a straightforward way.

8.2 Related Work

Neural Architecture Search. Intrinsically, NAS is a discrete optimization problem seeking the optimal configuration of operations (such as convolutions, poolings, and skip connections) in a constrained *search space* of computational graphs. To enable benchmarking within the NAS community, different search spaces have been proposed. The tabular benchmarks NAS-Bench-101 [Yin+19b] and NAS-Bench-201 [DY20] provide both an exhaustive covering of metrics and performances. NAS-Bench-NLP [Kly+22] provides a search space for natural language processing. In addition to tabular benchmarks, NAS-Bench-301 [Zel+22] is a surrogate benchmark that allows for fast evaluation of NAS methods on the DARTS [LSY19] search space by querying the validation accuracy. NAS-Bench-x11 [Yan+21] is another surrogate benchmark providing full training information for each architecture in all four mentioned benchmarks. NAS-Bench-Suite [Meh+22] facilitates reproducible search on these NAS benchmarks.

Early NAS approaches are based on discrete encodings of search spaces, such as in the form of adjacency matrices, and can be distinguished by their *search strategy*. Examples are random search [BB12; LT19], RL [ZL17; Li+18b], evolutionary methods [Rea+17; Rea+19], local search [WNS21b], and BO [Kan+18; Ru+21]. Recent NAS methods shift from discrete optimization to faster weight-sharing approaches, resulting in differentiable optimization methods [Pha+18; LSY19; Ben+18; CZH19; Xie+19b; Zel+20]. Several approaches map the discrete search space into a continuous architecture representation space [Luo+18; Zha+19; Yan+20; Luk+21] and search or optimize within this space using, for example, BO (e.g. Yan et al. [Yan+20]) or gradient-based point operation [Luo+18]. In this chapter, we also learn continuous architecture representation spaces. However, in contrast to former works, we propose to optimize the representation space, instead of performing point optimization within a fixed space such as e.g. Luo et al. [Luo+18]. A survey of different strategies can be found in Elsken, Metzen, and Hutter [EMH19].

All NAS approaches are dependent on *performance estimation* of intermediate architectures. To avoid the computation heavy training and evaluation of queries on the target dataset, methods to approximate the performance have been explored [Whi+21]. Common approaches include neural predictors that take path encodings [WNS21a] or graph embeddings learned by GNNs [Shi+20; Wen+20] as input. WeakNAS [Wu+21] proposes to progressively evaluate the search space towards finding high-performing architectures using a set of weak predictors. In our method, we integrate a weak expert predictor with a generator to yield an efficient interplay between predicting for high-performing architectures and generating them.

Graph Generative Models. Most graph generation models in NAS employ VAEs [KW14]. Luo et al. [Luo+18] uses an LSTM-based VAE, coupled with performance prediction for gradient-based architecture optimization. Note that Luo et al. [Luo+18] optimizes the latent point in a fixed latent space while our approach optimizes the latent space itself. Zhang et al. [Zha+19] use GNNs with asynchronous message-passing to train a VAE for BO. Huang and Chu [HC21] combines a generator with a supernet and searches for neural architectures for different device information. Yan et al. [Yan+20] facilitates Xu et al. [Xu+19] with an MLP decoder. Lukasik et al. [Luk+21] proposes Smooth Variational Graph embeddings (SVGe) using two-sided GNNs to capture the information flow within a neural architecture.

Our proposed model's generator is inspired by SVGe with the aim to inherit its flexible applicability to various search spaces. Yet, similar to Yan et al. [Yan+20], due to the intrinsic discretization and training setting, SVGe does not allow for backpropagation. Recently, Rezaei et al. [Rez+21] facilitates GNNs in a GAN [Goo+14] setting, where the backpropagation issue is circumvented using reinforcement learning. In contrast, our proposed GNN generator circumvents

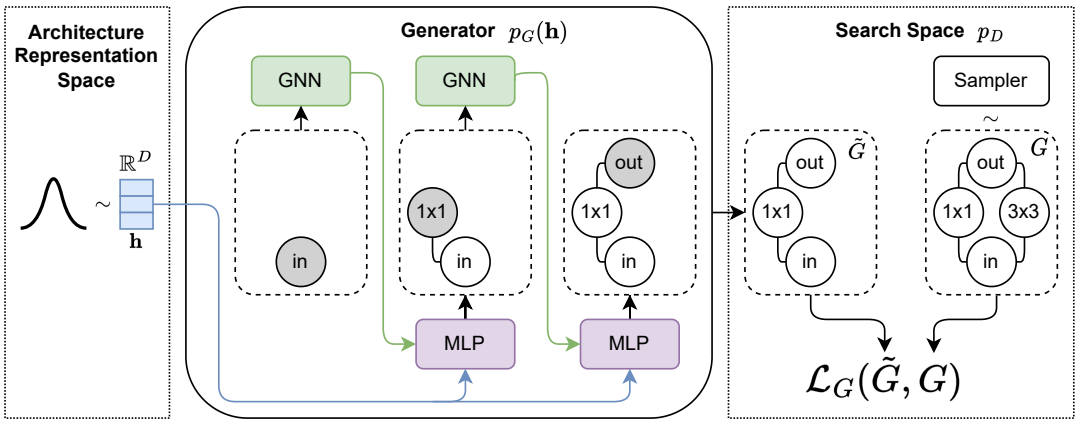


Figure 8.2: Representation of the training procedure for our generator in Architecture Generative Network (AG-Net). The input is a randomly sampled latent vector $\mathbf{h} \in \mathbb{R}^d$. First, the input node is generated, initialized and input to a GNN to generate a partial graph representation. The learning process iteratively generates node scores and edge scores using $\mathbf{h}$ and the partial graph representation until the output node is generated. The target for this generated graph is a randomly sampled architecture.

the intermediate architecture discretization and can therefore be trained by a single reconstruction loss using backpropagation. Its iterative optimization is inspired by Tripp, Daxberger, and Hernández-Lobato [TDH20], who proposes to use a VAE with weighted retraining w.r.t. a target function to adapt the latent space for the optimization of images and arithmetic functions using BO. Our model transfers the idea of weighted retraining to NAS. It uses our plain generator and improves sample efficiency by employing a differentiable surrogate model on the target function such that, in contrast to Tripp, Daxberger, and Hernández-Lobato [TDH20], no further black-box optimization step is needed. Next, we describe the proposed generator network.

8.3 Architecture Generative Model

Preliminaries. We aim to generate neural networks represented as Directed Acyclic Graphs (DAGs). This representation is in line with the cell-based architecture search spaces commonly used as tabular benchmarks [Yin+19b; DY20]. Each cell is a DAG denoted by $G = (V, E)$ with nodes $v \in V$ and edges $e \in E$. The graph representations differ between the various benchmarks in terms of their labeling of operations. For example, in NAS-Bench-101 [Yin+19b] each node is associated with an operation, whereas in NAS-Bench-201 [DY20] each edge is associated with an operation.

Generative Network. Commonly used graph generative networks are based on VAEs [KW14]. In contrast, our proposed network is a *purely generative* network, p_G (see Figure 8.2). To generate valid graphs, we build our model similar to the graph decoder from the VAE approach SVGe [Luk+21]. The generator takes a randomly sampled variable $\mathbf{h} \sim \mathcal{N}(0, 1)$ as input and reconstructs a randomly sampled graph from the cell-based search space. The model iteratively builds the graph: it starts with generating the input node v_0, followed by adding subsequent nodes v_i and their labels and connecting them with edges $e_{j,i}, j < i$, until the end node v_T with the label

output is generated. Additionally, we want to learn a surrogate for performance prediction on the generated data and allow for end-to-end training of both. To allow for backpropagation, we need to adapt several details of the generator model. We initialize the node-attributes for each node by one-hot encoded vectors, which are initialized during training using an MLP with two layers to replace the learnable look-up table proposed in SVGe. The output of our generator is a vector graph representation consisting of a concatenation of generated node scores and edge scores. It is important to note that the iterative generation process is independent of the ground truth data, which are only used as a target for the reconstruction loss. Note that the end-to-end trainability of the proposed generator is a prerequisite for our model: It allows to pair the generator with a learnable performance predictor such that information on the expected architectures' accuracy can be learned by the generator. This enables a stronger coupling with the predictor's target for the generation process and higher query efficiency (see Subsection 8.4.4). In contrast, previous models such as Huang and Chu [HC21], Lukasik et al. [Luk+21], and Yan et al. [Yan+20] are not fully differentiable and do not allow such optimization. Our generative model is pretrained on the task of reconstructing neural architectures, where for each randomly drawn latent space sample, we evaluate the reconstruction loss to a randomly drawn architecture. This simple procedure is facilitated by the heavily constrained search spaces of neural architectures, making it easy for the model to learn to generate valid architectures without being supported by a discriminator model as in GANs [Goo+14]. An evaluation of the generation ability of our model and its implementation details are provided in Section F.4 of Appendix F.

Performance Predictor. This generative model is coupled with a simple surrogate model, a 4-layer MLP with ReLU non-linearities, for target predictions C. These targets can be validation or test accuracy of the generated graph, or the latency with respect to a certain hardware. For comparison, we also include a tree-based method, XGBoost (XGB) [CG16] as an alternative prediction model. XGB [CG16] is used as a surrogate model in NAS-Bench-301 [Zel+22] and shows high prediction abilities. The input to XGB is the vector representation of the architectures. Since this method is non-differentiable, we additionally include a gradient estimation for rank-based metrics [Rol+20]. This way, we are able to include gradient information to the generator. Yet, it is important to note that this approach is not fully differentiable. This comparison will allow us to measure the trade-off between using supposedly stronger predictors over the capability to allow for full end-to-end learning.

Training Objectives. The generative model p_{G} learns to reconstruct a randomly sampled architecture G from search space p_{D} given a randomly sampled latent vector $\mathbf{h} \sim \mathcal{N}(0, 1)$. The objective function for this generation process can be formulated as the sum of node-level loss $\mathcal{L}_{\mathrm{V}}$ and edge-level loss $\mathcal{L}_{\mathrm{E}}$:

$$\mathcal{L}_{\mathrm{G}}(\tilde{G}, G) = \mathcal{L}_{\mathrm{V}} + \mathcal{L}_{\mathrm{E}};\ \tilde{G} \sim p_{\mathrm{G}}(\mathbf{h});\ G \sim p_{\mathrm{D}}, \tag{8.1}$$

where $\mathcal{L}_{\mathrm{V}}$ is the cross entropy loss between the predicted and the ground truth nodes and $\mathcal{L}_{\mathrm{E}}$ is the binary cross entropy loss between the predicted and ground truth edges of the generated graph $\tilde{G}$. This training step is *completely unsupervised*. Figure 8.2 presents an overview of the training process. To include the training of the surrogate model, the objective function is reformulated to:

$$\mathcal{L}(\tilde{G}, G) = (1 - \alpha)\mathcal{L}_{\mathrm{G}}(\tilde{G}, G) + \alpha\mathcal{L}_{\mathrm{C}}(\tilde{G}, G), \tag{8.2}$$

where α is a hyperparameter trading off generator loss $\mathcal{L}_{\mathrm{G}}$ and prediction loss $\mathcal{L}_{\mathrm{C}}$ for prediction targets C of graph G. We use mean squared error as predictor loss. Furthermore, each loss is optimized using mini-batch gradient descent.

Generative Latent Space Optimization. To facilitate the generation process, we optimize the architecture representation space via weighted retraining [TDH20], resulting in a sample efficient search algorithm. The intuition of this approach is to place more probability mass on high-scoring latent points, (e.g. high performing or low latency architectures) and less mass on low-scoring points. Thus, this strategy is not discarding low-scoring architectures completely, which would be inadequate for proper learning. The generative model is trained on a data distribution that systematically increases the probability of high-scoring latent points. This can be done by assigning a weight ω_i to each data point $G_i \sim p_{\mathrm{D}}$, indicating its likelihood to occur during batch-wise training. In addition, the training objective is weighted via a weighted empirical mean $\sum_{G_i \sim p_{\mathrm{D}}} \omega_i \, \mathcal{L}$ for each data point. As for the weights itself, Tripp, Daxberger, and Hernández-Lobato [TDH20] proposed a rank-based weight function

$$\begin{aligned} w(G; p_{\mathrm{D}}, k) &\propto \frac{1}{kN + \mathrm{rank}_{f, p_{\mathrm{D}}}(G)}, \\ \mathrm{rank}_{f, p_{\mathrm{D}}}(x) &= |\{G_i : f(G_i) > f(G), G_i \sim p_{\mathrm{D}}\}|, \end{aligned} \tag{8.3}$$

where $f(\cdot)$ is the evaluation function of the architecture G_i; for NAS-Bench-101 [Yin+19b] and NAS-Bench-201 [DY20] it is the tabular benchmark entry, for NAS-Bench-301 [Zel+22] and NAS-Bench-NLP [Kly+22] it is the surrogate benchmark prediction. Similar to Tripp, Daxberger, and Hernández-Lobato [TDH20], we set $k = 10^{-3}$. The retraining procedure itself then consists of finetuning the pretrained generative model coupled with the surrogate model, where loss functions and data points are both weighted by $w(G; p_{\mathrm{D}}, k)$. We provide an intuition of our approach in Section F.6.

8.4 Experiments

We evaluate the proposed Architecture Generative Network (AG-Net) on the two commonly used tabular benchmarks NAS-Bench-101 [Yin+19b] and NAS-Bench-201 [DY20], the surrogate benchmarks NAS-Bench-301 [Zel+22] evaluated on the DARTS search space [LSY19], NAS-Bench-NLP [Kly+22], and the first hardware device-induced benchmark [Li+21]. Additionally, we perform experiments on the ImageNet [Den+09] classification task and show state-of-the-art performance on the DARTS search space. In our experiments on the Hardware-Aware Benchmark we consider the latency information for the NAS-Bench-201 search space. Details about each search space can be found in Appendix F in Section F.1. Additionally, we report implementation details and hyperparameters in Section F.3 and Section F.5 respectively.

8.4.1 Experiments on Tabular Benchmarks

NAS-Bench-101. For our experiments on NAS-Bench-101, we first pretrain our generator to generate valid graphs on the NAS-Bench-101 search space. This step does not require information about the performance of architectures and is therefore inexpensive. The pretrained generator is then used for all experiments on NAS-Bench-101. Our NAS algorithm is initialized by randomly sampling 16 architectures from the search space, which are then weighted by the weighting function $\mathcal{W} = w(G)_{G \sim p_{\mathrm{D}}}$. Then, latent space optimized architecture search is performed by iteratively retraining the generator coupled with the MLP surrogate model for 15 epochs and generating 100 architectures of which the top 16 (according to their accuracy prediction) are evaluated and added to the training data. This step is repeated until the desired number of

queries is reached. When generating architectures, we sample from a grid containing the 99%-quantiles from $\mathcal{N}(0, 1)$ uniformly distributed. This way, we sample more distributed latent variables for better latent space coverage. We compare our method to the VAE-based search method Arch2vec [Yan+20] and predictor-based model WeakNAS [Wu+21], as well as state-of-the-art methods such as NAO [Luo+18], random search [LT19], local search [WNS21b], Bayesian optimization [Sno+15], regularized evolution [Rea+19], and BANANAS [WNS21a]. Additionally, we compare the proposed AG-Net applying an XGB predictor instead of an MLP (see Section F.3). The results of this comparison are listed in Table 8.1. Here, we report the mean over 10 runs. Results including the standard deviation can be found in the Appendix. Note, we search for the architecture with the best validation accuracy and report the corresponding test accuracy. Furthermore, we plot the search progress in Figure 8.3. As we can see, our model AG-Net improves over all state-of-the-art methods, not only after querying 300 data points, reaching a top-1 test accuracy of 94.2%, but is also almost any time better during the search process.

A direct comparison to the recently proposed GANAS [Rez+21] on NAS-Bench-101 is difficult, since GANAS searches on NAS-Bench-101 until they find the best architecture in terms of validation accuracy, whereas we limit our search to a maximal amount of 192 queries and are able to find high-performing architectures already in this small query setting. The comparison of AG-Net to the generator paired with an XGB [CG16] predictor shows that our end-to-end learnable approach is favorable even over potentially stronger predictors.

NAS-Bench-201. This benchmark contains three different image classification tasks: CIFAR-10, CIFAR-100 [Kri09], and ImageNet16-120 [CLH17]. For the experiments on NAS-Bench-201[DY20] we retrain AG-Net in the weighted manner for 30 epochs. In this setting, we also compare AG-Net to two recent generative models [Rez+21; HC21]. SGNAS [HC21] trains a supernet by uniform sampling, following SETN [DY19a]. Additionally a Convolutional Neural Network (CNN)-based architecture generator is trained to search architectures on the supernet. When comparing with Yan et al. [Yan+20], we also adopt their evaluation scheme of adding only the best-performing architecture (top-1) to the training data instead of top-16 as in our other experiments.

We report the search results for different numbers of queries for the NAS-Bench-201 dataset in Table 8.2 and Table 8.3. In addition, we plot the search progress in terms of queries in Figure 8.4. Our method provides state-of-the-art results on all datasets for a varying number of queries. Most importantly, AG-Net shows strong performance in the few-query regime compared to Yan et al. [Yan+20] with the exception of CIFAR-100, proving its high query efficiency.

8.4.2 Experiments on Surrogate Benchmarks

We furthermore apply our search method on larger search spaces as DARTS [LSY19] and NAS-Bench-NLP [Kly+22] without ground truth evaluations for the whole search space, making use of surrogate benchmarks as NAS-Bench-301 [Zel+22], NAS-Bench-X11 [Yan+21] and NAS-Bench-Suite [Meh+22].

NAS-Bench-301. Here, we report experiments on the cell-based search space DARTS [LSY19] using the surrogate benchmark NAS-Bench-301 [Zel+22] for the CIFAR-10 [Kri09] image classification task. The exact search procedure using the cells individually is described in Subsection F.3.5. The results are described in Table 8.4 and visualized in Figure 8.5. Our method is comparable to other state-of-the-art methods in this search space.

Table 8.1: Results on NAS-Bench-101 for the search of the best architecture in terms of validation accuracy on CIFAR-10 (mean over 10 trials).

NAS Method	**Val. Acc** (%)	**Test Acc** (%)	**Queries**
Optimum*	95.06	94.32	-
Arch2vec + RL [Yan+20]	-	94.10	400
Arch2vec + BO [Yan+20]	-	94.05	400
NAO ‡[Luo+18]	94.66	93.49	192
BANANAS† [WNS21a]	94.73	94.09	192
Bayesian Optimization† [Sno+15]	94.57	93.96	192
Local Search† [WNS21b]	94.57	93.97	192
Random Search†[LT19]	94.31	93.61	192
Regularized Evolution†[Rea+19]	94.47	93.89	192
WeakNAS [Wu+21]	-	**94.18**	200
XGB (ours)	94.61	94.13	192
XGB + ranking (ours)	94.60	94.14	192
AG-Net (ours)	**94.90**	**94.18**	192

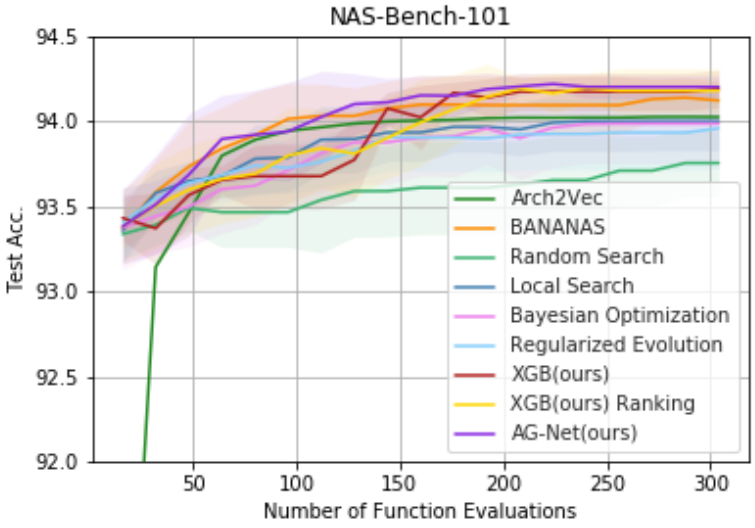


Figure 8.3: Architecture search evaluations on NAS-Bench-101.

Table 8.2: NAS on NAS-Bench-201. We report validation accuracies over the mean of 10 trials for searching the architecture with the highest validation accuracy on CIFAR-10 (CF-10), CIFAR-100 (CF-100), and ImageNet16-120 (IN16).

NAS Method	**CF-10** ↑	**CF-100** ↑	**IN16-120** ↑	**Queries**
Optimum*	91.61*	73.49*	46.77*	-
SGNAS [HC21]	90.18	70.28	44.65	Supernet
Arch2vec + BO [Yan+20]	91.41	**73.35**	46.34	100
AG-Net (ours)	**91.55**	73.20	46.31	96
AG-Net (ours, topk=1)	91.41	73.14	**46.42**	100
BANANAS† [WNS21a]	**91.56**	**73.49***	**46.65**	192
BO† [Sno+15]	91.54	73.26	46.43	192
RS † [LT19]	91.12	72.08	45.87	192
XGB (ours)	91.54	73.10	46.48	192
XGB + Ranking (ours)	91.48	73.20	46.40	192
AG-Net (ours)	**91.60**	**73.49***	46.64	192
GANAS [Rez+21]	-	-	-	444
AG-Net (ours)	**91.61***	**73.49***	**46.73**	400

Table 8.3: NAS on NAS-Bench-201. We report test accuracies over the mean of 10 trials for searching the architecture with the highest validation accuracy on CIFAR-10 (CF-10), CIFAR-100 (CF-100), and ImageNet16-120 (IN16).

NAS Method	**CF-10** ↑	**CF-100** ↑	**IN16-120** ↑	**Queries**
Optimum*	94.37*	73.51*	47.31*	-
SGNAS [HC21]	93.53	70.31	44.98	Supernet
Arch2vec + BO [Yan+20]	94.18	**73.37**	46.27	100
AG-Net (ours)	**94.24**	73.12	46.20	96
AG-Net (ours, topk=1)	94.16	73.15	**46.43**	100
BANANAS† [WNS21a]	94.30	73.50	**46.51**	192
BO† [Sno+15]	94.22	73.22	46.40	192
RS † [LT19]	93.89	72.07	45.98	192
XGB (ours)	94.34	72.93	46.08	192
XGB + Ranking (ours)	94.25	73.24	46.16	192
AG-Net (ours)	**94.37***	**73.51***	46.43	192
GANAS [Rez+21]	94.34	73.28	**46.80**	444
AG-Net (ours)	**94.37***	**73.51***	46.42	400

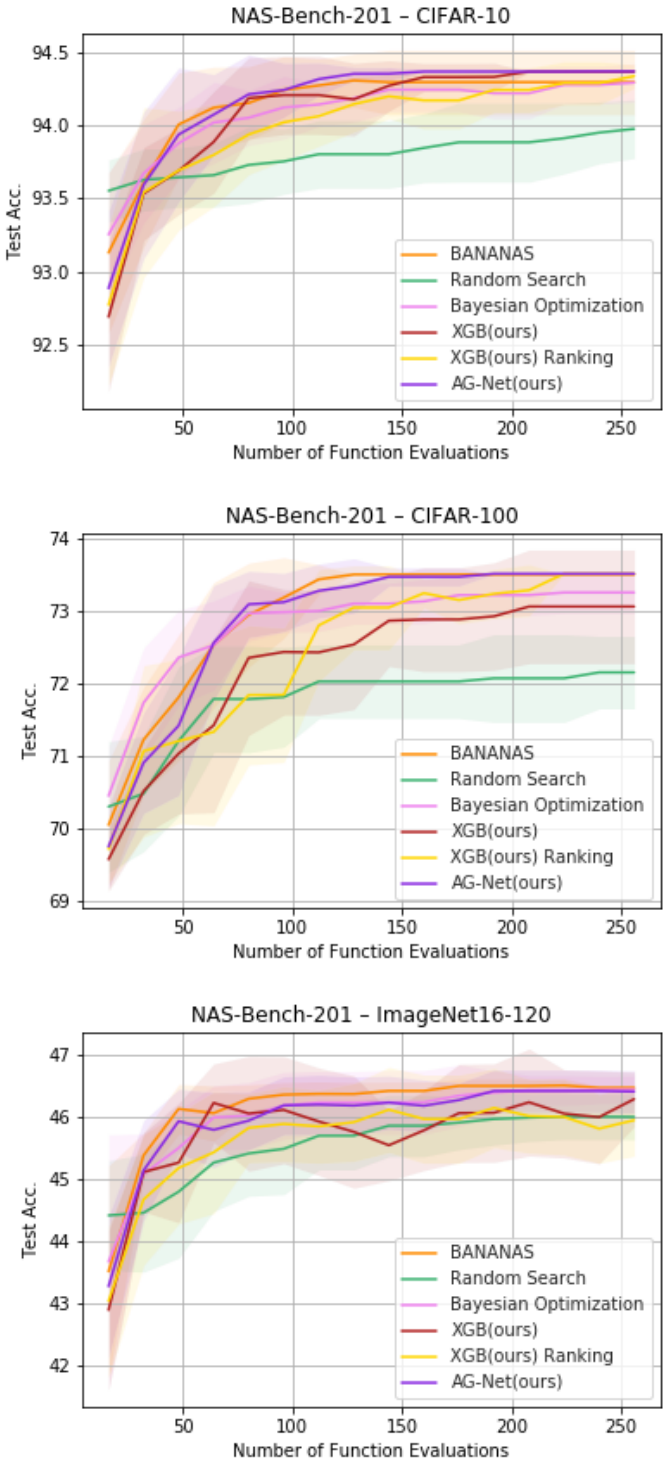


Figure 8.4: Architecture search evaluations on NAS-Bench-201.

Table 8.4: Results on NAS-Bench-301 (mean validation accuracy over 50 trials).

NAS Method	Val. Acc (%)	Queries
BANANAS† [WNS21a]	94.77	192
Bayesian Optimization† [Sno+15]	94.71	192
Local Search† [WNS21b]	**95.02**	192
Random Search† [LT19]	94.31	192
Regularized Evolution† [Rea+19]	94.75	192
XGB (ours)	94.79	192
XGB + Ranking (ours)	94.76	192
AG-Net (ours)	94.79	192

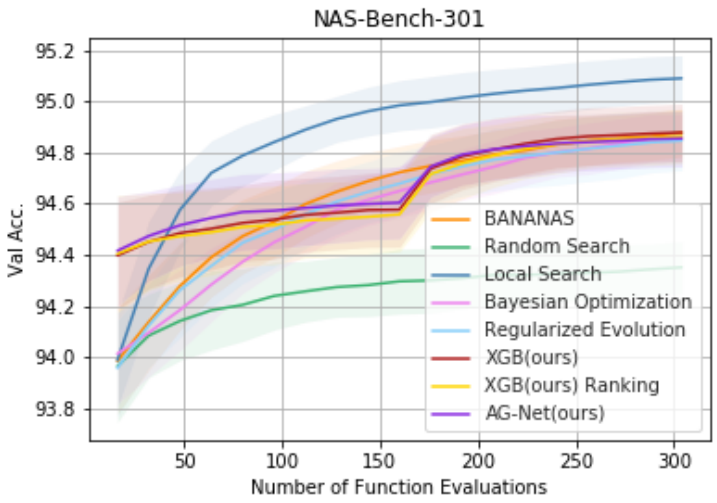


Figure 8.5: Architecture search evaluations on NAS-Bench-301.

NAS-Bench-NLP. Next, we evaluate AG-Net on NAS-Bench-NLP [Kly+22] for the language modeling task on Penn TreeBank [Mik+10]. We retrain AG-Net coupled with the surrogate model for 30 epochs to predict the validation perplexity. Note, since the search space considered in NAS-Bench-NLP is too large for a full tabular benchmark evaluation, we make use of the surrogate benchmark NAS-Bench-X11 [Yan+21] and NAS-Bench-Suite [Meh+22] instead of tabular entries. For fair comparison we compare our methods to the same state-of-the-art methods as in previous experiments. The results are reported in Table 8.5 and visualized in Figure 8.6. Our AG-Net improves over all state-of-the-art methods by a substantial margin and using XGB as a predictor even improves the search further.

ImageNet Experiments. The previous experiment on NAS-Bench-301 [Zel+22] shows the ability of our generator to generate valid architectures and to perform well in the DARTS [LSY19] search space. This allows for searching a well-performing architecture on ImageNet [Den+09]. Yet, evaluating up to 300 different found architectures on ImageNet is extremely expensive. Our first approach is to retrain the best found architectures on the CIFAR-10 [Kri09] image classification task from the previous experiment on NAS-Bench-301 (AG-Net and the XGB adaptions) on ImageNet [Den+09]. Our second approach is based on a training-free neural architecture search approach. The recently proposed TE-NAS [CGW21] provides a training-free neural architecture search approach, by ranking architectures via analysing the Neural Tangent Kernel (NTK) and the Number of Linear Regions (NLR) of each architecture. These two measurements are training-free and do not need any labels. The intuition between those two measurements is their implication towards trainability and expressivity of a neural architecture and also their correlation with the neural architecture's accuracy; NTK is negatively correlated and NLR positively correlated with the architecture's test accuracy. We adapt this idea for our search on ImageNet and search architectures in terms of their NTK value and their number of linear regions instead of their validation accuracy. We describe the detailed search process in Subsection F.3.5.

Table 8.6 shows the results. Note that our latter described search method on ImageNet is training-free (as TE-NAS [CGW21]) and the amount of queries displays the amount of data we evaluated for the zero cost measurements. Other query information include the amount of (partly) trained architectures. Furthermore, the displayed differentiable methods are based on training supernets which can lead to expensive training times. The best found architectures on NAS-Bench-301 [Zel+22] (CIFAR-10) result in comparable error rates on ImageNet to former approaches. As a result, our search method approach is highly efficient and outperforms previous methods in terms of needed GPU days. The result in terms of top-1 and top-5 error rates are even improving over the one from previous approaches when using the training-free approach.

8.4.3 Experiments on Hardware-Aware Benchmark

Next, we apply AG-Net to the Hardware-Aware NAS-Benchmark [Li+21]. We demonstrate in two settings that AG-Net can be used for multi-objective learning. The first setting (*Joint=1*) is formulated as constrained joint optimization:

$$\max_{G \sim p_{\mathrm{D}}} f(G) \wedge \min_{G \sim p_{\mathrm{D}},} g_h(G) \qquad \text{s.t. } g_h(G) \leq L, \exists\, h \in H, \tag{8.4}$$

where $f(\cdot)$ evaluates architecture G for accuracy and $g_h(\cdot)$ evaluates for latency given a hardware $h \in H$ and a user-defined latency constraint L. The second setting (*Joint=0*) is formulated as constraint objective:

$$\max_{G \sim p_{\mathrm{D}}} f(G) \qquad \text{s.t. } g_h(G) \leq L, \exists\, h \in H, \tag{8.5}$$

Table 8.5: Results on NAS-Bench-NLP (mean validation perplexity over 100 trials).

NAS Method	Val. Perplexity (%)	Queries
BANANAS† [WNS21a]	95.68	304
Bayesian Optimization† [Sno+15]	–	–
Local Search† [WNS21b]	95.69	304
Random Search† [LT19]	95.64	304
Regularized Evolution† [Rea+19]	95.66	304
XGB (ours)	**95.95**	304
XGB + Ranking (ours)	95.92	304
AG-Net (ours)	95.86	304

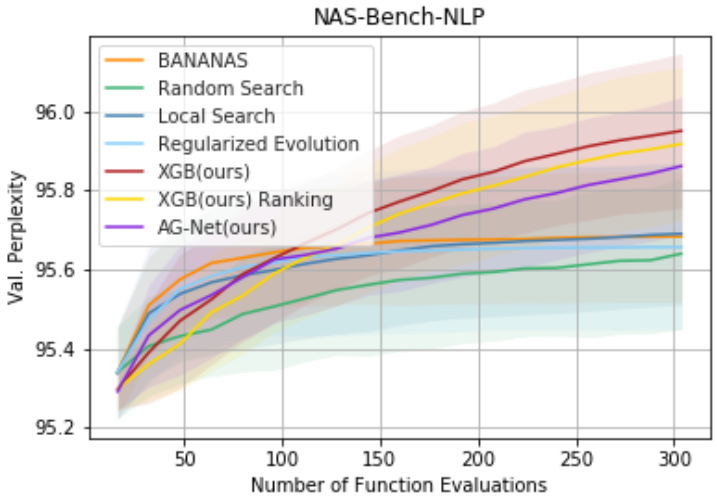


Figure 8.6: Architecture search evaluations on NAS-Bench-NLP.

where we drop the optimization on latency and only optimize accuracy with the latency constraint. The loss function to train our generator is updated from Equation 8.2 to:

$$\mathcal{L}(\tilde{G}, G) = (1-\alpha)\mathcal{L}_{\mathrm{G}}(\tilde{G}, G) + \alpha\big[\lambda\mathcal{L}_{\mathrm{C}_1}(\tilde{G}, G) + (1-\lambda)\mathcal{L}_{\mathrm{C}_2}(\tilde{G}, G)\big], \tag{8.6}$$

where α is a hyperparameter trading off generation and prediction loss, and λ is a hyperparameter trading off prediction targets C_1 (accuracy) and C_2 (latency).

To perform LSO in the joint objective setting from Equation 8.4, we rank the training data $\mathcal{D}$ for both accuracy and latency jointly by summing both individual rankings. To fulfill the optimization constraint, we further penalize the ranks via a multiplicative penalty if the latency does not fulfill the constraint. This overall ranking is then used for the weight calculation in Equation 8.3. The LSO for the constraint objective setting from Equation 8.5 only ranks architectures by accuracy and penalizes architectures with infeasible latency property. We choose random search as a baseline in this setting as it is generally regarded as a strong baseline in NAS [LT19]. Figure 8.7 (top) depicts searches with our model in both optimization settings on *Pixel 3* with different latency conditions. More results on different hardware and latency constraints are shown in Table 8.7. We observe that either optimization setting outperforms the random search baseline in almost all tasks. Additionally, our method is able to find the optimal architecture for a task regularly (in 15 out of 20 tasks), which random search was not able to provide. When considering mean accuracy and feasibility of the best architectures of all runs, we see that *Joint=1* is able to improve the ratio of feasible architectures found during the search substantially. This is to be expected given that the latent space is explicitly optimized for latency in this setting. Consequently, *Joint=1* is able to find better-performing architectures compared to *Joint=0* if the constraint restricts the space of feasible architectures strongly (see results on *Raspi 4*). The feasibility ratio of random search is an indicator on how restricted the space is. In most cases, the latency penalization seems to be sufficient to find enough well-performing and feasible architectures, as can be seen by the feasibility of *Joint=0* which improves compared to random search. We show the development of feasibility over time from Table 8.7 in Figure 8.7 (bottom).

8.4.4 Ablation Studies

In this section we analyse the impact of the LSO technique and the backpropagation ability to the search efficiency. Therefore, we compare our AG-Net with the latter named adaptions on the tabular benchmarks NAS-Bench-101 [Yin+19b] and NAS-Bench-201 [DY20]. The results of our ablation study are reported in Table 8.8. As we can see, the lack of weighted retraining decreases the performance of the search substantially. In addition, the results without backpropagation support that the coupling of the predictor's target and the generation process enables a more efficient architecture search over different search spaces. Thus, the combination of LSO and a fully differentiable approach improves the effectiveness of the search. We report additional ablation studies in Section F.2.

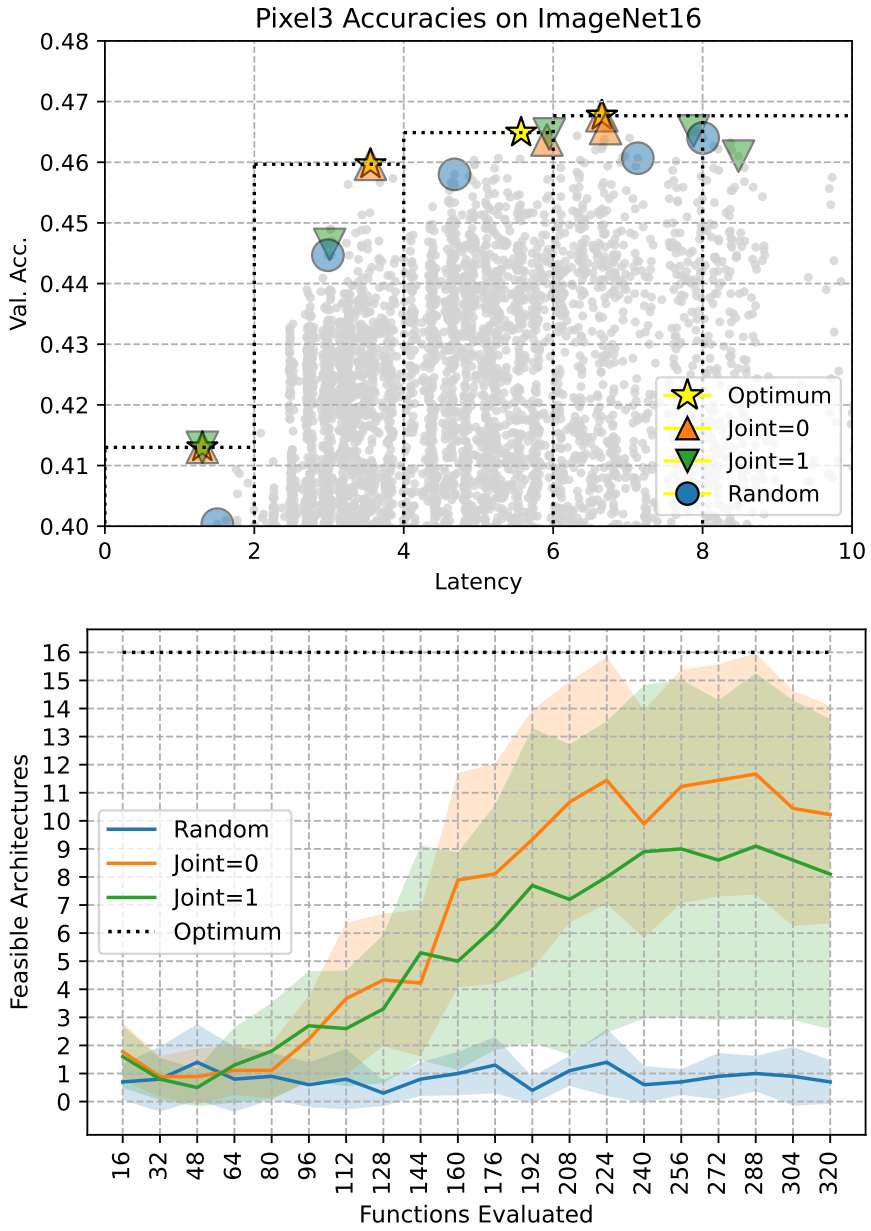


Figure 8.7: (**top**) Exemplary searches on HW-NAS-Bench for image classification on ImageNet16 with 192 queries on Pixel 3 and latency conditions $L \in \{2, 4, 6, 8, 10\}$ (y-axis zoomed for visibility). (**bottom**) Amount of architectures generated and selected in each search iteration (at most 16) that satisfy the latency constraint. In this example we searched on Edge GPU with $L = 2$.

Table 8.6: ImageNet **error** of neural architecture search on DARTS. We mark methods in the following way: $^{\text{CF}}$ for architecture search on CIFAR-10, $^{\text{IN}}$ for architecture search on ImageNet, and † for our methods.

NAS Method	**Top-1↓**	**Top-5↓**	**Queries**	**Search** GPU days
Mixed Methods				
NASNET-A $^{\text{CF}}$ [Zop+18]	26.0	8.4	20 000	2000
PNAS $^{\text{CF}}$ [Liu+18]	25.8	8.1	1160	225
NAO $^{\text{CF}}$ [Luo+18]	24.5	7.8	1000	200
Differentiable Methods				
DARTS $^{\text{CF}}$ [LSY19]	26.7	8.7	-	4.0
SNAS $^{\text{CF}}$ [Xie+19b]	27.3	9.2	-	1.5
PDARTS $^{\text{CF}}$ [Che+19]	**24.4**	**7.4**	-	0.3
PC-DARTS $^{\text{CF}}$ [Xu+20]	25.1	7.8	-	**0.1**
PC-DARTS $^{\text{IN}}$ [Xu+20]	**24.2**	**7.3**	-	3.8
Predictor-Based Methods				
WeakNAS $^{\text{IN}}$ [Wu+21]	**23.5**	**6.8**	800	2.5
XGB (NB-301) $^{\text{CF},\dagger}$	24.1	7.4	304	**0.02**
XGB + Ranking (NB-301) $^{\text{CF},\dagger}$	24.1	7.2	304	**0.02**
AG-Net (NB-301) $^{\text{CF},\dagger}$	24.3	7.3	304	0.21
Training-Free Methods				
TE-NAS $^{\text{CF}}$ [CGW21]	26.2	8.3	-	0.05
TE-NAS $^{\text{IN}}$ [CGW21]	24.5	7.5	-	0.17
AG-Net $^{\text{CF},\dagger}$	**23.5**	7.1	208	**0.02**
AG-Net $^{\text{IN},\dagger}$	**23.5**	6.9	208	**0.09**

Table 8.7: Results for searches with at most 200 queries on HW-NAS-Bench [Li+21] with varying devices and latency (Lat.) constraints in two multi-objective settings: *Joint=0* optimizes accuracy under latency constraint, while *Joint=1* optimizes for accuracy and latency jointly. We report the best found architecture out of 10 runs with their corresponding latency, as well as the mean of these runs. Feasibility (Feas.) is the proportion of evaluated architectures that satisfy the latency constraint. The optimal architecture (*) is the architecture with the highest accuracy satisfying the latency constraint.

Settings		**Best out of 10 runs**						**Mean**							
Constraint		**Joint=0**		**Joint=1**		**Random**		**Joint=0**		**Joint=1**		**Random**		**Optimum***	
Device	Lat.↓	Acc.↑	Lat.↓	Acc.↑	Lat.↓	Acc.↑	Lat.↓	Acc.↑	Feas.↑	Acc.↑	Feas.↑	Acc.↑	Feas.↑	Acc.↑	Lat.↓
Edge GPU	2	**0.406***	1.90	**0.406***	1.90	0.397	1.78	**0.397**	0.29	*0.391*	0.31	0.372	0.05	0.406	1.90
Edge GPU	4	**0.448***	3.49	**0.448***	3.49	0.437	3.35	*0.428*	0.29	**0.433**	0.43	0.417	0.22	0.448	3.49
Edge GPU	6	*0.458*	5.29	**0.464***	5.96	*0.458*	5.29	**0.453**	0.64	*0.450*	0.79	0.449	0.72	0.464	5.96
Edge GPU	8	*0.465*	6.81	**0.468***	6.81	0.464	7.44	**0.463**	0.98	*0.462*	0.99	0.457	1.00	0.468	6.81
Raspi 4	2	**0.355***	1.58	**0.355***	1.58	0.348	1.60	*0.346*	0.28	**0.347**	0.30	0.339	0.08	0.355	1.58
Raspi 4	4	*0.431*	3.83	**0.436***	3.79	0.427	3.85	*0.420*	0.47	**0.428**	0.50	0.419	0.37	0.436	3.79
Raspi 4	6	*0.449*	5.95	**0.452***	5.29	0.445	5.95	*0.440*	0.56	**0.441**	0.57	0.432	0.55	0.452	5.29
Raspi 4	8	*0.456*	6.33	0.455	7.96	**0.457**	7.97	**0.451**	0.69	*0.449*	0.79	0.447	0.76	0.465	7.43
Raspi 4	10	**0.466**	8.66	*0.465*	8.62	0.464	8.72	**0.464**	0.77	*0.454*	0.94	*0.454*	0.90	0.468	8.83
Raspi 4	12	**0.468***	8.83	0.463	9.05	*0.464*	8.72	**0.465**	0.91	*0.457*	0.98	0.456	0.96	0.468	8.83
Edge TPU	1	**0.468***	0.96	*0.466*	0.97	0.464	1.00	**0.464**	0.74	*0.457*	0.82	0.454	0.79	0.468	0.96
Pixel 3	2	**0.413***	1.30	**0.413***	1.30	0.400	1.50	**0.409**	0.48	*0.405*	0.59	0.388	0.30	0.413	1.30
Pixel 3	4	**0.460***	3.55	0.446	3.01	*0.447*	3.23	**0.453**	0.69	*0.441*	0.77	0.438	0.64	0.460	3.55
Pixel 3	6	*0.464*	5.92	**0.465***	5.95	0.458	4.68	**0.457**	0.77	*0.452*	0.94	0.451	0.88	0.465	5.57
Pixel 3	8	**0.468***	6.65	*0.465*	7.88	0.461	7.13	**0.464**	0.87	*0.457*	0.99	0.454	0.97	0.468	6.65
Pixel 3	10	**0.466**	6.70	0.461	8.48	*0.464*	8.01	**0.464**	0.96	0.455	1.00	*0.456*	0.99	0.468	6.65
Eyeriss	1	**0.452***	0.98	*0.449*	0.98	0.447	0.98	**0.445**	0.49	*0.436*	0.53	0.433	0.23	0.452	0.98
Eyeriss	2	**0.465**	1.65	**0.465**	1.65	0.464	1.65	**0.463**	0.87	*0.457*	0.99	*0.457*	0.95	0.468	1.65
FPGA	1	**0.440**	1.00	**0.440**	0.97	0.438	0.97	**0.433**	0.65	**0.433**	0.80	0.429	0.58	0.444	1.00
FPGA	2	**0.465***	1.60	0.460	1.60	*0.463*	1.97	**0.462**	0.82	0.451	0.99	*0.453*	0.97	0.465	1.60

Table 8.8: Ablation study results. Search with AG-Net on NAS-Bench-101 (101) and NAS-Bench-201 (201). We ablate the influence of backpropagation (BP) and LSO, and report the mean over 10 trials with a maximal query amount of 192.

NB	Dataset	Metric	Optimum*	AG-Net	w/o LSO	w/o BP
101	CIFAR-10	Val.	95.06	**94.90**	94.38	94.71
	CIFAR-10	Test	94.32	**94.18**	93.78	94.12
201	CIFAR-10	Val.	91.61	**91.60**	91.15	**91.60**
	CIFAR-10	Test	94.37	**94.37***	93.84	94.30
	CIFAR-100	Val.	73.49	**73.49***	71.72	73.38
	CIFAR-100	Test	73.51	**73.51***	71.83	73.22
	ImageNet16	Val.	46.77	**46.64**	45.33	46.62
	ImageNet16	Test	47.31	**46.43**	45.04	46.13

8.5 Conclusion and Outlook

Summary. We propose a simple architecture generative network (AG-Net), which allows us to directly generate architectures without any additional encoder or discriminator. AG-Net is fully differentiable, allowing to couple it with surrogate models for different target predictions. In contrast to former works, it enables to backpropagate the target information from the surrogate predictor into the generator. By iteratively optimizing the latent space of the generator, our model learns to focus on promising regions of the architecture space, so that it can generate high-scoring architectures directly in a query- and sample-efficient manner. Extensive experiments on common NAS benchmarks demonstrate that our model outperforms state-of-the-art methods at almost any time during architecture search and achieves state-of-the-art performance on ImageNet. It also allows for multi-objective optimization on the Hardware-Aware NAS-Benchmark.

Improving the Surrogate Model. To improve the query-efficiency of our method, we couple the generator with a surrogate model to predict promising candidate architectures to evaluate. In an ablation study in Table F.1, we demonstrate how essential the performance-predicting surrogate model for our search method is. Using a small MLP guiding the search we can improve over random search and even localized heuristics. At the same time, this means that our approach is sensitive to the effectiveness of the surrogate model, as the search outcome is dependent on its accuracy. Future research can improve this aspect of the search algorithm by improving the predictive ability of the surrogate model. For example, although XGBoost [CG16] is a strong predictor as shown by Zela et al. [Zel+22], we show in Table F.2 that our MLP-based approach outperforms it as surrogate model in our case. We contribute this to the fact that we cannot (or only approximately) backpropagate through this predictor, as XGBoost is not differentiable. However, Marton et al. [Mar+24] recently introduced a differentiable decision tree ensemble. Using this model in our search seems like a promising refinement for our method. Additionally, future research could investigate the effectiveness of ensembling multiple predictors. Taking confidence estimates into account could mitigate the risk of overly committing to certain regions in the latent space [LPB17]. One could even explore possible regularization methods when training the surrogate model, for example penalizing overconfident predictions.

Further Improving Query Efficency. In the current search approach, the generator is given random noise to produce architectures. These architectures are then assessed by the surrogate model, and the best candidates (according to the surrogate model) are selected for training and evaluation. Due to weighted retraining, the architecture latent space is biased towards architectures improving on the target function, which increases the likelihood to (randomly) sample well-performing architectures. Another possible future research direction could be to (also) optimize this sampling from the architecture search space. For example, in Chapter 7 we are able to select the globally optimal latent code by optimizing it via Mixed-Integer Optimization (MIO). Given the differentiable tree ensemble by Marton et al. [Mar+24], we could adapt a similar approach here. Another option would be gradient-based optimization, given any differentiable surrogate model is employed.

Search Space Dependency. Our model requires a generator that is pretrained to generate valid architectures for each search space individually. This limits the transferability of our approach. A natural improvement would be to incorporate more flexibility in terms of the generative modeling. Future research could investigate this aspect, for example by encoding operations in neural architectures as a dictionary of tokens and autoregressively sampling a computational graph like it is done in large language models [Vas+17; Bro+20].

Chapter 9
Data for Robust Neural Architecture Design

Contents of This Chapter

Chapter Topic. This chapter is based on Jung*, Lukasik*, and Keuper [JLK23]. It introduces a dataset designed to facilitate the research of robustness of neural architectures. This dataset is created by evaluating the whole search space of NAS-Bench-201 for several adversarial attacks and common corruptions. We discuss possible use cases of this dataset, among them is benchmarking of training-free robustness prediction metrics, but also neural architecture search. In particular, we use this dataset to apply the neural architecture search method proposed in Chapter 8 to find robust architectures optimized for multiple robustness measurements.

Chapter Outline. We introduce this chapter in Section 9.1. Then, in Section 9.2 we introduce evaluation metrics for robustness that we are interested in when creating the proposed dataset. We further elaborate on the dataset creation in Section 9.3, where we describe the evaluated architecture search space NAS-Bench-201 [DY20] in more detail and detail which robustness measurements we collect. In Section 9.4, we present interesting use cases for our dataset. In particular, we show the results of applying the search method proposed in Chapter 8 by optimizing for multiple robustness measurements. Lastly, we conclude this chapter in Section 9.5.

9.1 Introduction

ONE factor in the ever-improving performance of deep neural networks is based on innovations in architecture design. The starting point was the unprecedented result of AlexNet [KSH12] on the visual recognition challenge ImageNet [Den+09]. Since then, the goal is to find better performing models, surpassing human performance. However, human design of new better performing architectures requires a huge amount of trial-and-error and a good intuition, such that the automated search for new architectures (Neural Architecture Search (NAS)) receives rapid and growing interest [ZL17; Rea+17; Yin+19b; DY20]. The release of tabular benchmarks [Yin+19b; DY20] led to a research change; new NAS methods can be evaluated in a transparent and reproducible manner for better comparison.

The rapid growth in NAS research with the main focus on finding new architecture designs with ever-better performance is recently accompanied by the search for architectures that are robust against adversarial attacks and corruptions. This is important, since image classification networks can be easily fooled by small perturbations, which are invisible for humans. This leads to false predictions of the neural network with high confidence.

Robustness in NAS research combines the objective of high performing and robust architectures [DY19b; Dev+21; Don+25; HYX21; Mok+21]. However, there was no attempt so far to evaluate a full search space on robustness, but rather architectures in the wild. This paper is a first step towards closing this gap. We are the first to introduce a robustness dataset based on evaluating a *complete* NAS search space, such as to allow benchmarking neural architecture search approaches for the robustness of the found architectures. This will facilitate better streamlined research on neural architecture design choices and their robustness. We evaluate all 6466 unique pretrained architectures from the NAS-Bench-201 benchmark [DY20] on common adversarial attacks [GSS15; KGB17; CH20] and corruption types [HD19]. We thereby follow the argumentation in NAS research that employing one common training scheme for the entire search space will allow for comparability between architectures. Having the combination of pretrained models and the evaluation results in our dataset at hand, we further provide the evaluation of common training-free robustness measurements, such as the Frobenius norm of the Jacobian matrix [HRY19] and the largest eigenvalue of the Hessian matrix [Zha+20], on the full architecture search space and use these measurements as a method to find the supposedly most robust architecture. To show the application of our dataset in neural architecture search for robust models we perform several common NAS algorithms on the clean as well as on the robust accuracy of different image classification tasks. Additionally, we conduct an initial analysis of how architectural design choices affect robustness with the potential of doubling the robustness of networks with the same number of parameters. This is only possible since we evaluate the whole search space of NAS-Bench-201 [DY20], enabling us to investigate the effect of small architectural changes. To our knowledge we are the first paper to introduce a robustness dataset covering a full (widely used) search space allowing to track the outcome of fine-grained architectural changes. In summary we make the following contributions:

- We present the first robustness dataset evaluating a complete NAS search space.
- We present different use cases for this dataset; from training-free measurements for robustness to neural architecture search.
- Lastly, our dataset shows that a model's robustness against corruptions and adversarial attacks is highly sensitive towards the architectural design, and carefully crafting architectures can substantially improve their robustness.

9.2 Related Work

Common Corruptions. While neural architectures achieve results in image classification that supersede human performance [He+15], common corruptions such as Gaussian noise or blur can cause this performance to degrade substantially [DK17]. For this reason, Hendrycks and Dietterich [HD19] propose a benchmark that enables researchers to evaluate their network design on several common corruption types.

Adversarial Attacks. Szegedy et al. [Sze+14] showed that image classification networks can be fooled by crafting image perturbations, so called adversarial attacks, that maximize the networks' prediction towards a class different to the image label. Surprisingly, these perturbations can be small enough such that they are not visible to the human eye. One of the first adversarial attacks, called Fast Gradient Sign Method (FGSM) [GSS15], tries to flip the label of an image in a single perturbation step of limited size. This is achieved by maximizing the loss of the network and requires access to its gradients. Later gradient-based methods, like Projected Gradient Descent (PGD) [KGB17], iteratively perturb the image in multiple gradient steps. To evaluate robustness in a structured manner, Croce and Hein [CH20] propose an ensemble of different attacks, including an adaptive version of PGD called Adaptive Projected Gradient Descent (APGD) [CH20], and a blackbox attack called Square Attack [And+20] that has no access to network gradients. Croce et al. [Cro+21] conclude the next step in robustness research by providing an adversarial robustness benchmark, RobustBench, tracking state-of-the-art models in adversarial robustness.

Neural Architecture Search. NAS is an optimization problem with the objective to find an optimal combination of operations in a predefined, constrained *search space*. Early NAS approaches differ by their *search strategy* within the constraint search space. Common NAS strategies are evolutionary methods [Rea+17; Rea+19], Reinforcement Learning (RL) [ZL17; Li+18b], random search [BB12; LT19], local search [WNS21b], Bayesian Optimization (BO) [Kan+18; Ru+21; WNS21a]. Recently, several NAS approaches use generative models to search within a continuous latent space of architectures [Luk+21; Rez+21; LJK22]. To further improve the search strategy efficiency, the research focus shift from discrete optimization methods to faster differentiable search methods, using weight-sharing approaches [Pha+18; LSY19; Ben+18; CZH19; Xie+19b; Zel+20]. In order to compare NAS approaches properly, NAS benchmarks were introduced and opened the path for fast evaluations. The tabular benchmarks NAS-Bench-101 [Yin+19b] and NAS-Bench-201 [DY20] provide exhaustive evaluations of performances and metrics within their predefined search space on image classification tasks. TransNAS-Bench-101 [Dua+21] introduces a benchmark containing performance and metric information across different vision tasks. We will give a more detailed overview about the NAS-Bench-201 [DY20] benchmark in Subsection 9.3.1.

Robustness in NAS. With the increasing interest in NAS in general, the aspect of robustness of the optimized architectures has become more and more relevant. Devaguptapu et al. [Dev+21] provide a large-scale study that investigates how robust architectures, found by several NAS methods such as Liu, Simonyan, and Yang [LSY19], Cai, Zhu, and Han [CZH19], and Xu et al. [Xu+20], are against several adversarial attacks. They show that these architectures are vulnerable to various different adversarial attacks. Guo et al. [Guo+20] first search directly for a robust neural architecture using one-shot NAS and discover a family of robust architectures. Dong et al. [Don+25] constrain the architectures' parameters within a supernet to reduce the Lipschitz

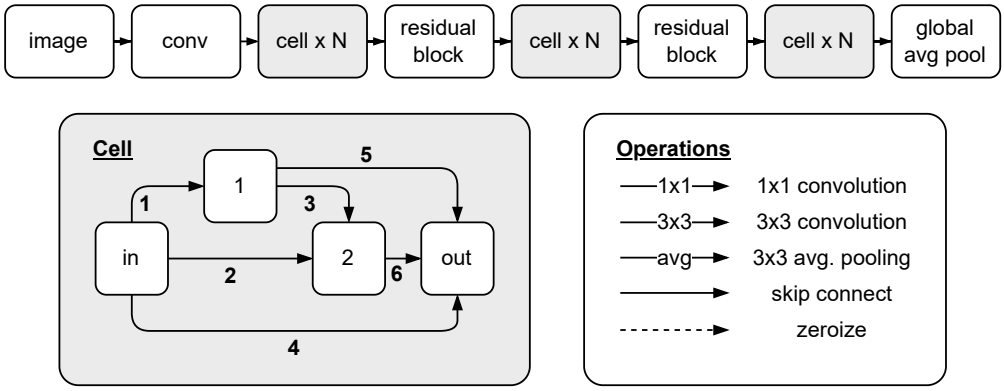


Figure 9.1: **(top)** Macro architecture. Gray highlighted cells differ between architectures, while the other components stay fixed. **(bottom)** Cell structure and the set of possible, predefined operations. (Figure adapted from [DY20])

constant and therefore increase the resulting networks' robustness. Few prior works such as Carlini et al. [Car+19], Xie et al. [Xie+19a], Pang et al. [Pan+21], and Xie et al. [Xie+21] propose more in-depth statistical analyses. In particular, Su et al. [Su+18] evaluate 18 ImageNet models with respect to their adversarial robustness. Ling et al. [Lin+19] and Dong et al. [Don+20] provide platforms to evaluate adversarial attacks. Tang et al. [Tan+21] provide a robustness investigation benchmark based on different architectures and training techniques on ImageNet. Recently a new line of differentiable robust NAS arose, namely including differentiable network measurements to the one-shot loss target to increase the robustness [HYX21; Mok+21]. Hosseini, Yang, and Xie [HYX21] define two differentiable metrics to measure the robustness of the architecture, certified lower bound and Jacobian norm bound, and searches for architectures by maximizing these metrics, respectively. Mok et al. [Mok+21] propose a search algorithm using the intrinsic robustness of a neural network being represented by the smoothness of the network's input loss landscape, i.e. the Hessian matrix.

9.3 Dataset Generation

9.3.1 Architectures in NAS-Bench-201

NAS-Bench-201 [DY20] is a cell-based architecture search space. Each cell has in total 4 nodes and 6 edges. The nodes in this search space correspond to the architecture's feature maps and the edges represent the architectures operation, which are chosen from the operation set $\mathcal{O} = \{ 1 \times 1 \text{ conv.}, 3 \times 3 \text{ conv.}, 3 \times 3 \text{ avg. pooling}, \text{skip}, \text{zero} \}$ (see Figure 9.1). This search space contains in total $5^6 = 15\,625$ architectures, from which only 6466 are unique, since the operations skip and zero can cause isomorphic cells (see Figure G.1, appendix), where the latter operation zero stands for dropping the edge. Each architecture is trained on three different image datasets for 200 epochs: CIFAR-10 [Kri09], CIFAR-100 [Kri09] and ImageNet16-120 [CLH17]. For our evaluations, we consider all unique architectures in the search space and test splits of the corresponding datasets. Hence, we evaluate $3 \cdot 6466 = 19\,398$ pretrained networks in total. Section G.1 in Appendix G describes technical details about the generation and structure of our dataset. In the following, we describe evaluations we collect.

9.3.2 Robustness to Adversarial Attacks

We start by collecting evaluations on different adversarial attacks, namely FGSM, PGD, APGD, and Square Attack. Following, we describe each attack and the collection of their results.

FGSM. Given function $\text{sign}(\cdot)$ returning the sign of its input, FGSM [GSS15] finds adversarial examples via

$$\tilde{\mathbf{x}} = \mathbf{x} + \epsilon \cdot \text{sign}\big(\Delta_{\mathbf{x}} \mathcal{L}(\theta, \mathbf{x}, y)\big), \tag{9.1}$$

where $\tilde{\mathbf{x}}$ is the adversarial example, $\mathbf{x}$ is the input image, y the corresponding label, ϵ the magnitude of the perturbation, and θ the network parameters. $\mathcal{L}(\theta, \mathbf{x}, y)$ is the loss function used to train the attacked network. In the case of architectures trained for NAS-Bench-201, this is Cross Entropy (CE). Since attacks via FGSM can be evaluated fairly efficient, we evaluate all architectures for $\epsilon \in E_{FGSM} = \{.1, .5, 1, 2, \ldots, 8, 255\}/255$, so for a total of $|E_{FGSM}| = 11$ times for each architecture. We use Foolbox [RBB17] to perform the attacks, and collect (a) accuracy, (b) average prediction confidences, as well as (c) confusion matrices for each combination of network and perturbation magnitude ϵ.

PGD. While FGSM perturbs the image in a single step of size ϵ, PGD [KGB17] iteratively perturbs the image via

$$\tilde{\mathbf{x}}_{n+1} = \text{clip}_{\epsilon,\mathbf{x}}\Big(\tilde{\mathbf{x}}_n - \alpha \cdot \text{sign}\big(\Delta_{\mathbf{x}} \mathcal{L}(\theta, \tilde{\mathbf{x}}_n, y)\big)\Big), \quad \tilde{\mathbf{x}}_0 = \mathbf{x}, \tag{9.2}$$

where $\tilde{\mathbf{x}}_n$ is the adversarial example at iteration n, α the step size in each iteration, and $\text{clip}_{\epsilon,\mathbf{x}}(\cdot)$ is a function clipping to range $[\mathbf{x} - \epsilon, \mathbf{x} + \epsilon]$. Due to its iterative nature, PGD is more efficient in finding adversarial examples, but requires more computation time. Therefore, we find it sufficient to evaluate PGD for $\epsilon \in E_{PGD} = \{.1, .5, 1, 2, 3, 4, 8\}/255$, so for a total of $|E_{PGD}| = 7$ times for each architecture. Same as for FGSM, we use Foolbox [RBB17] to perform the attacks using their L_∞ PGD implementation and keep the default settings, which are $\alpha = 0.01/0.3$ for 40 attack iterations. We collect (a) accuracy, (b) average prediction confidences, and (c) confusion matrices for each network and ϵ combination.

APGD. AutoAttack [CH20] offers an adaptive version of PGD that reduces its step size over time without the need for hyperparameters. We perform this attack using the L_∞ implementation provided by Croce and Hein [CH20] on CE and choose $E_{APGD} = E_{PGD}$. We kept the default number of attack iterations that is 100. We collect (a) accuracy, (b) average prediction confidences, and (c) confusion matrices for each network and ϵ combination.

Square Attack. In contrast to the before-mentioned attacks, Square Attack is a blackbox attack that has no access to the networks' gradients. It solves the following optimization problem using random search:

$$\min_{\tilde{\mathbf{x}}} \{f_{y,\theta}(\tilde{\mathbf{x}}) - \max_{c \neq y} f_{c,\theta}(\tilde{\mathbf{x}})\}, \text{ s.t. } \|\tilde{\mathbf{x}} - \mathbf{x}\|_p \leq \epsilon, \tag{9.3}$$

where $f_{c,\theta}(\cdot)$ are the network predictions for class c. We perform this attack using the L_∞ implementation provided by Croce and Hein [CH20] and choose $E_{Square} = E_{PGD}$. We kept the default number of search iterations at 5000. We collect (a) accuracy, (b) average prediction confidences, and (c) confusion matrices for each network and ϵ combination.

Summary. Figure 9.2 shows aggregated evaluation results on the before-mentioned attacks on CIFAR-10 w.r.t. accuracy. Growing gaps between mean and max accuracies indicate that the architecture has an impact on robust performances. Figure 9.3 depicts the correlation of ranking all architectures based on different attack scenarios. While there is larger correlation within the same adversarial attack and different values of ϵ, there seem to be architectural distinctions for susceptibility to different attacks. We depict these results for CIFAR-100 and ImageNet16-120 in Section G.4 in Appendix G.

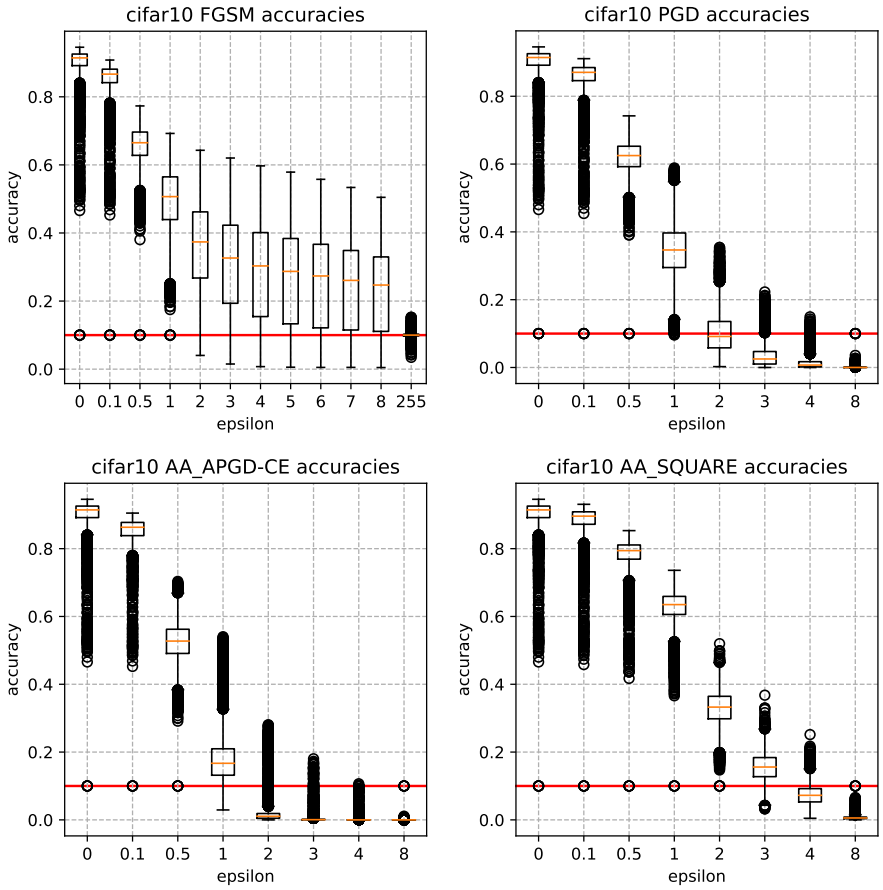


Figure 9.2: Accuracy boxplots over all 6466 unique architectures in NAS-Bench-201 for different adversarial attacks (FGSM [GSS15], PGD [KGB17], APGD [CH20], Square [And+20]) and perturbation magnitude values ϵ, evaluated on CIFAR-10. Red line corresponds to guessing. The large spread indicates towards architectural influence on robust performance.

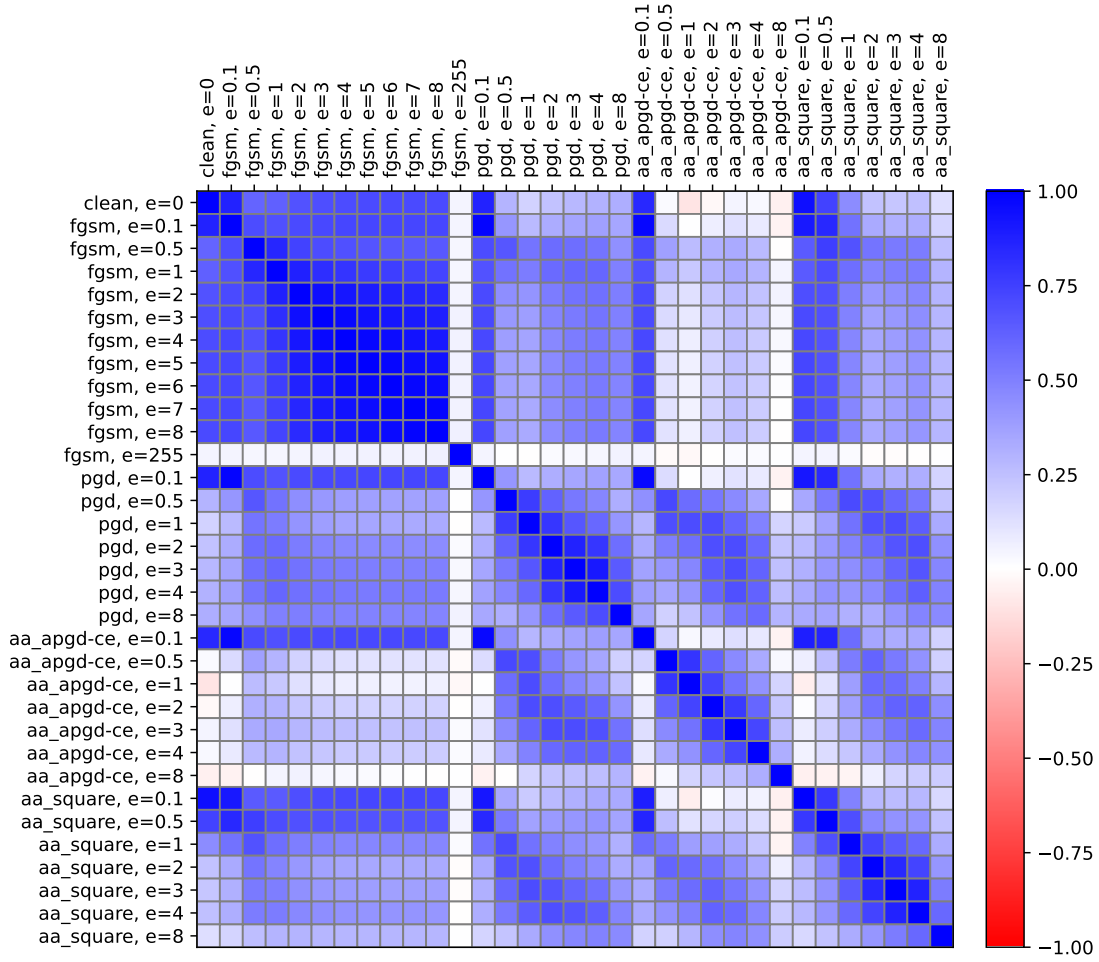


Figure 9.3: Kendall rank correlation coefficient between clean accuracies and robust accuracies on different attacks and magnitude values ϵ on CIFAR-10 for all unique architectures in NAS-Bench-201. There seem to be architectural distinctions for susceptibility to different attacks. We depict these correlations for CIFAR-100 and ImageNet16-120 in Section G.4 in Appendix G.

9.3.3 Robustness to Common Corruptions

To evaluate all unique NAS-Bench-201 [DY20] architectures on common corruptions, we evaluate them on the benchmark data provided by Hendrycks and Dietterich [HD19]. Two datasets are available: CIFAR10-C, which is a corrupted version of CIFAR-10 and CIFAR-100-C, which is a corrupted version of CIFAR-100. Both datasets are perturbed with a total of 15 corruptions at 5 severity levels (see Section G.3 in Appendix G for an example). The training procedure of NAS-Bench-201 only augments the training data with random flipping and random cropping. Hence, no influence should be expected of the training augmentation pipeline on the performance of the networks to those corruptions. We evaluate each of the $15 \cdot 5 = 75$ datasets individually for each network and collect (a) accuracy, (b) average prediction confidences, and (c) confusion matrices.

Summary. Figure 9.4 depicts mean accuracies for different corruptions at increasing severity levels. Similar to Figure 9.2, a growing gap between mean and max accuracies for most of the corruptions can be observed, which indicates towards architectural influences on robustness to common corruptions. Figure 9.5 depicts the ranking correlation for all architectures between clean and corrupted accuracies. Ranking architectures based on accuracy on different kinds of corruption is mostly uncorrelated. This indicates a high diversity of sensitivity to different kinds of corruption based on architectural design. We depict these results for CIFAR-100 in Section G.4 in Appendix G.

9.4 Dataset Use Cases

9.4.1 Training-Free Measurements for Robustness

Recently, a new research focus in differentiable NAS shifted towards finding not only high-scoring architectures but also adversarially robust architectures against several adversarial attacks [HYX21; Mok+21] using training characteristics of neural networks. On the one hand, Hosseini, Yang, and Xie [HYX21] uses Jacobian-based differentiable metrics to measure robustness. On the other hand, Mok et al. [Mok+21] improves the search for robust architectures by including the smoothness of the loss landscape of a neural network. In this section, we evaluate these training-free gradient-based measurements with our dataset.

Jacobian-Based Robustness Predictions. To improve the robustness of neural architectures, Hoffman, Roberts, and Yaida [HRY19] introduced an efficient Jacobian regularization method with the goal to minimize the network's output change in case of perturbed input data, by minimizing the Frobenius norm of the network's Jacobian matrix, $\mathcal{J}$. Let $f_\theta : \mathbb{R}^D \to \mathbb{R}^C$ be a neural network with weights denoted by θ and let $\mathbf{x} \in \mathbb{R}^D$ be the input data. Let $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$ be a perturbed input, with $\boldsymbol{\epsilon} \in \mathbb{R}^D$ being a perturbation vector. The output of the neural network shifts then to $f_{\theta,c}(\mathbf{x} + \boldsymbol{\epsilon}) - f_{\theta,c}(\mathbf{x})$. The input-output Jacobian matrix can be used as a measurement for the networks stability against input perturbations [HRY19]:

$$f_{\theta,c}(\mathbf{x} + \boldsymbol{\epsilon}) - f_{\theta,c}(\mathbf{x}) \approx \sum_{d=1}^{D} \boldsymbol{\epsilon}_d \cdot \frac{\partial f_{\theta,c}}{\partial \mathbf{x}_d}(\mathbf{x}) = \sum_{d=1}^{D} \mathcal{J}_{\theta,c;d}(\mathbf{x}) \cdot \boldsymbol{\epsilon}_d, \tag{9.4}$$

according to Taylor-expansion. From Equation 9.4, we can directly see that the larger the Jacobian components, the larger is the output change and thus the more unstable is the neural network against perturbed input data. In order to increase the stability of the network, Hoffman, Roberts,

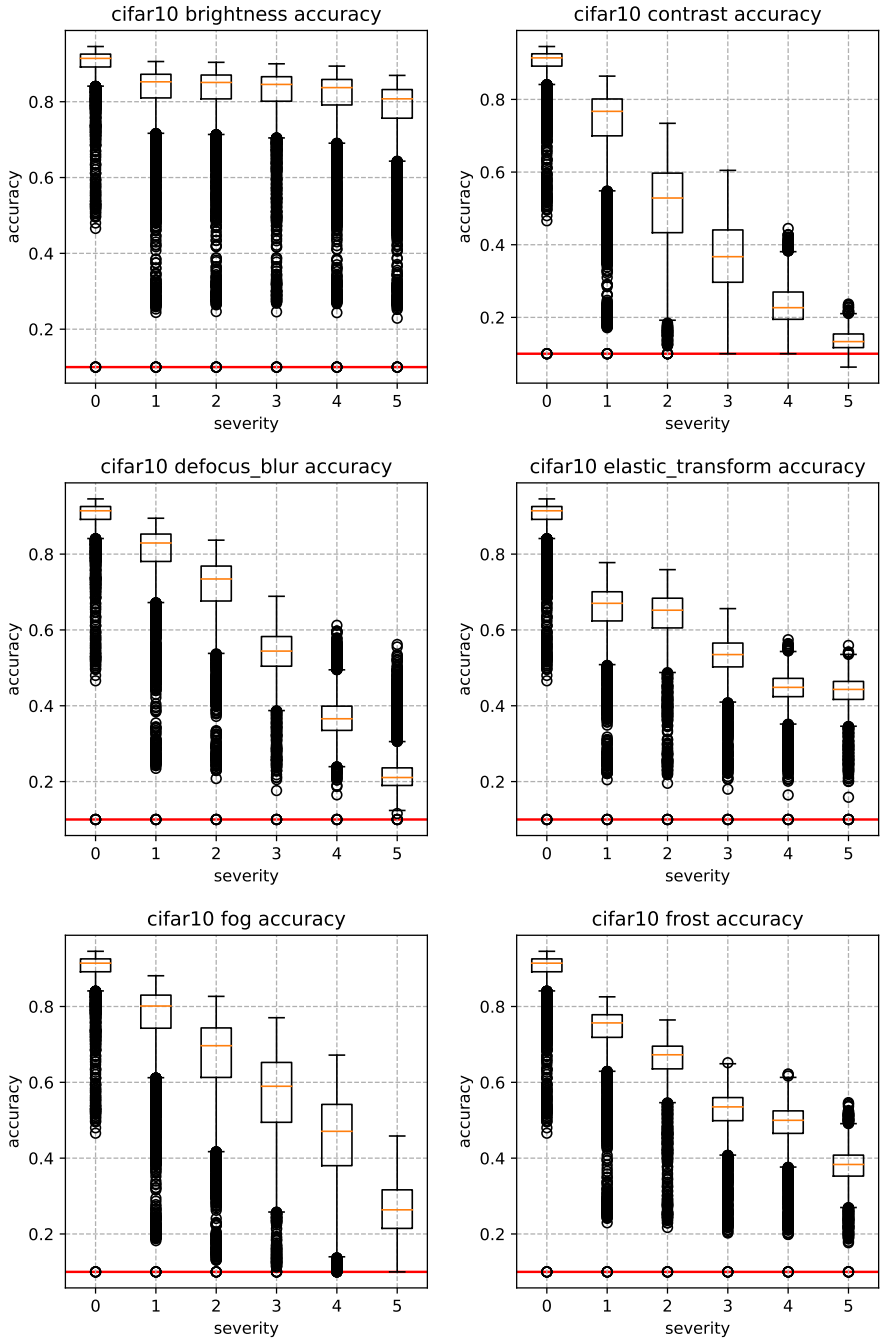


Figure 9.4: Accuracy boxplots over all unique architectures in NAS-Bench-201 for different corruption types at different severity levels, evaluated on CIFAR-10-C. Red line corresponds to guessing. All corruptions can be found in Figure G.14. The large spread indicates towards architectural influence on robust performance.

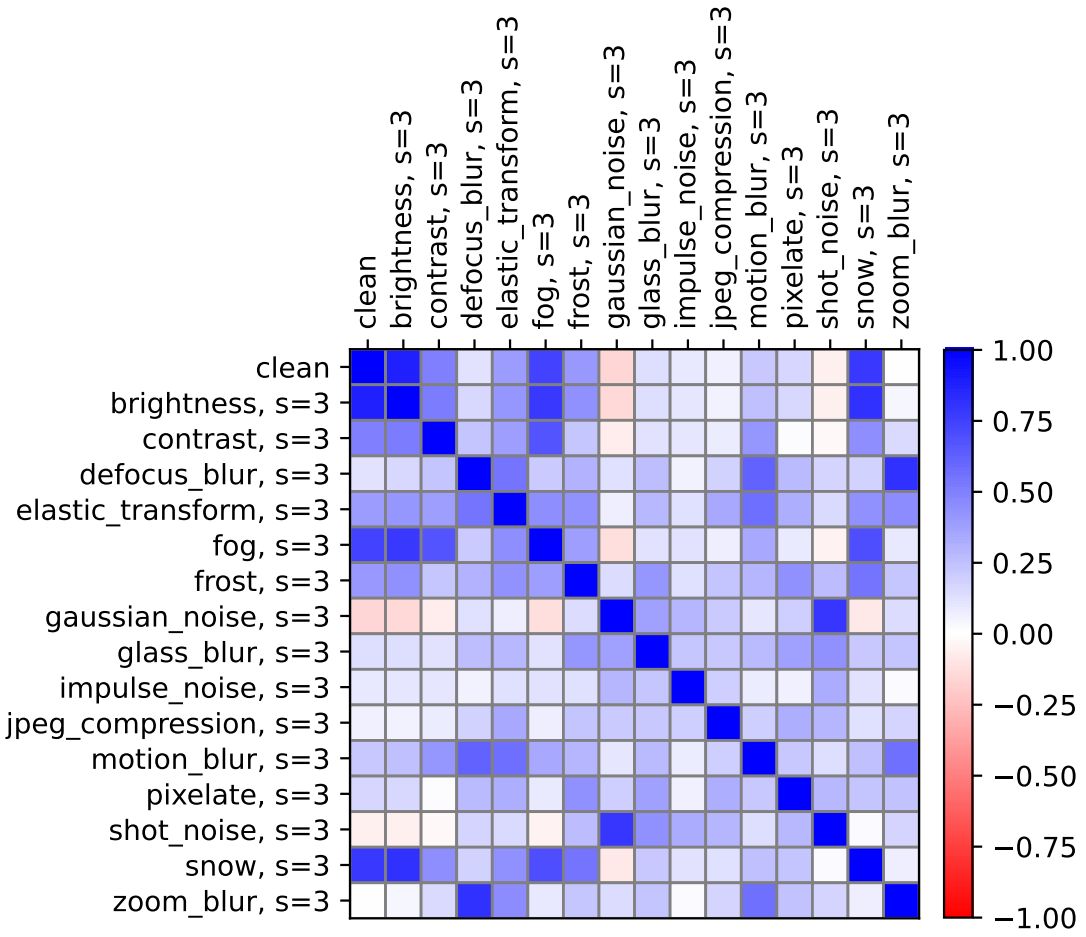


Figure 9.5: Kendall rank correlation coefficient between clean accuracies and accuracies on different corruptions at severity level 3 on CIFAR-10-C for all unique architectures in NAS-Bench-201. The mostly uncorrelated ranking indicates towards high diversity of sensitivity to different kinds of corruption based on architectural design.

and Yaida [HRY19] proposes to decrease the Jacobian components by minimizing the square of the Frobenius norm of the Jacobian. Following Hosseini, Yang, and Xie [HYX21], we use the efficient algorithm presented in Hoffman, Roberts, and Yaida [HRY19] to compute the Frobenius norm based on random projection for each neural network in the NAS-Bench-201 [DY20] benchmark.

Jacobian Benchmarking Results. The smaller the Frobenius norm of the Jacobian of a network, the more robust the network is supposed to be. Our dataset allows for a direct evaluation of this statement on all 6466 unique architectures. We use 10 mini-batches of size 256 of the training as well as test dataset for both randomly initialized and pretrained networks and compute the mean Frobenius norm. The results in terms of ranking correlation to adversarial robustness is shown in Figure 9.6 (top), and in terms of ranking correlation to robustness towards common corruptions in Figure 9.6 (bottom). We can observe that the Jacobian-based measurement correlates well with rankings after attacks by FGSM and smaller ϵ values for other attacks. However, this is not true anymore when ϵ increases, especially in the case of APGD.

Hessian-Based Robustness Prediction. Zhao et al. [Zha+20] investigate the loss landscape of a regular neural network and robust neural network against adversarial attacks. Let $\mathcal{L}(f_\theta(\mathbf{x}))$ denote the standard classification loss of a neural network f_θ for clean input data $\mathbf{x} \in \mathbb{R}^D$ and $\mathcal{L}(f_\theta(\mathbf{x}+\boldsymbol{\epsilon}))$ be the adversarial loss with perturbed input data $\mathbf{x}+\boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \in \mathbb{R}^D$. Zhao et al. [Zha+20] provide theoretical justification that the latter adversarial loss is highly correlated with the largest eigenvalue of the input Hessian matrix $\mathbf{H}(\mathbf{x})$ of the clean input data $\mathbf{x}$, denoted by $\lambda_{\max}$. Therefore, the eigenspectrum of the Hessian matrix of the regular network can be used for quantifying the robustness: large Hessian spectrum implies a sharp minimum resulting in a more vulnerable neural network against adversarial attacks. Whereas in the case of a neural network with small Hessian spectrum, implying a flat minimum, more perturbation on the input is needed to leave the minimum. We make use of Chatzimichailidis et al. [Cha+19] to compute the largest eigenvalue $\lambda_{\max}$ for each neural network in the NAS-Bench-201 [DY20] benchmark.

Hessian Benchmarking Results. For this measurement, we calculate the largest eigenvalues of all unique architectures using the Hessian approximation in Chatzimichailidis et al. [Cha+19]. We use 10 mini-batches of size 256 of the training as well as test dataset for both randomly initialized and pretrained networks and compute the mean largest eigenvalue. These results are also shown in Figure 9.6. We can observe that the Hessian-based measurement behaves similarly to the Jacobian-based measurement.

9.4.2 NAS on Robustness

In this section, we perform different NAS algorithms, among them the method introduced in Chapter 8, on the clean accuracy and robust accuracies for different adversarial attacks (with L_∞ perturbation maximum $\epsilon = 1/255$) in the NAS-Bench-201 [DY20] search space for CIFAR-10. We evaluate the best found architectures on provided clean and robust accuracies, as well as common corruptions. Searches are performed with random search [LT19], regularized evolution [Rea+19], local search [WNS21b], and our generative search method (see Chapter 8) with a maximal query amount of 300. Performing search with AG-Net allows us to optimize for multiple objectives at the same time. Here, we evaluate settings where we optimize for clean accuracy combined with each of the adversarial attacks. Additionally, we evaluate one setting where we optimize for all target accuracies. The results are shown in Table 9.1.

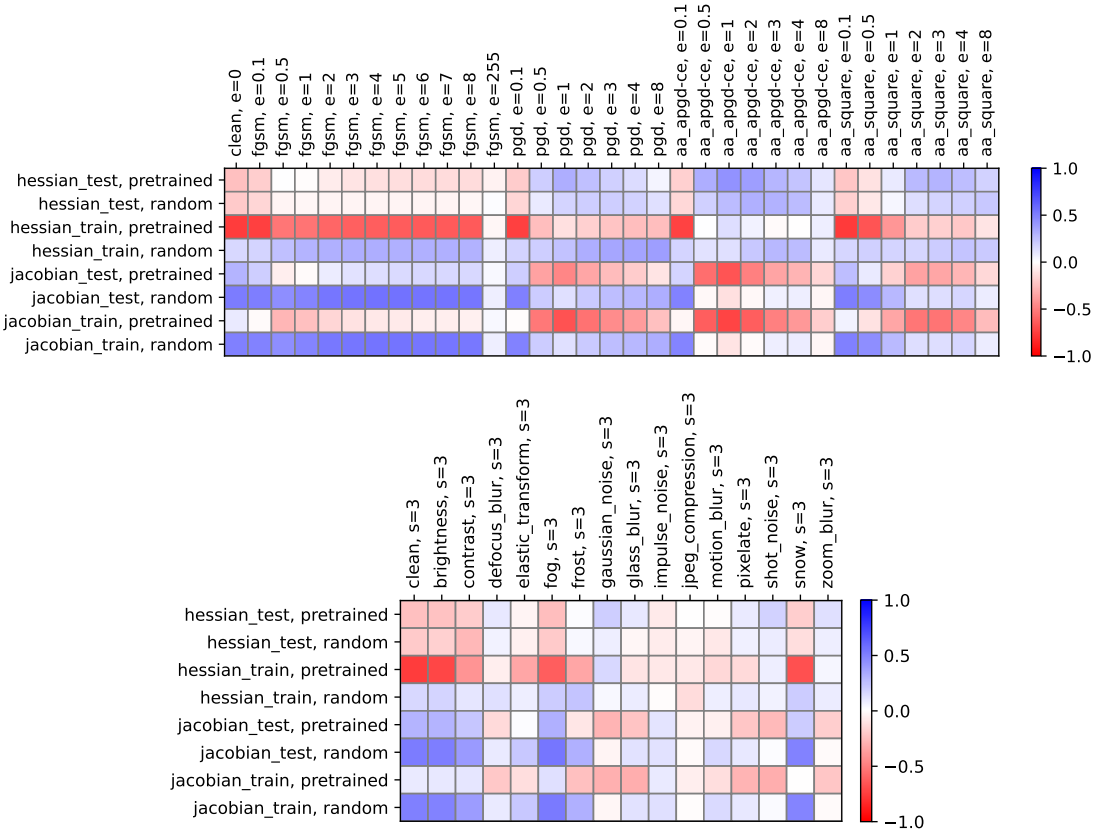


Figure 9.6: Kendall rank correlation coefficient between Jacobian- and Hessian-based robustness measurements computed on all unique NAS-Bench-201 architectures to corresponding rankings given by **(top)** different adversarial attacks and **(bottom)** different common corruptions. Measurements and accuracies are computed on CIFAR-10 / CIFAR-10-C. Measurements are computed on randomly initialized and pretrained networks contained in NAS-Bench-201. Jacobian-based and Hessian-based measurements correlate well for smaller ϵ values, but not for larger ϵ values.

Single-Objective Setting. Comparing different optimization targets in the single-objective setting, we can see that clean accuracy decreases as soon as we optimize for adversarial robustness. This is unsurprising at first, given that Tsipras et al. [Tsi+19] showed there is an inherent trade-off between performance on clean images and adversarial robustness of a model. We also see that this result reflects the correlation between clean and robust accuracies shown in Figure 9.3. However, it is interesting to see that this trade-off not only exists when adversarially training networks, but also when modifying their architecture while solely training on unperturbed data, as shown here. Furthermore, it is interesting that methods performing localized changes (evolutionary search, local search) can (mostly) improve robustness over random search, hence are able to find paths that modify networks in ways improving their adversarial robustness. This result indicates that tweaking the design of network architectures can improve adversarial robustness. Additionally, we can see that our generative search method performs best in all single-objective settings, hinting towards the efficiency that latent space optimization combined with predictor guidance provides. We assume that our network learns faster which design choices are important to improve the optimization target, as it is not restricted to produce architectures that are one edit distance away (changing only one operation, like it is the case for evolutionary search and local search).

Localized Changes on PGD. One exceptional result is the performance of localized methods on PGD. Here we see that random search outperforms both, and outperforms local search substantially. We hypothesize that this is the result of noise in the dataset, as evaluations are only performed once and not as an aggregate over multiple training runs. Both methods seem to get stuck at a local optimum, which is too far away in edit distance from architectures that perform better on PGD. For local search this effect is more prone than for evolutionary search, as it is initialized with a smaller population. We tried increasing the initial population size of local search, but it was still not on par with random search. From the results on APGD we can see that optimizing on this target seems to provide a more direct path towards network designs performing well in both targets, PGD as well as APGD.

Multi-Objective Setting. In the multi-objective setting we can see that AG-Net tries to find a trade-off between both targets, generally improving in regards to clean accuracy when also optimizing an adversarial attack, as opposed to only optimizing for the corresponding attack. However, albeit to be expected, this comes at the cost of decreasing adversarial robustness towards this attack. This is, again, especially pronounced with PGD and APGD, where robust accuracies substantially decrease. Overall, it seems that our method puts more emphasis on clean accuracy compared to the adversarial optimization target. This might be caused by the predictors, for which we assume that clean accuracy is easier to learn (as shown by the variances in Figure 9.2). This, in turn, reflects the difficulties evolutionary search and local search face when optimizing PGD. Additionally, the fact that FGSM and Squares are more strongly correlated with clean accuracy than with PGD and APGD (see Figure 9.3) implies that performance decreases on these targets are less substantial. This is again reflected in the setting where we optimize all the discussed targets simultaneously. From these results, we conclude that our generative search method effectively enables multiobjective optimization. However, target values should be normalized and decorrelated (a direction we leave for future work).

Table 9.1: Neural architecture search optimizing for clean test accuracy and adversarial robustness under several attacks (with L_∞ maximum perturbation magnitude $\epsilon = 1/255$). We highlight optimization targets with a light gray background. The search is performed by the listed methods on CIFAR-10 in the NAS-Bench-201 search space (mean over 100 runs). Results are the mean accuracies of the best architectures found and the mean accuracy over all corruptions and severity levels for common corruptions (CIFAR-10-C).

	Method	Test Accuracy ($\epsilon = 1/255$) Clean	FGSM	PDG	APGD	Squares	Clean
		CIFAR-10					CF-10-C
	Optimum	94.68	69.24	58.85	54.02	73.61	58.55
Clean	Random Search [LT19]	94.08	62.73	39.28	17.32	67.89	55.34
	Regularized Evolution [Rea+19]	94.48	64.39	42.14	19.82	69.98	56.88
	Local Search [WNS21b]	94.53	64.61	**43.10**	**20.68**	**70.63**	**57.79**
	AG-Net (Chapter 8)	**94.55**	**64.84**	42.86	20.48	70.50	57.45
FGSM	Random Search [LT19]	93.55	66.53	45.77	21.02	68.25	55.00
	Regularized Evolution [Rea+19]	93.74	68.75	**48.18**	22.79	69.09	56.03
	Local Search [WNS21b]	93.54	68.52	47.72	**23.38**	69.51	56.75
	AG-Net (Chapter 8)	**94.07**	**69.18**	46.38	21.76	**70.25**	**57.80**
PGD	Random Search [LT19]	82.03	57.07	57.36	52.21	63.76	52.14
	Regularized Evolution [Rea+19]	83.06	57.81	57.07	50.76	64.34	52.97
	Local Search [WNS21b]	**86.57**	**60.79**	54.67	42.67	**66.66**	**54.12**
	AG-Net (Chapter 8)	82.72	57.94	**58.21**	**53.00**	64.52	53.05
APGD	Random Search [LT19]	81.80	57.15	57.49	52.50	63.49	52.01
	Regularized Evolution [Rea+19]	82.00	57.69	58.11	53.21	63.95	52.60
	Local Search [WNS21b]	**82.17**	**57.80**	**58.27**	53.32	**64.33**	**53.36**
	AG-Net (Chapter 8)	81.33	57.48	58.07	**53.59**	62.95	52.13
Square	Random Search [LT19]	91.58	59.30	46.33	29.50	70.50	52.76
	Regularized Evolution [Rea+19]	92.26	61.62	48.95	31.62	72.19	54.31
	Local Search [WNS21b]	92.28	61.62	49.12	**31.77**	72.74	53.90
	AG-Net (Chapter 8)	**92.95**	**62.88**	**49.39**	31.21	**72.83**	**54.99**
Joint	AG-Net (Chapter 8)	94.20	**68.19**	45.15	21.23	70.48	57.02
		93.99	66.25	47.75	24.06	69.86	55.45
		94.00	66.29	**47.92**	**24.29**	69.77	55.12
		94.57	65.26	43.88	21.27	**71.05**	**58.05**
		94.05	67.65	46.75	22.71	70.24	56.99

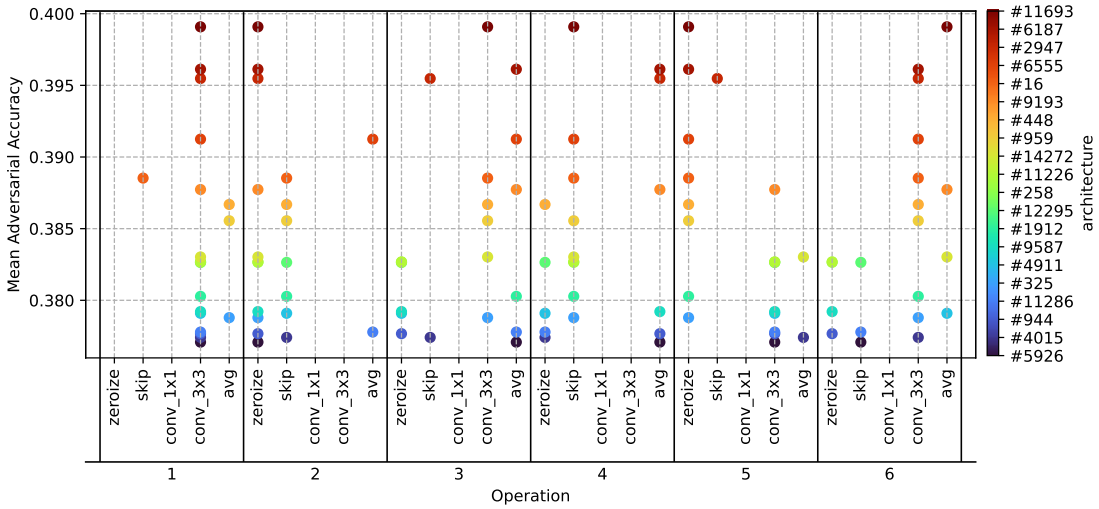


Figure 9.7: Top-20 architectures (out of 408) with exactly two 3×3 convolutions and no 1×1 convolutions according to mean adversarial accuracy on CIFAR-10. The operation number (1-6) corresponds to the edge in the cell, see Figure 9.1 for cell connectivity and operations. Stacking convolutions seems to be an important part of robust architectural design.

9.4.3 Effect of Architecture Design on Robustness

In Figure 9.7, we show the top-20 performing architectures (color-coded, one operation for each edge) with exactly two 3×3 convolutions and no 1×1 convolutions (hence, the same parameter count), according to the mean adversarial accuracy over all attacks as described in Subsection 9.3.2 on CIFAR-10. It is interesting to see that there are no convolutions on edges 2 and 4, and additionally no dropping (operation zeroize) or skipping (operation skip-connect) of edge 1. In the case of edge 4, it seems that a single convolutional layer connecting input and output of the cell increases sensitivity of the network. Hence, most of the top-20 robust architectures stack convolutions (via edge 1, followed by either edge 3 or 5), from which we hypothesize that stacking convolution operations can improve robustness (in contrast to combining parallel ones) when designing architectures. At the same time, skipping input to output via edge 4 seems not to affect robustness negatively, as long as the input feature map is combined with stacked convolutions. Further analyses can be found in Section G.5. We find that optimizing architecture design can have a substantial impact on the robustness of a network. In this setting, where networks have *the same parameter count*, we can see a large range of mean adversarial accuracies $[0.21, 0.4]$ showing the potential of doubling the robustness of a network by carefully crafting its topology. Important to note here is that this is a first observation, which can be made by using our provided dataset. This observation functions as a motivation for how this dataset can be used to analyze robustness in combination with architecture design.

9.5 Conclusion and Outlook

Summary. We introduce a dataset for neural architecture design and robustness to provide the research community with more resources for analyzing what constitutes robust networks.

We have evaluated *all* 6466 unique architectures from the commonly used NAS-Bench-201 benchmark against several adversarial attacks and image dataset corruptions. With this full evaluation at hand, we presented three use cases for this dataset: First, the correlation between the robustness of the architectures and two differentiable architecture measurements. We showed that these measurements are a good first approach for the architecture's robustness, but have to be taken with caution when the perturbation increases. Second, neural architecture search directly on the robust accuracies, which indeed finds more robust architectures for different adversarial attacks. And last, an initial analysis of architectural design, where we showed that it is possible to improve robustness of networks with the same number of parameters by carefully designing their topology.

Adversarially Trained Networks. Adversarial training is a standard method to harden neural networks against adversarial attacks [GSS15; Mad+18]. While it is interesting to investigate how neural architectures can be designed to naturally resist adversarial attacks, investigating the interplay between neural architecture design and adversarial training offers another direction for future research. Subsequent work evaluated this aspect by enriching our dataset with evaluations of architectures from NAS-Bench-201 [DY20] that were adversarially trained [Wu+24]. This opens up further possibilities for future work, investigating whether the findings of this chapter transfer when the same architectures are trained against adversarial attacks.

Further Distribution Shifts. While the evaluations on NAS-Bench-201 [DY20] that we provide are based on adversarial attacks [GSS15; KGB17; CH20] and common corruptions [HD19], there are further distribution shifts that would be interesting to investigate from the perspective of architecture design. For example, future research could collect and investigate robustness towards lense distortions [MBK23], or evaluate the shape vs. texture bias [Gei+19] of networks in NAS-Bench-201.

Class-Wise Fairness. Another interesting direction for future investigation is fairness. Specifically, the potential bias a classification network may learn by sacrificing performance on one class to improve performance on another. Blakeney et al. [Bla+22] introduce two metrics to measure fairness considerations. Extending the introduced dataset with fairness metrics could enable future research to investigate the influence of architecture design on fairness.

Evaluations on Multiple Seeds. When we collected all evaluations as described in this chapter, we relied on the pretrained checkpoints provided by NAS-Bench-201 [DY20]. While some architectures are trained on multiple different seeds, this is not true for all architectures. Consequently, we decided to only evaluate one of the seeds that was available for all architectures. At the same time, White, Nolen, and Savani [WNS21b] argue that training and evaluating a network only once during neural architecture search instead of multiple times introduces noise. They show that the search results of local search improve substantially when the performance of found architectures is averaged over multiple training runs. This makes intuitively sense given the randomness that is involved with initializing and training neural networks. Hence, there is a strong case for future work to retrain and evaluate the search space across multiple random seeds to reduce the impact of noise in the dataset.

Chapter 10
Conclusion

REPRESENTATION LEARNING has become a cornerstone of modern computer vision, enabling models to extract informative, task-relevant features directly from raw data. Yet, despite substantial advances, challenges remain in guiding these models to learn representations that are not only accurate but also generalizable, robust, and aligned with task-specific objectives [Gav+25; Man+25; Zhu+24]. This thesis was motivated by the realization that learning algorithms require carefully designed inductive biases to achieve such goals. The No Free Lunch (NFL) theorems [WM97] remind us that there is no universally optimal learning method. Instead, effective models must be steered using domain knowledge and assumptions about the structure of learned representations. Throughout this work, we explored how regularization strategies can serve as vehicles for encoding such assumptions.

In particular, this thesis explored encoding assumptions through three complementary regularization strategies: (Part I) manually designed penalty terms, (Part II) matching extracted features, and (Part III) augmenting and assigning importance to data. Each of these parts contributed novel methods that inject inductive biases into representation learning models. We summarize these contributions in Section 10.1, and then discuss key findings and open research questions with possible directions for further academic efforts in Section 10.2.

10.1 Thesis Summary

Part I: Penalty-Based Regularization. In the first part of the thesis, we derived penalty terms from the combinatorial Minimum Cost Multicut Problem (MP) [CR93; DL97] to regularize representation learning neural networks.

In Chapter 3, cycle consistency constraints from the MP were incorporated as penalty terms into training of a graph neural network-based solver for the same problem, encouraging it to produce feasible solutions. For this, we additionally proposed architectural modifications to common Message Passing Neural Network (MPNN) [Gil+17; KW17] to account for real-valued edge weights that are necessary to learn MP graph instances. Due to the lack of available large-scale training data for the MP, we introduced two synthetic datasets, IrisMP and RandomMP, that our models were trained on. Experiments on these datasets and real-world benchmarks showed that our model provides efficient and competitive solutions and scales better than highly optimized heuristics (i.e. Greedy Additive Edge Contraction (GAEC) [Keu+15]). Compared to heuristics, this representation learning-based method has the ability to provide gradients for downstream tasks. Applying the proposed penalty term steers the model towards more feasible solutions with the cost of finding less optimal ones (encoded by the edge classification loss on the ground truth optimal solutions during training). This demonstrated that we successfully encoded a preference for feasible solutions, introducing a steerable trade-off. Overall, this yielded edge representations that more reliably satisfied the discrete optimization constraints, and thereby improving the validity of solutions.

In Chapter 4, cycle consistency constraints were injected into the training loss of a convolutional network for edge detection. For this, we introduced an adaptive higher-order Conditional Random Field (CRF) and combined it with the Richer Convolutional Features (RCF) [Liu+19] model. The CRF energy function is defined by unary and pattern-based potentials that resemble cycle consistency constraints of the MP on the edge maps produced by the RCF model. Our experiments showed that combining the proposed CRF penalty with the RCF model results in sharper edge maps and closed contours. Due to its general design, the CRF can also be applied as a post-processing method for edge maps. We showed that this is effective in the case of electron microscopy data.

Together, Chapter 3 and Chapter 4 demonstrated that manually designed penalty terms encoding structural (cycle consistency) constraints can effectively regularize both graph-based and image-based representations, yielding outputs that align with our encoded preferences for feasible solutions and closed contours in edge maps.

Part II: Regularization via Feature Matching. In the second part of the thesis, we regularized generative models by penalizing discrepancies between features of generated samples and those of the training data.

Chapter 5 leveraged pretrained feature representations by directly incorporating the Fréchet Inception Distance (FID) [Heu+17] into the Generative Adversarial Network (GAN) [Goo+14] training objective. By minimizing FID during training, the generator was steered using learned features as a proxy for visual quality. This approach revealed intriguing flaws in the FID metric itself. We showed that a generator optimized for FID can achieve intriguingly high quality evaluation results (according to FID itself), while synthesized images clearly not align with the training data from a human perspective. Additionally, we showed that a ranking in terms of image quality provided by FID is highly subjective to the feature extractor used. In particular, data augmentations the feature extractor is trained with play a crucial role into which types of image corruptions FID is more or less sensitive to. These results highlight the need for caution when using aggregate feature distances as evaluation metrics.

In Chapter 6, we addressed a specific under-researched aspect of GAN output quality: the frequency distribution of generated images in the spectral domain. We found that standard GANs exhibit distribution mismatches in the spectral domain. To counter this, we introduced an additional spectral discriminator network that learns to distinguish training data from generated images based on frequency spectra. The training signal provided by this discriminator encourages alignment of generated images with training images across both spatial and frequency domains. This spectral regularization strategy led to better generalization, yielding synthetic images with frequency statistics nearly matching the training data.

Overall, Part II demonstrated that minimizing the discrepancy between extracted features from generated images and training images can be an effective tool to encode a preference for certain features over others. By stepping away from comparing outputs directly, we were able to set a focus on certain aspects of generated images, such as discriminative features from Inception v3 and frequency spectra.

Part III: Data-Based Regularization. In the final part, we moved beyond regularized loss terms and feature matching to impose preferences by weighting and augmenting training data. This strategy treats the selection and importance of training instances as a vehicle for regularization, shaping the bias of the model towards desired properties through data itself.

Chapter 7 introduced a novel framework to regularize Vector Quantized Variational Autoencoders (VQ-VAEs). In particular, we optimized the categorical representation space of a VQ-VAE

for image synthesis by biasing the training data toward a desired attribute: the smiling intensity in face images. For this, we generate additional training examples with globally optimal values of the smiling attribute using an external predictor as a judge. By assigning importance to each training instance and retraining the model on this modified data distribution, we successfully guided the representation space to produce images with increased smiling degrees. This data-driven biasing method effectively encoded a high-level preference without altering the model architecture or adding an explicit term to the training loss.

Chapter 8 extended the idea of preference-driven data weighting and retraining to Neural Architecture Search (NAS). Here, we introduce a novel generative NAS approach by combining a generator network +with a performance predictor to optimize the latent space of the generator towards attributes we seek in neural architectures. Trivially, this can be accuracy for image classification networks. However, with this approach we can also encode multiple objectives at the same time by employing multiple predictors. We demonstrated the effectiveness of this approach by optimizing for accuracy as well as minimal latency. The supporting Chapter 9 contributed a robustness benchmarking dataset of neural architecture design that can be used in NAS, providing measurements of model performance under various perturbations and attacks. This facilitated the NAS procedure in Chapter 8 to search for neural architectures with improved robustness against multiple adversarial attacks, while balancing the trade-offs between them.

In summary, Part III demonstrated that weighting and augmenting training data can encode desired properties in the latent space of generative models.

Across these three parts, the thesis introduced a spectrum of techniques to encode preferences into representation learning models. By introducing penalty terms, feature matching, and preference-based weighted retraining, we have shown how representation learning can be steered toward solutions that are more consistent and aligned with specified criteria. Although implemented in different tasks across computer vision, they share the common theme of improving generalization in specific aspects by integrating inductive biases into the training process.

10.2 Open Problems and Future Directions

We discussed chapter-specific insights and research outlooks in each respective chapter. Here, we want to draw common conclusions and provide future directions from a higher-level perspective.

Computational Overhead of Regularization. Regularization often comes with a computational price, raising important questions about efficiency. We can see this throughout the methods described in this thesis. For example, we discussed the overhead of enumerating many cycle consistency constraints in Section 3.6. This can become intractable, as the number of constraints can grow exponentially. Even after enumerating possible constraints, we also have to add those penalties to the loss (possibly hundreds to thousands), which substantially increases the computational graph, and therefore the complexity of backpropagation. Similar is true for the penalties in Chapter 4. In both cases, we resort to approximations of the cycle constraints, which means we leave potential mistakes unpenalized. Additionally, for both approaches we resort to finetuning the model with penalties, and pretraining them on the unregularized task to improve training times. For our FID-based regularizer, we discuss limitations in Section 5.5. Also here we resort to approximate FID on small batch sizes because of memory constraints. For these methods, we are only able to penalize approximations of the actual structures we are trying to encode (cycle consistency constraints, FID on large batch sizes). For the spectral discriminator in

Chapter 6, we extract 1D features to reduce the number of parameters and keep the discriminator lightweight. This might have diminishing effects as discussed in Section 6.6. In Chapter 7 we discussed that the selecting the next optimal latent space for data augmentation might become infeasible with higher-resolution models. These cases exemplify a general trade-off that we see, namely that more advanced regularization tends to improve model quality at the cost of extra computation or memory requirements. This overhead can make otherwise promising regularizers impractical for large-scale use. Future research could investigate these methods to find possible levers that improve efficiency. We discussed some possibilities in the respective chapters.

A Model is the Sum of its Biases. The NFL theorems motivate the necessity of encoding our preferences into the model via inductive biases [WM97; SG21]. As we have seen in Section 2.4, there is a large number of possibilities to influence the inductive bias of a model, from architectural design, to penalty terms, optimization, and data augmentation. This can be an explicit choice by encoding preferences, like we did throughout this thesis. However, this can also be implicit and easily overlooked. For example, in Subsection 2.4.2 we discussed implicit regularizing effects that architectural design choices induce, such as batch normalization. Or in Subsection 2.4.1 we discussed implicit effects of Stochastic Gradient Descent (SGD), which are still not fully understood and being investigated until today [Gal+25]. All the decisions we make about a model and its training are reflected in its inductive bias, and we might not even be aware of some of them. Examples are the lack of spectral fidelity that we tackled in Chapter 6, or the lack of human alignment of FID that we uncovered in Chapter 5. The more subtle a misalignment between inductive biases of a model and our expectations of its behavior is, the longer it might take to be uncovered.

One example of such a misalignment is the story of susceptibility of representation learning networks to noise, in particular the susceptibility to adversarial attacks [GSS15]. While barely visible to the human eye, these crafted perturbations enforce model decisions that are not understandable from a human perspective. We have seen in Chapter 9 that improving robustness towards these attacks comes with a trade-off with performance on clean images [Tsi+19]. A straightforward solution to this problem seems to be augmenting the training data with such perturbations [CH20]. Indeed, it has been shown that adversarial training not only hardens the models, but also leads to model decisions that are more aligned with human reasoning [GKK23]. However, albeit our intention is simply to harden the network against adversarial attacks, it was shown that this comes at the cost of sacrificing performance of some classes over others, leading to a class-wise performance imbalance [Ben+21]. This is an example of an inductive bias, where regularization (adversarial training) not only has the intended effect of improving robustness, but also side effects that has to be taken into account. In our work on this topic by Medi, Jung, and Keuper [MJK25], we propose a method to improve class-wise fairness when training networks adversarially.

In summary, cases like the misalignment with human ranking in FID (due to data augmenting the backbone), as well as the effect on fairness when training adversarially have shown us that we should take caution when regularizing models to encode preferences. We need a holistic view, and might need to take more metrics and benchmarks into account when training representation learning models under the pretext of guiding them. Since we have seen, a model is the sum of all its inductive biases.

Bibliography

[ACB17] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein Generative Adversarial Networks". In: *International Conference on Machine Learning (ICML)*. 2017.

[Ach+13] P. Acharjya, A. Sinha, S. Sarkar, S. Dey, and S. Ghosh. "A new Approach of Watershed Algorithm Using Distance Transform Applied to Image Segmentation". In: *International Journal of Innovative Research in Computer and Communication Engineering* 1.2 (2013), pp. 185–189.

[AJK24] S. Agnihotri†, S. Jung†, and M. Keuper. "CosPGD: an efficient white-box adversarial attack for pixel-wise prediction tasks". In: *International Conference on Machine Learning (ICML)*. 2024.

[Alb+19] I. Albuquerque, J. Monteiro, T. Doan, B. Considine, T. Falk, and I. Mitliagkas. "Multi-Objective Training of Generative Adversarial Networks with Multiple Discriminators". In: *International Conference on Machine Learning (ICML)*. 2019.

[Alf+22] M. Alfarra, J. C. Pérez, A. Frühstück, P. H. S. Torr, P. Wonka, and B. Ghanem. "On the Robustness of Quality Measures for GANs". In: *European Conference on Computer Vision (ECCV)*. 2022.

[And+11] B. Andres, J. H. Kappes, T. Beier, U. Köthe, and F. A. Hamprecht. "Probabilistic Image Segmentation with Closedness Constraints". In: *International Conference on Computer Vision (ICCV)*. 2011.

[And+12] B. Andres, T. Kroeger, K. L. Briggman, W. Denk, N. Korogod, G. Knott, U. Koethe, and F. A. Hamprecht. "Globally Optimal Closed-Surface Segmentation for Connectomics". In: *European Conference on Computer Vision (ECCV)*. 2012.

[And+13] B. Andres, J. Yarkony, B. S. Manjunath, S. Kirchhoff, E. Turetken, C. Fowlkes, and H. Pfister. "Segmenting Planar Superpixel Adjacency Graphs w.r.t. Non-planar Superpixel Affinity Graphs". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2013.

[And+20] M. Andriushchenko, F. Croce, N. Flammarion, and M. Hein. "Square Attack: A Query-Efficient Black-Box Adversarial Attack via Random Search". In: *European Conference on Computer Vision (ECCV)*. 2020.

[And+23] M. Andriushchenko, F. Croce, M. Müller, M. Hein, and N. Flammarion. "A Modern Look at the Relationship between Sharpness and Generalization". In: *International Conference on Machine Learning (ICML)*. 2023.

[Ara+21] E. Arakelyan, D. Daza, P. Minervini, and M. Cochez. "Complex Query Answering with Neural Link Predictors". In: *International Conference on Learning Representations (ICLR)*. 2021.

[Arb+09] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik. "From Contours to Regions: An Empirical Evaluation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[Arb+11] P. Arbeláez, M. Maire, C. Fowlkes, and J. Malik. "Contour Detection and Hierarchical Image Segmentation". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 33.5 (2011), pp. 898–916.

[Arb+14] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik. "Multiscale Combinatorial Grouping". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[Arb06] P. Arbelaez. "Boundary Extraction in Natural Images Using Ultrametric Contour Maps". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2006.

[Arg+15] I. Arganda-Carreras, S. C. Turaga, D. R. Berger, D. Cireşan, A. Giusti, L. M. Gambardella, J. Schmidhuber, D. Laptev, S. Dwivedi, J. M. Buhmann, T. Liu, M. Seyedhosseini, T. Tasdizen, L. Kamentsky, R. Burget, V. Uher, X. Tan, C. Sun, T. D. Pham, E. Bas, M. G. Uzunbas, A. Cardona, J. Schindelin, and H. S. Seung. "Crowdsourcing the creation of image segmentation algorithms for connectomics". In: *Frontiers in Neuroanatomy* 9 (2015), p. 142.

[AS21] A. Abbas and P. Swoboda. "Combinatorial Optimization for Panoptic Segmentation: A Fully Differentiable Approach". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[AS24] A. Abbas and P. Swoboda. "DOGE-Train: Discrete Optimization on GPU with End-to-end Training". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2024.

[Bai+20] Y. Bai, Y. Guo, J. Wei, L. Lu, R. Wang, and Y. Wang. "Fake Generated Painting Detection via Frequency Analysis". In: *International Conference on Image Processing (ICIP)*. 2020.

[Bam24] Q. Bammey. "Synthbuster: Towards Detection of Diffusion Model Generated Images". In: *IEEE Open Journal of Signal Processing* 5 (2024), pp. 1–9.

[BB12] J. Bergstra and Y. Bengio. "Random Search for Hyper-Parameter Optimization". In: *Journal of Machine Learning Research (JMLR)* 13.10 (2012), pp. 281–305.

[BBC04] N. Bansal, A. Blum, and S. Chawla. "Correlation Clustering". In: *Machine Learning* 56.1–3 (2004), pp. 89–113.

[BCF10] E. Brochu, V. M. Cora, and N. d. Freitas. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. 2010. arXiv: 1012.2599.

[BCV13] Y. Bengio, A. Courville, and P. Vincent. "Representation Learning: A Review and New Perspectives". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35.8 (2013), pp. 1798–1828.

[BD21] D. Barrett and B. Dherin. "Implicit Gradient Regularization". In: *International Conference on Learning Representations (ICLR)*. 2021.

[BDS19] A. Brock, J. Donahue, and K. Simonyan. "Large Scale GAN Training for High Fidelity Natural Image Synthesis". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Bei+14] T. Beier, T. Kroeger, J. Kappes, U. Köthe, and F. Hamprecht. "Cut, Glue, & Cut: A Fast, Approximate Solver for Multicut Partitioning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[Bei+16] T. Beier, B. Andres, U. Köthe, and F. A. Hamprecht. "An Efficient Fusion Move Algorithm for the Minimum Cost Lifted Multicut Problem". In: *European Conference on Computer Vision (ECCV)*. 2016.

[Bei+17] T. Beier, C. Pape, N. Rahaman, T. Prange, S. Berg, D. D. Bock, A. Cardona, G. W. Knott, S. M. Plaza, L. K. Scheffer, U. Koethe, A. Kreshuk, and F. A. Hamprecht. "Multicut brings automated neurite segmentation closer to human performance". In: *Nature Methods* 14.2 (2017), pp. 101–102.

[Bel+17] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. "Neural Combinatorial Optimization with Reinforcement Learning". In: *International Conference on Learning Representations (ICLR) Workshops*. 2017.

[Ben+18] G. Bender, P. Kindermans, B. Zoph, V. Vasudevan, and Q. V. Le. "Understanding and Simplifying One-Shot Architecture Search". In: *International Conference on Machine Learning (ICML)*. 2018.

[Ben+21] P. Benz, C. Zhang, A. Karjauv, and I. S. Kweon. "Robustness May Be at Odds with Fairness: An Empirical Study on Class-wise Accuracy". In: *Advances in Neural Information Processing Systems (NeurIPS) Workshops*. 2021.

[Beu+21] T. Beucler, M. Pritchard, S. Rasp, J. Ott, P. Baldi, and P. Gentine. "Enforcing Analytic Constraints in Neural Networks Emulating Physical Systems". In: *Physical Review Letters* 126 (9 2021), p. 098302.

[Beu79] S. Beucher. "Use of Watersheds in Contour Detection". In: *Proceedings of the International Workshop on Image Processing*. 1979.

[BHK15] T. Beier, F. A. Hamprecht, and J. H. Kappes. "Fusion Moves for Correlation Clustering". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[Bia+20] G. Biau, B. Cadre, M. Sangnier, and U. Tanielian. "Some Theoretical Properties of GANs". In: *Annals of Statistics* 48.3 (2020), pp. 1539–1566.

[Bis06] C. Bishop. *Pattern Recognition and Machine Learning*. Vol. 4. Springer New York, 2006.

[BKC17] V. Badrinarayanan, A. Kendall, and R. Cipolla. "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.12 (2017), pp. 2481–2495.

[Bla+22] C. Blakeney, G. Atkinson, N. Huish, Y. Yan, V. Metsis, and Z. Zong. "Measuring Bias and Fairness in Multiclass Classification". In: *International Conference on Networking, Architecture and Storage (NAS)*. 2022.

[Blo19] M. Blondel. "Structured Prediction with Projection Oracles". In: *Advances in Neural Information Processing Systems (NeurIPS)*. Curran Associates Inc., 2019.

[Bor19] A. Borji. "Pros and Cons of GAN Evaluation Measures". In: *Computer Vision and Image Understanding* 179 (2019), pp. 41–65.

[Bor22] A. Borji. "Pros and Cons of GAN Evaluation Measures: New Developments". In: *Computer Vision and Image Understanding* 215 (2022), p. 103329.

[Bre01] L. Breiman. "Random Forests". In: *Machine Learning* 45.1 (2001), pp. 5–32.

[Bro+20] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei. "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[BS18] S. T. Barratt and R. Sharma. "A Note on the Inception Score". In: *International Conference on Machine Learning (ICML) Workshops*. 2018.

[BSF94] Y. Bengio, P. Simard, and P. Frasconi. "Learning long-term dependencies with gradient descent is difficult". In: *IEEE Transactions on Neural Networks* 5.2 (1994), pp. 157–166.

[BST15] G. Bertasius, J. Shi, and L. Torresani. "DeepEdge: A Multi-Scale Bifurcated Deep Network for Top-Down Contour Detection". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[BTG06] H. Bay, T. Tuytelaars, and L. V. Gool. "SURF: Speeded Up Robust Features". In: *European Conference on Computer Vision (ECCV)*. 2006.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[Can86] J. Canny. "A computational approach to edge detection". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 6 (1986), pp. 679–698.

[Car+10] A. Cardona, S. Saalfeld, S. Preibisch, B. Schmid, A. Cheng, J. Pulokas, P. Tomancak, and V. Hartenstein. "An Integrated Micro- and Macroarchitectural Analysis of the Drosophila Brain by Computer-Assisted Serial Section Electron Microscopy". In: *PLOS Biology* 8.10 (2010), e1000502.

[Car+12] A. Cardona, S. Saalfeld, J. Schindelin, I. Arganda-Carreras, S. Preibisch, M. Longair, P. Tomancak, V. Hartenstein, and R. J. Douglas. "TrakEM2 software for neural circuit reconstruction". In: *PLOS One* 7.6 (2012), e38011.

[Car+19] N. Carlini, A. Athalye, N. Papernot, W. Brendel, J. Rauber, D. Tsipras, I. J. Goodfellow, A. Madry, and A. Kurakin. *On Evaluating Adversarial Robustness*. 2019. arXiv: 1902.06705.

[CCL25] G. E. Constante-Flores, H. Chen, and C. Li. *Enforcing Hard Linear Constraints in Deep Learning Models with Decision Rules*. 2025. arXiv: 2505.13858.

[CF20] M. J. Chong and D. Forsyth. "Effectively Unbiased FID and Inception Score and Where to Find Them". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[CFL24] H. Chen, G. E. C. Flores, and C. Li. "Physics-informed neural networks with hard linear equality constraints". In: *Computers & Chemical Engineering* 189 (2024), p. 108764.

[CFP21] C. R. Chen, Q. Fan, and R. Panda. "CrossViT: Cross-Attention Multi-Scale Vision Transformer for Image Classification". In: *International Conference on Computer Vision (ICCV)*. 2021.

[CG16] T. Chen and C. Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *International Conference on Knowledge Discovery and Data Mining*. 2016.

[CGW21] W. Chen, X. Gong, and Z. Wang. "Neural Architecture Search on ImageNet in Four GPU Hours: A Theoretically Inspired Perspective". In: *International Conference on Learning Representations (ICLR)*. 2021.

[CH20] F. Croce and M. Hein. "Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks". In: *International Conference on Machine Learning (ICML)*. 2020.

[CH22] J. Choi and B. Han. "MCL-GAN: Generative Adversarial Networks with Multiple Specialized Discriminators". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[Cha+19] A. Chatzimichailidis, J. Keuper, F. Pfreundt, and N. R. Gauger. "GradVis: Visualization and Second Order Analysis of Optimization Surfaces during the Training of Deep Neural Networks". In: *Workshop on Machine Learning in High Performance Computing Environments (MLHPC)*. 2019.

[Che+16a] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. "InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.

[Che+16b] Y. Chen, T. Krishna, J. S. Emer, and V. Sze. "14.5 Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks". In: *International Solid-State Circuits Conference (ISSCC)*. 2016.

[Che+18] S. Chen, K. Saulnier, N. Atanasov, D. D. Lee, V. Kumar, G. J. Pappas, and M. Morari. "Approximating Explicit Model Predictive Control Using Constrained Neural Networks". In: *Annual American Control Conference (ACC)*. 2018, pp. 1520–1527.

[Che+19] X. Chen, L. Xie, J. Wu, and Q. Tian. "Progressive Differentiable Architecture Search: Bridging the Depth Gap Between Search and Evaluation". In: *International Conference on Computer Vision (ICCV)*. 2019.

[Che+25] H. Chen, Q. Xiang, J. Hu, M. Ye, C. Yu, H. Cheng, and L. Zhang. "Comprehensive exploration of diffusion models in image generation: a survey". In: *Artificial Intelligence Review* 58.4 (Jan. 2025).

[Chi21] R. Child. "Very Deep VAEs Generalize Autoregressive Models and Can Outperform Them on Images". In: *International Conference on Learning Representations (ICLR)*. 2021.

[Cho+18] Y. Choi, M. Choi, M. Kim, J. Ha, S. Kim, and J. Choo. "StarGAN: Unified Generative Adversarial Networks for Multi-Domain Image-to-Image Translation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Chu+25] B. Chu, X. Xu, X. Wang, Y. Zhang, W. You, and L. Zhou. "FIRE: Robust Detection of Diffusion-Generated Images via Frequency-Guided Reconstruction Error". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2025.

[CLH17] P. Chrabaszcz, I. Loshchilov, and F. Hutter. *A Downsampled Variant of ImageNet as an Alternative to the CIFAR datasets*. 2017. arXiv: `1707.08819`.

[CM02] D. Comaniciu and P. Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 24.5 (2002), pp. 603–619.

[Cog+16] M. Cogswell, F. Ahmed, R. Girshick, L. Zitnick, and D. Batra. "Reducing Overfitting in Deep Networks by Decorrelating Representations". In: *International Conference on Learning Representations (ICLR)*. 2016.

[Cor+23] R. Corvi, D. Cozzolino, G. Poggi, K. Nagano, and L. Verdoliva. "Intriguing Properties of Synthetic Images: From Generative Adversarial Networks to Diffusion Models". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2023.

[CR93] S. Chopra and M. R. Rao. "The Partition Problem". In: *Mathematical Programming* 59.1 (1993), pp. 87–115.

[Cro+21] F. Croce, M. Andriushchenko, V. Sehwag, E. Debenedetti, N. Flammarion, M. Chiang, P. Mittal, and M. Hein. "RobustBench: a standardized adversarial robustness benchmark". In: *NeurIPS Datasets and Benchmarks*. 2021.

[Csi+07] I. Csiszár, G. O. H. Katona, G. Tardos, and G. Wiener. *Entropy, Search, Complexity*. Vol. 16. Springer Science & Business Media, 2007.

[Csi75] I. Csiszár. "I-Divergence Geometry of Probability Distributions and Minimization Problems". In: *The Annals of Probability* (1975), pp. 146–158.

[Cub+19] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le. "AutoAugment: Learning Augmentation Policies from Data". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[Cub+20] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le. "RandAugment: Practical automated data augmentation with a reduced search space". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2020.

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems* 2.4 (Dec. 1989), pp. 303–314.

[CZ20] Y. Chen and B. Zhang. *Learning to Solve Network Flow Problems via Neural Decoding*. 2020. arXiv: 2002.04091.

[CZH19] H. Cai, L. Zhu, and S. Han. "ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Dai+17a] B. Dai, S. Fidler, R. Urtasun, and D. Lin. "Towards Diverse and Natural Image Descriptions via a Conditional GAN". In: *International Conference on Computer Vision (ICCV)*. 2017.

[Dai+17b] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song. "Learning Combinatorial Optimization Algorithms over Graphs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[DB16] A. Dosovitskiy and T. Brox. "Generating Images with Perceptual Similarity Metrics based on Deep Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.

[Dem+06] E. D. Demaine, D. Emanuel, A. Fiat, and N. Immorlica. "Correlation Clustering in General Weighted Graphs". In: *Theoretical Computer Science* 361.2–3 (2006), pp. 172–187.

[Den+09] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[Dev+21] C. Devaguptapu, D. Agarwal, G. Mittal, P. Gopalani, and V. N. Balasubramanian. "On Adversarial Robustness: A Neural Architecture Search perspective". In: *International Conference on Computer Vision (ICCV) Workshops*. 2021.

[DFO20] M. P. Deisenroth, A. A. Faisal, and C. S. Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020.

[DGF16] E. L. Denton, S. Gross, and R. Fergus. "Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks". In: *CoRR* abs/1611.06430 (2016). arXiv: 1611.06430.

[DGM17] I. Durugkar, I. Gemp, and S. Mahadevan. "Generative Multi-Adversarial Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Din+20] J. Ding, C. Zhang, L. Shen, S. Li, B. Wang, Y. Xu, and L. Song. "Accelerating Primal Solution Findings for Mixed Integer Programs Based on Solution Prediction". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2020.

[DK17] S. F. Dodge and L. J. Karam. *A Study and Comparison of Human and Deep Learning Recognition Performance Under Visual Distortions*. 2017. arXiv: 1705.02498.

[DKD17] J. Donahue, P. Krähenbühl, and T. Darrell. "Adversarial Feature Learning". In: *International Conference on Learning Representations (ICLR)*. 2017.

[DKK20] R. Durall, M. Keuper, and J. Keuper. "Watch your Up-Convolution: CNN Based Generative Deep Neural Networks are failing to reproduce Spectral Distributions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[DL97] M. M. Deza and M. Laurent. *Geometry of Cuts and Metrics*. Springer, 1997.

[DMT18] T. Derr, Y. Ma, and J. Tang. "Signed Graph Convolutional Networks". In: *IEEE International Conference on Data Mining*. 2018.

[DN21] P. Dhariwal and A. Nichol. "Diffusion Models Beat GANs on Image Synthesis". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[Don+20] Y. Dong, Q. Fu, X. Yang, T. Pang, H. Su, Z. Xiao, and J. Zhu. "Benchmarking Adversarial Robustness on Image Classification". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[Don+25] M. Dong, Y. Li, Y. Wang, and C. Xu. "Adversarially Robust Neural Architectures". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 47.5 (2025), pp. 4183–4197.

[DTP21] P. Dragone, S. Teso, and A. Passerini. "Neuro-Symbolic Constraint Programming for Structured Prediction". In: *International Workshop on Neural-Symbolic Learning and Reasoning*. 2021.

[Dua+21] Y. Duan, X. Chen, H. Xu, Z. Chen, X. Liang, T. Zhang, and Z. Li. "TransNAS-Bench-101: Improving Transferability and Generalizability of Cross-Task Neural Architecture Search". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[Dur+20] R. Durall, M. Keuper, F. Pfreundt, and J. Keuper. *Unmasking DeepFakes with simple Features*. 2020. arXiv: 1911.00686.

[DV18] V. Dumoulin and F. Visin. *A guide to convolution arithmetic for deep learning*. 2018. arXiv: 1603.07285.

[DY19a] X. Dong and Y. Yang. "One-Shot Neural Architecture Search via Self-Evaluated Template Network". In: *International Conference on Computer Vision (ICCV)*. 2019.

[DY19b] X. Dong and Y. Yang. "Searching for a Robust Neural Architecture in Four GPU Hours". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[DY20] X. Dong and Y. Yang. "NAS-Bench-201: Extending the Scope of Reproducible Neural Architecture Search". In: *International Conference on Learning Representations (ICLR)*. 2020.

[DZ13] P. Dollár and C. L. Zitnick. "Structured Forests for Fast Edge Detection". In: *International Conference on Computer Vision (ICCV)*. 2013.

[EMH19] T. Elsken, J. H. Metzen, and F. Hutter. "Neural Architecture Search: A Survey". In: *Journal of Machine Learning Research (JMLR)* 20.1 (2019), pp. 1997–2017.

[Ess+24] P. Esser, S. Kulal, A. Blattmann, R. Entezari, J. Müller, H. Saini, Y. Levi, D. Lorenz, A. Sauer, F. Boesel, D. Podell, T. Dockhorn, Z. English, K. Lacey, A. Goodwin, Y. Marek, and R. Rombach. *Scaling Rectified Flow Transformers for High-Resolution Image Synthesis*. 2024. arXiv: 2403.03206.

[Est+21] A. Esteva, K. Chou, S. Yeung, N. Naik, A. Madani, A. Mottaghi, Y. Liu, E. Topol, J. Dean, and R. Socher. "Deep learning-enabled medical computer vision". In: *npj Digital Medicine* 4.1 (2021), p. 5.

[Fis36] R. A. Fisher. "The use of multiple measurements in taxonomic problems". In: *Annals of Eugenics* 7.2 (1936), pp. 179–188.

[FJK25] M. Fatima, S. Jung, and M. Keuper. "Corner Cases: How Size and Position of Objects Challenge ImageNet-Trained Models". In: *Transactions on Machine Learning Research (TMLR)* (2025).

[FL19] M. Fey and J. E. Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *International Conference on Learning Representations (ICLR) Workshops*. 2019.

[FNW07] M. A. T. Figueiredo, R. D. Nowak, and S. J. Wright. "Gradient Projection for Sparse Reconstruction: Application to Compressed Sensing and Other Inverse Problems". In: *IEEE Journal of Selected Topics in Signal Processing* 1.4 (2007), pp. 586–597.

[Fra+20] J. Frank, T. Eisenhofer, L. Schönherr, A. Fischer, D. Kolossa, and T. Holz. "Leveraging Frequency Analysis for Deep Fake Image Recognition". In: *International Conference on Machine Learning (ICML)*. 2020.

[Fri01] J. H. Friedman. "Greedy function approximation: A gradient boosting machine." In: *The Annals of Statistics* 29.5 (2001), pp. 1189–1232.

[Fu+23] S. Fu, N. Y. Tamir, S. Sundaram, L. Chai, R. Zhang, T. Dekel, and P. Isola. "DreamSim: learning new dimensions of human visual similarity using synthetic data". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023.

[Gal+25] T. Galanti, Z. S. Siegel, A. Gupte, and T. A. Poggio. "SGD with Weight Decay Secretly Minimizes the Ranks of Your Neural Networks". In: *Conference on Parsimony and Learning*. 2025.

[Gal16] J. Gallier. *Spectral Theory of Unsigned and Signed Graphs. Applications to Graph Clustering: a Survey*. 2016. arXiv: 1601.04692.

[Gas+19] M. Gasse, D. Chételat, N. Ferroni, L. Charlin, and A. Lodi. "Exact Combinatorial Optimization with Graph Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[Gav+25] P. Gavrikov, J. Lukasik, S. Jung, R. Geirhos, B. Lamm, M. J. Mirza, M. Keuper, and J. Keuper. "Can We Talk Models Into Seeing the World Differently?" In: *International Conference on Learning Representations (ICLR)*. 2025.

[GBC16] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[GBD92] S. Geman, E. Bienenstock, and R. Doursat. "Neural Networks and the Bias/Variance Dilemma". In: *Neural Computation* 4.1 (Jan. 1992), pp. 1–58.

[Gei+19] R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. *ImageNet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness*. 2019.

[Gho+20] P. Ghosh, M. S. M. Sajjadi, A. Vergari, M. J. Black, and B. Schölkopf. "From Variational to Deterministic Autoencoders". In: *International Conference on Learning Representations (ICLR)*. 2020.

[Gil+17] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl. "Neural Message Passing for Quantum Chemistry". In: *International Conference on Machine Learning (ICML)*. 2017.

[Gir+14] R. Girshick, J. Donahue, T. Darrell, and J. Malik. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2014.

[GKK23] P. Gavrikov, J. Keuper, and M. Keuper. "An Extended Study of Human-Like Behavior Under Adversarial Training". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2023.

[Góm+18] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. "Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules". In: *ACS Central Science* 4.2 (2018), pp. 268–276.

[Goo+14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. "Generative Adversarial Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014.

[Goo21a] Google LLC. *Advanced Guide to Inception v3 on Cloud TPU*. Accessed: 2021-10-08. 2021. URL: `https://cloud.google.com/tpu/docs/inception-v3-advanced`.

[Goo21b] Google LLC. *Edge TPU Compiler*. Accessed: 2021-11-17. 2021. URL: `https://coral.ai/docs/dev-board/get-started/`.

[Goo21c] Google LLC. *Pixel 3*. Accessed: 2021-11-17. 2021. URL: `https://g.co/kgs/pVRc1Y`.

[Gra+22] J. Grabinski, S. Jung, J. Keuper, and M. Keuper. "FrequencyLowCut Pooling - Plug and Play Against Catastrophic Overfitting". In: *European Conference on Computer Vision (ECCV)*. 2022.

[Gro+19] A. Grover, J. Song, A. Agarwal, K. Tran, A. Kapoor, E. Horvitz, and S. Ermon. "Bias Correction of Learned Generative Models Using Likelihood-Free Importance Weighting". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[GSS15] I. J. Goodfellow, J. Shlens, and C. Szegedy. "Explaining and Harnessing Adversarial Examples". In: *International Conference on Learning Representations (ICLR)*. 2015.

[GSV17] S. Gurumurthy, R. K. Sarvadevabhatla, and R. B. Venkatesh. "DeLiGAN: Generative Adversarial Networks for Diverse and Limited Data". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Gul+17a] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. C. Courville. "Improved Training of Wasserstein GANs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[Gul+17b] I. Gulrajani, K. Kumar, F. Ahmed, A. A. Taiga, F. Visin, D. Vazquez, and A. Courville. "PixelVAE: A Latent Variable Model for Natural Images". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Guo+17] C. Guo, G. Pleiss, Y. Sun, and K. Q. Weinberger. "On Calibration of Modern Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2017.

[Guo+20] M. Guo, Y. Yang, R. Xu, Z. Liu, and D. Lin. "When NAS Meets Robustness: In Search of Robust Architectures Against Adversarial Attacks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[Gur21] Gurobi Optimization, LLC. *Gurobi Optimizer Reference Manual*. 2021. URL: https://www.gurobi.com.

[GW18] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. Pearson, 2018.

[GW90] M. Grötschel and Y. Wakabayashi. "Facets of the clique partitioning polytope". In: *Mathematical Programming* 47.1 (1990), pp. 367–387.

[HB20] D. Hernandez and T. B. Brown. *Measuring the Algorithmic Efficiency of Neural Networks*. 2020. arXiv: 2005.04305.

[HC21] S. Huang and W. Chu. "Searching by Generating: Flexible and Efficient One-Shot NAS With Architecture Generator". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[HD19] D. Hendrycks and T. Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations". In: *International Conference on Learning Representations (ICLR)*. 2019.

[He+15] K. He, X. Zhang, S. Ren, and J. Sun. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *International Conference on Computer Vision (ICCV)*. 2015.

[He+16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[He+19] J. He, S. Zhang, M. Yang, Y. Shan, and T. Huang. "Bi-Directional Cascade Network for Perceptual Edge Detection". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[Hen+20] D. Hendrycks, N. Mu, E. D. Cubuk, B. Zoph, J. Gilmer, and B. Lakshminarayanan. *AugMix: A Simple Data Processing Method to Improve Robustness and Uncertainty*. 2020.

[Heu+17] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. "GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[Hig+17] I. Higgins, L. Matthey, A. Pal, C. P. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. "beta-VAE: Learning Basic Visual Concepts with a Constrained Variational Framework". In: *International Conference on Learning Representations (ICLR)*. 2017.

[HJA20] J. Ho, A. Jain, and P. Abbeel. "Denoising Diffusion Probabilistic Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[HK20] A. Hernández-García and P. König. *Data augmentation instead of explicit regularization*. 2020. arXiv: 1806.03852.

[HLS13] D. W. Hosmer, S. Lemeshow, and R. X. Sturdivant. *Applied Logistic Regression*. Wiley Series in Probability and Statistics. Wiley, 2013.

[Ho+20] K. Ho, A. Kardoost, F. Pfreundt, J. Keuper, and M. Keuper. "A Two-Stage Minimum Cost Multicut Approach to Self-Supervised Multiple Person Tracking". In: *Asian Conference on Computer Vision (ACCV)*. 2020.

[Hou05] Y. P. Hou. "Bounds for the Least Laplacian Eigenvalue of a Signed Graph". In: *Acta Math Sinica* 21.4 (2005), pp. 955–960.

[How+17] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. 2017. arXiv: 1704.04861.

[HRY19] J. Hoffman, D. A. Roberts, and S. Yaida. *Robust Learning with Jacobian Regularization*. 2019. arXiv: 1908.02729.

[HS06] G. E. Hinton and R. R. Salakhutdinov. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313.5786 (2006), pp. 504–507.

[Hua+17] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. "Densely connected convolutional networks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Hua+18] X. Huang, M. Liu, S. Belongie, and J. Kautz. "Multimodal Unsupervised Image-to-Image Translation". In: *European Conference on Computer Vision (ECCV)*. 2018.

[Hua+21] D. Huang, H. Zhang, X. Song, and R. Shibasaki. *Differentiable Projection for Constrained Deep Learning*. 2021. arXiv: 2111.10785.

[HVD15] G. Hinton, O. Vinyals, and J. Dean. *Distilling the Knowledge in a Neural Network*. 2015. arXiv: 1503.02531.

[HYX21] R. Hosseini, X. Yang, and P. Xie. "DSRNA: Differentiable Search of Robust Neural Architectures". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021.

[Ins+16] E. Insafutdinov, L. Pishchulin, B. Andres, M. Andriluka, and B. Schiele. "DeeperCut: A Deeper, Stronger, and Faster Multi-Person Pose Estimation Model". In: *European Conference on Computer Vision (ECCV)*. 2016.

[IS15] S. Ioffe and C. Szegedy. "Batch normalization: Accelerating deep network training by reducing internal covariate shift". In: *International Conference on Machine Learning (ICML)*. 2015.

[ISI17] S. Iizuka, E. Simo-Serra, and H. Ishikawa. "Globally and Locally Consistent Image Completion". In: *ACM Transactions on Graphics (ToG)* 36.4 (2017), p. 107.

[Iso+17] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Jaj+24] D. Jajeśniak, P. Kościelniak, P. Klocek, and M. Mazur. "Interpoint Inception Distance: Gaussian-Free Evaluation of Deep Generative Models". In: *International Conference on Computational Science (ICCS)*. 2024.

[Jan+20] J. Janai, F. Güney, A. Behl, and A. Geiger. "Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art". In: *Foundations and Trends in Computer Graphics and Vision* 12.1–3 (2020), pp. 1–308.

[Jay+24] S. Jayasumana, S. Ramalingam, A. Veit, D. Glasner, A. Chakrabarti, and S. Kumar. "Rethinking FID: Towards a Better Evaluation Metric for Image Generation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2024.

[JBJ20] W. Jin, R. Barzilay, and T. Jaakkola. "Junction Tree Variational Autoencoder for Molecular Graph Generation". In: *Artificial Intelligence in Drug Discovery*. The Royal Society of Chemistry, 2020.

[JG18] D. Jakubovitz and R. Giryes. "Improving DNN Robustness to Adversarial Attacks Using Jacobian Regularization". In: *European Conference on Computer Vision (ECCV)*. 2018.

[Jia+21] Y. Jiang, Z. Huang, X. Pan, C. C. Loy, and Z. Liu. "Talk-to-Edit: Fine-Grained Facial Editing via Dialog". In: *International Conference on Computer Vision (ICCV)*. 2021.

[JK21a] S. Jung and M. Keuper. "Internalized Biases in Fréchet Inception Distance". In: *Advances in Neural Information Processing Systems (NeurIPS) Workshop on Distribution Shifts: Connecting Methods and Applications*. 2021.

[JK21b] S. Jung and M. Keuper. "Spectral Distribution Aware Image Generation". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2021.

[JK22] S. Jung and M. Keuper. "Learning to solve Minimum Cost Multicuts efficiently using Edge-Weighted Graph Convolutional Neural Networks". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. 2022.

[JLB19] C. K. Joshi, T. Laurent, and X. Bresson. *An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem*. 2019. arXiv: 1906.01227.

[JLK23] S. Jung*, J. Lukasik*, and M. Keuper. "Neural Architecture Design and Robustness: A Dataset". In: *International Conference on Learning Representations (ICLR)*. 2023.

[JT21] L. Jing and Y. Tian. "Self-Supervised Visual Feature Learning With Deep Neural Networks: A Survey". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 43.11 (2021), pp. 4037–4058.

[Jun+22] S. Jung, S. Ziegler, A. Kardoost, and M. Keuper. "Optimizing Edge Detection for Image Segmentation with Multicut Penalties". In: *German Conference on Pattern Recognition (GCPR)*. 2022.

[Jun+23] S. Jung, J. C. Schwedhelm, C. Schillings, and M. Keuper. "Happy People–Image Synthesis as Black-Box Optimization Problem in the Discrete Latent Space of Deep Generative Models". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshop: Generative Models for Computer Vision*. 2023.

[KAB15] M. Keuper, B. Andres, and T. Brox. "Motion Trajectory Segmentation via Minimum Cost Multicuts". In: *International Conference on Computer Vision (ICCV)*. 2015.

[Kan+18] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing. "Neural Architecture Search with Bayesian Optimisation and Optimal Transport". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Kap+11] J. H. Kappes, M. Speth, B. Andres, G. Reinelt, and C. Schnörr. "Globally Optimal Image Partitioning by Multicuts". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2011.

[Kap+15a] J. H. Kappes, B. Andres, F. A. Hamprecht, C. Schnörr, S. Nowozin, D. Batra, S. Kim, B. X. Kausler, T. Kröger, J. Lellmann, N. Komodakis, B. Savchynskyy, and C. Rother. "A Comparative Study of Modern Inference Techniques for Structured Discrete Energy Minimization Problems". In: *International Journal of Computer Vision (IJCV)* 115.2 (2015), pp. 155–184.

[Kap+15b] J. H. Kappes, P. Swoboda, B. Savchynskyy, T. Hazan, and C. Schnörr. "Probabilistic Correlation Clustering and Image Partitioning Using Perturbed Multicuts". In: *Scale Space and Variational Methods in Computer Vision*. 2015.

[Kap+16] J. H. Kappes, M. Speth, G. Reinelt, and C. Schnörr. "Higher-order Segmentation via Multicuts". In: *Computer Vision and Image Understanding* 143 (2016), pp. 104–119.

[Kar+17] T. Karras, T. Aila, S. Laine, and J. Lehtinen. "Progressive growing of gans for improved quality, stability, and variation". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Kar+20a] A. Kardoost, K. Ho, P. Ochs, and M. Keuper. "Self-supervised Sparse to Dense Motion Segmentation". In: *Asian Conference on Computer Vision (ACCV)*. 2020.

[Kar+20b] T. Karras, S. Laine, M. Aittala, J. Hellsten, J. Lehtinen, and T. Aila. "Analyzing and Improving the Image Quality of StyleGAN". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[Kar+22] T. Karras, M. Aittala, T. Aila, and S. Laine. "Elucidating the Design Space of Diffusion-Based Generative Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[Kar+25] D. Karageorgiou, S. Papadopoulos, I. Kompatsiaris, and E. Gavves. "Any-Resolution AI-Generated Image Detection by Spectral Learning". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2025.

[Kat04] Y. Katznelson. *An Introduction to Harmonic Analysis*. Cambridge University Press, 2004.

[KB15] D. P. Kingma and J. Ba. "Adam: A Method for Stochastic Optimization". In: *International Conference on Learning Representations (ICLR)*. 2015.

[KBK22] K. Kawaguchi, Y. Bengio, and L. Kaelbling. "Generalization in Deep Learning". In: *Mathematical Aspects of Deep Learning*. Cambridge University Press, Dec. 2022, pp. 112–148.

[Kes+17] N. S. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang. "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Keu+15] M. Keuper, E. Levinkov, N. Bonneel, G. Lavoué, T. Brox, and B. Andres. "Efficient Decomposition of Image and Mesh Graphs by Lifted Multicuts". In: *International Conference on Computer Vision (ICCV)*. 2015.

[Keu+20] M. Keuper, S. Tang, B. Andres, T. Brox, and B. Schiele. "Motion Segmentation Multiple Object Tracking by Correlation Co-Clustering". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 42.1 (2020), pp. 140–153.

[Keu17] M. Keuper. "Higher-Order Minimum Cost Lifted Multicuts for Motion Segmentation". In: *International Conference on Computer Vision (ICCV)*. 2017.

[KGB17] A. Kurakin, I. J. Goodfellow, and S. Bengio. "Adversarial Machine Learning at Scale". In: *International Conference on Learning Representations (ICLR)*. 2017.

[KGC17] J. Kukačka, V. Golkov, and D. Cremers. *Regularization for Deep Learning: A Taxonomy*. 2017. arXiv: 1710.10686.

[KHW19] W. Kool, H. V. Hoof, and M. Welling. "Attention, Learn to Solve Routing Problems!" In: *International Conference on Learning Representations (ICLR)*. 2019.

[Kim+11] S. Kim, S. Nowozin, P. Kohli, and C. D. Yoo. "Higher-Order Correlation Clustering for Image Segmentation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2011.

[Kim+14] S. Kim, C. Yoo, S. Nowozin, and P. Kohli. "Image Segmentation Using Higher-Order Correlation Clustering". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 36 (9 2014), pp. 1761–1774.

[KK18] A. Kardoost and M. Keuper. "Solving Minimum Cost Lifted Multicut Problems by Node Agglomeration". In: *Asian Conference on Computer Vision (ACCV)*. 2018.

[KK21] A. Kardoost and M. Keuper. "Uncertainty in Minimum Cost Multicuts for Image and Motion Segmentation". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2021.

[KLA19] T. Karras, S. Laine, and T. Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[Kly+22] N. Klyuchnikov, I. Trofimov, E. Artemova, M. Salnikov, M. Fedorov, A. Filippov, and E. Burnaev. "NAS-Bench-NLP: Neural Architecture Search Benchmark for Natural Language Processing". In: *IEEE Access* 10 (2022), pp. 45736–45747.

[Kod+17] N. Kodali, J. Abernethy, J. Hays, and Z. Kira. *On Convergence and Stability of GANs*. 2017. arXiv: 1705.07215.

[Kok17] I. Kokkinos. "Ubernet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Kol+21] A. Kolesnikov, A. Dosovitskiy, D. Weissenborn, G. Heigold, J. Uszkoreit, L. Beyer, M. Minderer, M. Dehghani, N. Houlsby, S. Gelly, T. Unterthiner, and X. Zhai. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *International Conference on Learning Representations (ICLR)*. 2021.

[KP09] N. Komodakis and N. Paragios. "Beyond pairwise energies: Efficient optimization for higher-order MRFs". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2009.

[Kra88] D. Kraft. *A Software Package for Sequential Quadratic Programming*. Deutsche Forschungs- und Versuchsanstalt für Luft- und Raumfahrt Köln: Forschungsbericht. Wiss. Berichtswesen d. DFVLR, 1988.

[Kri09] A. Krizhevsky. "Learning Multiple Layers of Features from Tiny Images". In: *University of Toronto*. 2009.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2012.

[Kum+22] N. Kumari, R. Zhang, E. Shechtman, and J. Zhu. "Ensembling Off-the-shelf Models for GAN Training". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[Kun+10] J. Kunegis, S. Schmidt, A. Lommatzsch, J. Lerner, E. W. D. Luca, and S. Albayrak. "Spectral Analysis of Signed Graphs for Clustering, Prediction and Visualization". In: *SIAM International Conference on Data Mining*. 2010.

[KVB88] N. Kanopoulos, N. Vasanthavada, and R. L. Baker. "Design of an image edge detection filter using the Sobel operator". In: *Journal of Solid-State Circuits* 23.2 (1988), pp. 358–367.

[KW14] D. P. Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *International Conference on Learning Representations (ICLR)*. 2014.

[KW17] T. N. Kipf and M. Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Kyn+23] T. Kynkäänniemi, T. Karras, M. Aittala, T. Aila, and J. Lehtinen. "The Role of ImageNet Classes in Fréchet Inception Distance". In: *International Conference on Learning Representations (ICLR)*. 2023.

[Lam21] J. Lamy-Poirier. *Layered gradient accumulation and modular pipeline parallelism: fast and efficient training of large language models*. 2021. arXiv: 2106.02679.

[LBH15] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning". In: *Nature* 521.7553 (2015), p. 436.

[LCK18] Z. Li, Q. Chen, and V. Koltun. "Combinatorial Optimization with Graph Convolutional Networks and Guided Tree Search". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Le+12] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng. "Building High-level Features Using Large Scale Unsupervised Learning". In: *International Conference on Machine Learning (ICML)*. 2012.

[Lec+98] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[Lev+23] E. Levinkov, A. Kardoost, B. Andres, and M. Keuper. "Higher-Order Multicuts for Geometric Model Fitting and Motion Segmentation". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 45.1 (2023), pp. 608–622.

[LHW18] Q. Li, Z. Han, and X. Wu. "Deeper Insights into Graph Convolutional Networks for Semi-Supervised Learning". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2018.

[Li+17] C. Li, W. Chang, Y. Cheng, Y. Yang, and B. Póczos. "MMD GAN: Towards Deeper Understanding of Moment Matching Network". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[Li+18a] J. Li, A. Madry, J. Peebles, and L. Schmidt. "On the Limitations of First-Order Approximation in GAN Dynamics". In: *International Conference on Machine Learning (ICML)*. 2018.

[Li+18b] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia. *Learning Deep Generative Models of Graphs*. 2018. arXiv: 1803.03324.

[Li+21] C. Li, Z. Yu, Y. Fu, Y. Zhang, Y. Zhao, H. You, Q. Yu, Y. Wang, C. Hao, and Y. Lin. "HW-NAS-Bench: Hardware-Aware Neural Architecture Search Benchmark". In: *International Conference on Learning Representations (ICLR)*. 2021.

[Li+23] M. Li, Y. Jiang, Y. Zhang, and H. Zhu. "Medical image analysis using deep learning algorithms". In: *Frontiers in Public Health* 11 (2023).

[Li+24] Z. Li, X. Yang, Y. Zhang, S. Zeng, J. Yuan, J. Liu, Z. Liu, and H. Han. "Deep Graph Reinforcement Learning for Solving Multicut Problem". In: *Transactions on Neural Networks and Learning Systems* (2024), pp. 1–14.

[Lin+19] X. Ling, S. Ji, J. Zou, J. Wang, C. Wu, B. Li, and T. Wang. "DEEPSEC: A Uniform Platform for Security Analysis of Deep Learning Model". In: *IEEE Symposium on Security and Privacy (SP)*. 2019, pp. 673–690.

[Liu+15] Z. Liu, P. Luo, X. Wang, and X. Tang. "Deep Learning Face Attributes in the Wild". In: *International Conference on Computer Vision (ICCV)*. 2015.

[Liu+18] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy. "Progressive Neural Architecture Search". In: *European Conference on Computer Vision (ECCV)*. 2018.

[Liu+19] Y. Liu, M. Cheng, X. Hu, J. Bian, L. Zhang, X. Bai, and J. Tang. "Richer Convolutional Features for Edge Detection". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 41.8 (2019), pp. 1939–1946.

[Liu+21] Z. Liu, Y. Lin, Y. Cao, H. Hu, Y. Wei, Z. Zhang, S. Lin, and B. Guo. "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows". In: *International Conference on Computer Vision (ICCV)*. 2021.

[Liu+22] Z. Liu, H. Mao, C. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. "A ConvNet for the 2020s". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[LJK22] J. Lukasik*, S. Jung*, and M. Keuper. "Learning where to look–generative nas is surprisingly efficient". In: *European Conference on Computer Vision (ECCV)*. 2022.

[LM98] T. Leung and J. Malik. "Contour continuity in region based image segmentation". In: *European Conference on Computer Vision (ECCV)*. 1998.

[Lon+15] M. Long, Y. Cao, J. Wang, and M. Jordan. "Learning Transferable Features with Deep Adaptation Networks". In: *International Conference on Machine Learning (ICML)*. 2015.

[Low04] D. G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision (IJCV)* 60.2 (2004), pp. 91–110.

[LPB17] B. Lakshminarayanan, A. Pritzel, and C. Blundell. "Simple and Scalable Predictive Uncertainty Estimation using Deep Ensembles". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[LRP19] P. Luo, J. Ren, and Z. Peng. "Towards Understanding Regularization in Batch Normalization". In: *International Conference on Learning Representations (ICLR)*. 2019.

[LS08] U. von Luxburg and B. Schoelkopf. *Statistical Learning Theory: Models, Concepts, and Results*. 2008. arXiv: 0810.4752.

[LSD15] J. Long, E. Shelhamer, and T. Darrell. "Fully convolutional networks for semantic segmentation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[LSY19] H. Liu, K. Simonyan, and Y. Yang. "DARTS: Differentiable Architecture Search". In: *International Conference on Learning Representations (ICLR)*. 2019.

[LSZ15] Y. Li, K. Swersky, and R. Zemel. "Generative Moment Matching Networks". In: *International Conference on Machine Learning (ICML)*. 2015.

[LT19] L. Li and A. Talwalkar. "Random Search and Reproducibility for Neural Architecture Search". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2019.

[Lu+18] X. Lu, J. Gonzalez, Z. Dai, and N. Lawrence. "Structured Variationally Auto-encoded Optimization". In: *International Conference on Machine Learning (ICML)*. 2018.

[Luk+21] J. Lukasik, D. Friede, A. Zela, F. Hutter, and M. Keuper. "Smooth Variational Graph Embeddings for Efficient Neural Architecture Search". In: *International Joint Conference on Neural Networks (IJCNN)*. 2021.

[Luo+18] R. Luo, F. Tian, T. Qin, E. Chen, and Tie-Yan Liu. "Neural Architecture Optimization". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Luo+23] X. Luo, Y. Zhu, S. Xu, and D. Liu. "On the Effectiveness of Spectral Discriminators for Perceptual Quality Improvement". In: *International Conference on Computer Vision (ICCV)*. 2023.

[Luz+24] L. Luzi, H. Jenne, C. O. Marrero, and R. Murray. "Using Skew to Assess the Quality of GAN-generated Image Features". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* (2024).

[Ma+20] Q. Ma, S. Ge, D. He, D. Thaker, and I. Drori. "Combinatorial Optimization by Graph Pointer Networks and Hierarchical Reinforcement Learning". In: *Association for the Advancement of Artificial Intelligence (AAAI) Workshops*. 2020.

[Mad+18] A. Madry, A. Makelov, L. Schmidt, D. Tsipras, and A. Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks". In: *International Conference on Learning Representations (ICLR)*. 2018.

[Man+16a] K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. Van Gool. "Convolutional Oriented Boundaries". In: *European Conference on Computer Vision (ECCV)*. 2016.

[Man+16b] E. Mansimov, E. Parisotto, J. L. Ba, and R. Salakhutdinov. "Generating Images from Captions with Attention". In: *International Conference on Learning Representations (ICLR)*. 2016.

[Man+25] L. Manduchi, K. Pandey, C. Meister, R. Bamler, R. Cotterell, S. Däubener, S. Fellenz, A. Fischer, T. Gärtner, M. Kirchler, M. Kloft, Y. Li, C. Lippert, G. d. Melo, E. Nalisnick, B. Ommer, R. Ranganath, M. Rudolph, K. Ullrich, G. V. d. Broeck, J. E. Vogt, Y. Wang, F. Wenzel, F. Wood, S. Mandt, and V. Fortuin. *On the Challenges and Opportunities in Generative AI*. 2025. arXiv: 2403.00025.

[Mao+17] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley. "Least Squares Generative Adversarial Networks". In: *International Conference on Computer Vision (ICCV)*. 2017.

[Mar+01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. "A Database of Human Segmented Natural Images and its Application to Evaluating Segmentation Algorithms and Measuring Ecological Statistics". In: *International Conference on Computer Vision (ICCV)*. 2001.

[Mar+24] S. Marton, S. Lüdtke, C. Bartelt, and H. Stuckenschmidt. "GRANDE: Gradient-Based Decision Tree Ensembles for Tabular Data". In: *International Conference on Learning Representations (ICLR)*. 2024.

[Mas+18] I. Masi, Y. Wu, T. Hassner, and P. Natarajan. "Deep Face Recognition: A Survey". In: *Conference on Graphics, Patterns and Images (SIBGRAPI)*. 2018.

[MBK23] P. Müller, A. Braun, and M. Keuper. "Classification Robustness to Common Optical Aberrations". In: *International Conference on Computer Vision (ICCV)*. 2023.

[MCS19] S. Mo, M. Cho, and J. Shin. "Instance-aware Image-to-Image Translation". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Meh+22] Y. Mehta, C. White, A. Zela, A. Krishnakumar, G. Zabergja, S. Moradian, M. Safari, K. Yu, and F. Hutter. *NAS-Bench-Suite: NAS Evaluation is (Now) Surprisingly Easy*. 2022. arXiv: 2201.13396.

[MGN18] L. Mescheder, A. Geiger, and S. Nowozin. "Which Training Methods for GANs do actually Converge?" In: *International Conference on Machine Learning (ICML)*. 2018.

[MH21] A. Mathiasen and F. Hvilshøj. *Backpropagating through Fréchet Inception Distance*. 2021. arXiv: 2009.14075.

[MH79] D. C. Marr and E. C. Hildreth. "Theory of edge detection". In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 207 (1979), pp. 187–217.

[Mic09] A. Micheli. "Neural Network for Graphs: A Contextual Constructive Approach". In: *Transactions on Neural Networks* 20.3 (2009), pp. 498–511.

[Mik+10] T. Mikolov, M. Karafiát, L. Burget, J. Cernocký, and S. Khudanpur. "Recurrent neural network based language model". In: *Conference of the International Speech Communication Association*. 2010.

[Mit80] T. M. Mitchell. *The Need for Biases in Learning Generalizations*. Tech. rep. Rutgers University, 1980.

[Mit97] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[Miy+18] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida. "Spectral Normalization for Generative Adversarial Networks". In: *International Conference on Learning Representations (ICLR)*. 2018.

[MJK25] T. Medi, S. Jung, and M. Keuper. "FAIR-TAT: Improving Model Fairness Using Targeted Adversarial Training". In: *Winter Conference on Applications of Computer Vision (WACV)*. 2025.

[MO14] M. Mirza and S. Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784.

[Moc74] J. Mockus. "On Bayesian Methods for Seeking the Extremum". In: *Optimization Techniques, IFIP Technical Conference*. Springer, 1974.

[Mok+21] J. Mok, B. Na, H. Choe, and S. Yoon. "AdvRush: Searching for Adversarially Robust Neural Architectures". In: *International Conference on Computer Vision (ICCV)*. 2021.

[MP69] M. Minsky and S. Papert. *Perceptrons; an Introduction to Computational Geometry*. MIT Press, 1969.

[MSF17] P. Márquez-Neila, M. Salzmann, and P. Fua. "Imposing Hard Constraints on Deep Networks: Promises and Limitations". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2017.

[Naz+18] M. Nazari, A. Oroojlooy, L. Snyder, and M. Takac. "Reinforcement Learning for Solving the Vehicle Routing Problem". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2018.

[Ngu+17] T. Nguyen, T. Le, H. Vu, and D. Phung. "Dual Discriminator Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[NH10] V. Nair and G. E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *International Conference on Machine Learning (ICML)*. 2010.

[NTS15] B. Neyshabur, R. Tomioka, and N. Srebro. "In Search of the Real Inductive Bias: On the Role of Implicit Regularization in Deep Learning". In: *International Conference on Learning Representations (ICLR) Workshops*. 2015.

[NVI21] NVIDIA. *Jetson TX2*. Accessed: 2021-11-17. 2021. URL: https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-tx2/.

[OB19] Y. Oulabi and C. Bizer. "Extending Cross-Domain Knowledge Bases with Long Tail Entities using Web Table Data". In: *Advances in Database Technology*. 2019.

[ODO16] A. Odena, V. Dumoulin, and C. Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016). URL: http://distill.pub/2016/deconv-checkerboard/.

[Oor+16] A. van den Oord, N. Kalchbrenner, O. Vinyals, L. Espeholt, A. Graves, and K. Kavukcuoglu. "Conditional Image Generation with PixelCNN Decoders". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.

[OVK17] A. van den Oord, O. Vinyals, and K. Kavukcuoglu. "Neural Discrete Representation Learning". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[Pan+21] T. Pang, X. Yang, Y. Dong, H. Su, and J. Zhu. "Bag of Tricks for Adversarial Training". In: *International Conference on Learning Representations (ICLR)*. 2021.

[Pap+17] C. Pape, T. Beier, P. Li, V. Jain, D. D. Bock, and A. Kreshuk. "Solving large multicut problems for connectomics via domain decomposition". In: *International Conference on Computer Vision (ICCV) Workshops*. 2017.

[Pas+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[Pat+16] D. Pathak, P. Krahenbuhl, J. Donahue, T. Darrell, and A. A. Efros. "Context encoders: Feature learning by inpainting". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Per+17] G. Pereyra, G. Tucker, J. Chorowski, Ł. Kaiser, and G. Hinton. "Regularizing Neural Networks by Penalizing Confident Output Distributions". In: *International Conference on Learning Representations (ICLR) Workshops*. 2017.

[PH24] A. Peleg and M. Hein. "Bias of Stochastic Gradient Descent or the Architecture: Disentangling the Effects of Overparameterization of Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2024.

[Pha+18] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. "Efficient Neural Architecture Search via Parameter Sharing". In: *International Conference on Machine Learning (ICML)*. 2018.

[PM20] J. A. Paulson and A. Mesbah. "Approximate Closed-Loop Robust Model Predictive Control With Guaranteed Stability and Constraint Satisfaction". In: *IEEE Control Systems Letters* 4.3 (2020), pp. 719–724.

[Pon+17] J. Pont-Tuset, P. Arbelaez, J. T.Barron, F. Marques, and J. Malik. "Multiscale Combinatorial Grouping for Image Segmentation and Object Proposal Generation". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 39.1 (2017), pp. 128–140.

[Pra+19] M. O. R. Prates, P. H. C. Avelar, H. Lemos, L. Lamb, and M. Vardi. "Learning to Solve NP-Complete Problems - A Graph Neural Network for the Decision TSP". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2019.

[Pra+23a] K. Prasse, S. Jung, I. B. Bravo, S. Walter, and M. Keuper. "Towards Understanding Climate Change Perceptions: A Social Media Dataset". In: *Advances in Neural Information Processing Systems (NeurIPS) Workshops: Tackling Climate Change with Machine Learning*. 2023.

[Pra+23b] K. Prasse, S. Jung, Y. Zhou, and M. Keuper. "Local Spherical Harmonics Improve Skeleton-Based Hand Action Recognition". In: *German Conference on Pattern Recognition (GCPR)*. 2023.

[Pre96] L. Prechelt. "Early Stopping-But When?" In: *Neural Networks: Tricks of the Trade*. Ed. by Genevieve B. Orr and Klaus-Robert Müller. Vol. 1524. Lecture Notes in Computer Science. Springer, 1996, pp. 55–69.

[Pu+22] M. Pu, Y. Huang, Y. Liu, Q. Guan, and H. Ling. "EDTER: Edge Detection With Transformer". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[PY10] S. J. Pan and Q. Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (2010), pp. 1345–1359.

[PZZ22] G. Parmar, R. Zhang, and J. Zhu. "On Aliased Resizing and Surprising Subtleties in GAN Evaluation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[Rad+21] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, and I. Sutskever. *Learning Transferable Visual Models From Natural Language Supervision*. 2021.

[Ram+21] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, A. Radford, M. Chen, and I. Sutskever. "Zero-Shot Text-to-Image Generation". In: *International Conference on Machine Learning (ICML)*. 2021.

[Ram+22] A. Ramesh, P. Dhariwal, A. Nichol, C. C., and M. Chen. *Hierarchical Text-Conditional Image Generation with CLIP Latents*. 2022. arXiv: 2204.06125.

[Ras21] Raspberry Pi Foundation. *Raspberry Pi 4*. Accessed: 2021-11-17. 2021. URL: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/.

[Raz+14] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. "CNN Features Off-the-Shelf: An Astounding Baseline for Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2014.

[RBB17] J. Rauber, W. Brendel, and M. Bethge. "Foolbox: A Python toolbox to benchmark the robustness of machine learning models". In: *International Conference on Machine Learning (ICML) Workshops*. 2017.

[Rea+17] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, J. Tan, Q. V. Le, and A. Kurakin. "Large-Scale Evolution of Image Classifiers". In: *International Conference on Machine Learning (ICML)*. 2017.

[Rea+19] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le. "Regularized Evolution for Image Classifier Architecture Search". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2019.

[Ree+16] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. "Generative Adversarial Text to Image Synthesis". In: *International Conference on Machine Learning (ICML)*. 2016.

[Rez+21] S. S. C. Rezaei, F. X. Han, D. Niu, M. Salameh, K. G. Mills, S. Lian, W. Lu, and S. Jui. "Generative Adversarial Neural Architecture Search". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2021.

[RFB15] O. Ronneberger, P. Fischer, and T. Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*. 2015.

[Ric+24] J. Ricker, S. Damm, T. Holz, and A. Fischer. "Towards the Detection of Diffusion Model Deepfakes". In: *International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISAPP)*. 2024.

[Rif+11] S. Rifai, P. Vincent, X. Muller, X. Glorot, and Y. Bengio. "Contractive Auto-Encoders: Explicit Invariance During Feature Extraction". In: *International Conference on Machine Learning (ICML)*. 2011.

[RMC15] A. Radford, L. Metz, and S. Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *International Conference on Learning Representations (ICLR)*. 2015.

[Rol+20] M. Rolínek, V. Musil, A. Paulus, M. Vlastelica, C. Michaelis, and G. Martius. "Optimizing Rank-Based Metrics With Blackbox Differentiation". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[Rom+22] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022.

[Ru+21] B. Ru, X. Wan, X. Dong, and M. Osborne. "Interpretable Neural Architecture Search via Bayesian Optimisation with Weisfeiler-Lehman Kernels". In: *International Conference on Learning Representations (ICLR)*. 2021.

[RW05] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Nov. 2005.

[SA17] P. Swoboda and B. Andres. "A Message Passing Algorithm for the Minimum Cost Multicut Problem". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Sah+22a] C. Saharia, W. Chan, S. Saxena, L. Li, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. G. Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi. *Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding*. 2022.

[Sah+22b] C. Saharia, W. Chan, S. Saxena, L. Lit, J. Whang, E. Denton, S. K. S. Ghasemipour, B. K. Ayan, S. S. Mahdavi, R. Gontijo-Lopes, T. Salimans, J. Ho, D. J. Fleet, and M. Norouzi. "Photorealistic Text-to-Image Diffusion Models with Deep Language Understanding". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[Sal+16] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. "Improved Techniques for Training GANs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2016.

[Sau+21] A. Sauer, K. Chitta, J. Müller, and A. Geiger. "Projected GANs Converge Faster". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[SB14] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[Sca+09] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini. "The Graph Neural Network Model". In: *Transactions on Neural Networks* 20.1 (2009), pp. 61–80.

[Sel+19] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. "Learning a SAT Solver from Single-Bit Supervision". In: *International Conference on Learning Representations (ICLR)*. 2019.

[SG21] T. F. Sterkenburg and P. D. Grünwald. "The no-free-lunch theorems of supervised learning". In: *Synthese* 199.3-4 (Dec. 2021), pp. 9979–10015.

[Sha+16] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. de Freitas. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (2016), pp. 148–175.

[She+15] W. Shen, X. Wang, Y. Wang, X. Bai, and Z. Zhang. "DeepContour: A Deep Convolutional Feature Learned by Positive-sharing Loss for Contour Detection". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[Shi+20] H. Shi, R. Pi, H. Xu, Z. Li, J. T. Kwok, and T. Zhang. "Bridging the Gap between Sample-based and One-shot Neural Architecture Search with BONAS". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[Shi+21] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun. "Masked Label Prediction: Unified Message Passing Model for Semi-Supervised Classification". In: *International Joint Conference on Artificial Intelligence (IJCAI)*. 2021.

[Sin+20] S. Sinha, Z. Zhao, A. Goyal, C. Raffel, and A. Odena. "Top-k training of GANs: improving GAN performance by throwing away bad samples". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[SK17] M. Simonovsky and N. Komodakis. "Dynamic Edge-Conditioned Filters in Convolutional Neural Networks on Graphs". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[SK19] C. Shorten and T. M. Khoshgoftaar. "A survey on Image Data Augmentation for Deep Learning". In: *Journal of Big Data* 6.1 (July 2019).

[SLG21] K. Schwarz, Y. Liao, and A. Geiger. "On the Frequency Bias of Generative Models". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[SM00] J. Shi and J. Malik. "Normalized Cuts and Image Segmentation". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 22.8 (2000), pp. 888–905.

[Sno+15] J. Snoek, O. Rippel, K. Swersky, R. Kiros, N. Satish, N. Sundaram, M. Patwary, M. Prabhat, and R. Adams. "Scalable Bayesian Optimization Using Deep Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2015.

[Soh+15] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. "Deep Unsupervised Learning using Nonequilibrium Thermodynamics". In: *International Conference on Machine Learning (ICML)*. 2015.

[Son+19] J. Song, B. Andres, M. Black, O. Hilliges, and S. Tang. "End-to-end Learning for Graph Decomposition". In: *International Conference on Computer Vision (ICCV)*. 2019.

[Sri+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research (JMLR)* 15.56 (2014), pp. 1929–1958.

[Su+18] D. Su, H. Zhang, H. Chen, J. Yi, P. Chen, and Y. Gao. "Is Robustness the Cost of Accuracy? - A Comprehensive Study on the Robustness of 18 Deep Image Classification Models". In: *European Conference on Computer Vision (ECCV)*. 2018.

[Sun+17] C. Sun, A. Shrivastava, S. Singh, and A. Gupta. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *International Conference on Computer Vision (ICCV)*. 2017.

[Sun+18] D. Sungatullina, E. Zakharov, D. Ulyanov, and V. Lempitsky. "Image Manipulation with Perceptual Discriminators". In: *European Conference on Computer Vision (ECCV)*. 2018.

[SZ15] K. Simonyan and A. Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations (ICLR)*. 2015.

[Sze+14] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus. "Intriguing properties of neural networks". In: *International Conference on Learning Representations (ICLR)*. 2014.

[Sze+15] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going Deeper with Convolutions". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015.

[Sze+16] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the Inception Architecture for Computer Vision". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[Sze+17] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi. "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2017.

[Sze22] R. Szeliski. *Computer Vision: Algorithms and Applications*. 2nd. Springer, 2022.

[Tan+21] S. Tang, R. Gong, Y. Wang, A. Liu, J. Wang, X. Chen, F. Yu, X. Liu, D. X. Song, A. L. Yuille, P. H. S. Torr, and D. Tao. *RobustART: Benchmarking Robustness on Architecture Design and Training Techniques*. 2021. arXiv: 2109.05211.

[TDH20] A. Tripp, E. Daxberger, and J. M. Hernández-Lobato. "Sample-Efficient Optimization in the Latent Space of Deep Generative Models via Weighted Retraining". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[The+21] A. Thebelt, J. Kronqvist, M. Mistry, R. M. Lee, N. Sudermann-Merx, and R. Misener. "ENTMOOT: A Framework for Optimization over Ensemble Tree Models". In: *Computers & Chemical Engineering* 151 (2021), p. 107343.

[Tib96] R. Tibshirani. "Regression Shrinkage and Selection via the Lasso". In: *Journal of the Royal Statistical Society. Series B (Methodological)* 58.1 (1996), pp. 267–288.

[TIF24] A. Torralba, P. Isola, and W.T. Freeman. *Foundations of Computer Vision*. Adaptive Computation and Machine Learning series. MIT Press, 2024.

[TKD25] B. Tang, E. B. Khalil, and J. Drgoňa. *Learning to Optimize for Mixed-Integer Non-linear Programming with Feasibility Guarantees*. 2025. arXiv: 2410.11061.

[TL19] M. Tan and Q. Le. "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks". In: *International Conference on Machine Learning (ICML)*. 2019.

[TOB16] L. Theis, A. van den Oord, and M. Bethge. "A note on the evaluation of generative models". In: *International Conference on Learning Representations (ICLR)*. 2016.

[Tou+21] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jegou. "Training data-efficient image transformers & distillation through attention". In: *International Conference on Machine Learning (ICML)*. 2021.

[Tsi+19] D. Tsipras, S. Santurkar, L. Engstrom, A. Turner, and A. Madry. "Robustness May Be at Odds with Accuracy". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Tui+22] D. Tuia, B. Kellenberger, S. Beery, B. R. Costelloe, S. Zuffi, B. Risse, A. Mathis, M. W. Mathis, F. v. Langevelde, T. Burghardt, R. Kays, H. Klinck, M. Wikelski, I. D. Couzin, G. v. Horn, M. C. Crofoot, C. V. Stewart, and T. Berger-Wolf. "Perspectives in Machine Learning for Wildlife Conservation". In: *Nature Communications* 13.1 (2022), p. 792.

[Vas+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. "Attention is All You Need". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[VFJ15] O. Vinyals, M. Fortunato, and N. Jaitly. "Pointer Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2015.

[Vin+10] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. Manzagol. "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion". In: *Journal of Machine Learning Research (JMLR)* 11 (2010), pp. 3371–3408.

[VR22] T. Vu and R. Raich. "On Asymptotic Linear Convergence of Projected Gradient Descent for Constrained Least Squares". In: *IEEE Transactions on Signal Processing* 70 (2022), pp. 4061–4076.

[VS91] L. Vincent and P. Soille. "Watersheds in digital spaces: an efficient algorithm based on immersion simulations". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 13.6 (1991), pp. 583–598.

[VWT14] V. Vineet, J. Warrell, and P. H. S. Torr. "Filter-based mean-field inference for random fields with higher-order terms and product label-spaces". In: *International Journal of Computer Vision (IJCV)* 110.3 (2014), pp. 290–307.

[Wan+20] S. Wang, O. Wang, R. Zhang, A. Owens, and A. A. Efros. "CNN-Generated Images Are Surprisingly Easy to Spot... for Now". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[Wel62] B. P. Welford. "Note on a Method for Calculating Corrected Sums of Squares and Products". In: *Technometrics* 4.3 (1962), pp. 419–420.

[Wen+20] W. Wen, H. Liu, Y. Chen, H. H. Li, G. Bender, and P. Kindermans. "Neural Predictor for Neural Architecture Search". In: *European Conference on Computer Vision (ECCV)*. 2020.

[Whi+21] C. White, A. Zela, B. Ru, Y. Liu, and F. Hutter. "How Powerful are Performance Predictors in Neural Architecture Search?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[WKM20] C. Wei, S. Kakade, and T. Ma. "The Implicit and Explicit Regularization Effects of Dropout". In: *International Conference on Machine Learning (ICML)*. 2020.

[WM97] D. Wolpert and W. Macready. "No Free Lunch Theorems for Optimization". In: *IEEE Transactions on Evolutionary Computation* 1.1 (1997), pp. 67–82.

[WNS21a] C. White, W. Neiswanger, and Y. Savani. "BANANAS: Bayesian Optimization with Neural Architectures for Neural Architecture Search". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2021.

[WNS21b] C. White, S. Nolen, and Y. Savani. "Exploring the loss landscape in neural architecture search". In: *Conference on Uncertainty in Artificial Intelligence (UAI)*. 2021.

[Wol02] D. H. Wolpert. "The Supervised Learning No-Free-Lunch Theorems". In: *Soft Computing and Industry: Recent Applications*. Springer London, 2002, pp. 25–42.

[Wu+19] B. Wu, X. Dai, P. Zhang, Y. Wang, F. Sun, Y. Wu, Y. Tian, P. Vajda, Y. Jia, and K. Keutzer. "FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[Wu+21] J. Wu, X. Dai, D. Chen, Y. Chen, M. Liu, Y. Yu, Z. Wang, Z. Liu, M. Chen, and L. Yuan. "Stronger NAS with Weaker Predictors". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[Wu+24] Y. Wu, F. Liu, C. Simon-Gabriel, G. Chrysos, and V. Cevher. "Robust NAS under adversarial training: benchmark, theory, and beyond". In: *International Conference on Learning Representations (ICLR)*. 2024.

[WW23] Z. Wang and L. Wu. "Theoretical Analysis of the Inductive Biases in Deep Convolutional Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2023.

[Xie+19a] C. Xie, Y. Wu, L. van der Maaten, A. L. Yuille, and K. He. "Feature Denoising for Improving Adversarial Robustness". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2019.

[Xie+19b] S. Xie, H. Zheng, C. Liu, and L. Lin. "SNAS: Stochastic Neural Architecture Search". In: *International Conference on Learning Representations (ICLR)*. 2019.

[Xie+21] C. Xie, M. Tan, B. Gong, A. Yuille, and Q. V. Le. *Smooth Adversarial Training*. 2021. arXiv: 2006.14536.

[Xie+25] E. Xie, J. Chen, J. Chen, H. Cai, H. Tang, Y. Lin, Z. Zhang, M. Li, L. Zhu, Y. Lu, and S. Han. "SANA: Efficient High-Resolution Text-to-Image Synthesis with Linear Diffusion Transformers". In: *International Conference on Learning Representations (ICLR)*. 2025.

[Xil21a] Xilinx Inc. *Vivado High-Level Synthesis*. Accessed: 2021-11-17. 2021. URL: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[Xil21b] Xilinx Inc. *Zynq 7000 SoC ZC706 Evaluation Kit*. Accessed: 2021-11-17. 2021. URL: https://www.xilinx.com/products/boards-and-kits/ek-z7-zc706-g.html.

[XT15] S. Xie and Z. Tu. "Holistically-Nested Edge Detection". In: *International Conference on Computer Vision (ICCV)*. 2015.

[Xu+19] K. Xu, W. Hu, J. Leskovec, and S. Jegelka. "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations (ICLR)*. 2019.

[Xu+20] Y. Xu, L. Xie, X. Zhang, X. Chen, G. Qi, Q. Tian, and H. Xiong. "PC-DARTS: Partial Channel Connections for Memory-Efficient Architecture Search". In: *International Conference on Learning Representations (ICLR)*. 2020.

[Yan+19] T. Yang, M. D. Collins, Y. Zhu, J. Hwang, T. Liu, X. Zhang, V. Sze, G. Papandreou, and L. Chen. *DeeperLab: Single-Shot Image Parser*. 2019. arXiv: 1902.05093.

[Yan+20] S. Yan, Y. Zheng, W. Ao, X. Zeng, and M. Zhang. "Does Unsupervised Architecture Representation Learning Help Neural Architecture Search?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

[Yan+21] S. Yan, C. White, Y. Savani, and F. Hutter. "NAS-Bench-x11 and the Power of Learning Curves". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[YDF19] N. Yu, L. Davis, and M. Fritz. "Attributing Fake Images to GANs: Learning and Analyzing GAN Fingerprints". In: *International Conference on Computer Vision (ICCV)*. 2019.

[Yin+19a] D. Yin, R. G. Lopes, J. Shlens, E. D. Cubuk, and J. Gilmer. "A Fourier Perspective on Model Robustness in Computer Vision". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[Yin+19b] C. Ying, A. Klein, E. Christiansen, E. Real, K. Murphy, and F. Hutter. "NAS-Bench-101: Towards Reproducible Neural Architecture Search". In: *International Conference on Machine Learning (ICML)*. 2019.

[Yos+14] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2014.

[Yua+21] L. Yuan, Y. Chen, T. Wang, W. Yu, Y. Shi, Z. Jiang, F. E. Tay, J. Feng, and S. Yan. "Tokens-to-Token ViT: Training Vision Transformers From Scratch on ImageNet". In: *International Conference on Computer Vision (ICCV)*. 2021.

[Zal+25] A. Zalcher, N. Wasserman, R. Beliy, O. Heinimann, and M. Irani. *Don't Judge Before You CLIP: A Unified Approach for Perceptual Tasks*. 2025. arXiv: 2503.13260.

[Zel+20] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter. "Understanding and Robustifying Differentiable Architecture Search". In: *International Conference on Learning Representations (ICLR)*. 2020.

[Zel+22] A. Zela, J. N. Siems, L. Zimmer, J. Lukasik, M. Keuper, and F. Hutter. "Surrogate NAS Benchmarks: Going Beyond the Limited Search Spaces of Tabular NAS Benchmarks". In: *International Conference on Learning Representations (ICLR)*. 2022.

[ZF14] M. D. Zeiler and R. Fergus. "Visualizing and Understanding Convolutional Networks". In: *European Conference on Computer Vision (ECCV)*. 2014.

[Zha+16] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks". In: *IEEE Signal Processing Letters* 23.10 (2016), pp. 1499–1503.

[Zha+17a] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. "Understanding deep learning requires rethinking generalization". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Zha+17b] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. "Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks". In: *International Conference on Computer Vision (ICCV)*. 2017.

[Zha+17c] X. Zhang, Z. Li, C. Change Loy, and D. Lin. "Polynet: A pursuit of structural diversity in very deep networks". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017.

[Zha+18a] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. "mixup: Beyond Empirical Risk Minimization". In: *International Conference on Learning Representations (ICLR)*. 2018.

[Zha+18b] H. Zhang, T. Xu, H. Li, S. Zhang, X. Wang, X. Huang, and D. N. Metaxas. "Stackgan++: Realistic image synthesis with stacked generative adversarial networks". In: *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 41.8 (2018), pp. 1947–1962.

[Zha+18c] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Zha+19] M. Zhang, S. Jiang, Z. Cui, R. Garnett, and Y. Chen. "D-VAE: A Variational Autoencoder for Directed Acyclic Graphs". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2019.

[Zha+20] P. Zhao, P. Chen, P. Das, K. N. Ramamurthy, and X. Lin. "Bridging Mode Connectivity in Loss Landscapes and Adversarial Robustness". In: *International Conference on Learning Representations (ICLR)*. 2020.

[Zha22] D. Zhao. "Combining Implicit and Explicit Regularization for Efficient Learning in Deep Networks". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2022.

[Zhe+15] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Su, D. Du, C. Huang, and P. H. S. Torr. "Conditional Random Fields as Recurrent Neural Networks". In: *International Conference on Computer Vision (ICCV)*. 2015.

[Zhu+17a] J. Zhu, T. Park, P. Isola, and A. A. Efros. "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: *International Conference on Computer Vision (ICCV)*. 2017.

[Zhu+17b] J. Zhu, R. Zhang, D. Pathak, T. Darrell, A. A. Efros, O. Wang, and E. Shechtman. "Toward Multimodal Image-to-Image Translation". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2017.

[Zhu+24] K. Zhu, J. Wang, J. Zhou, Z. Wang, H. Chen, Y. Wang, L. Yang, W. Ye, Y. Zhang, N. Gong, and X. Xie. "PromptRobust: Towards Evaluating the Robustness of Large Language Models on Adversarial Prompts". In: *ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis*. 2024.

[ZKC19] X. Zhang, S. Karaman, and S. Chang. "Detecting and Simulating Artifacts in GAN Fake Images". In: *International Workshop on Information Forensics and Security (WIFS)*. 2019.

[ZL17] B. Zoph and Q. V. Le. "Neural Architecture Search with Reinforcement Learning". In: *International Conference on Learning Representations (ICLR)*. 2017.

[Zop+18] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. "Learning Transferable Architectures for Scalable Image Recognition". In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2018.

[Zop+20] B. Zoph, G. Ghiasi, T. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. V. Le. "Rethinking Pre-training and Self-training". In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2020.

This page intentionally left blank.

Chapter A

Representation Learning in Graphs with Discrete Constraints

In this supplementary material, we provide several additional details, ablations and visualizations. We provide:

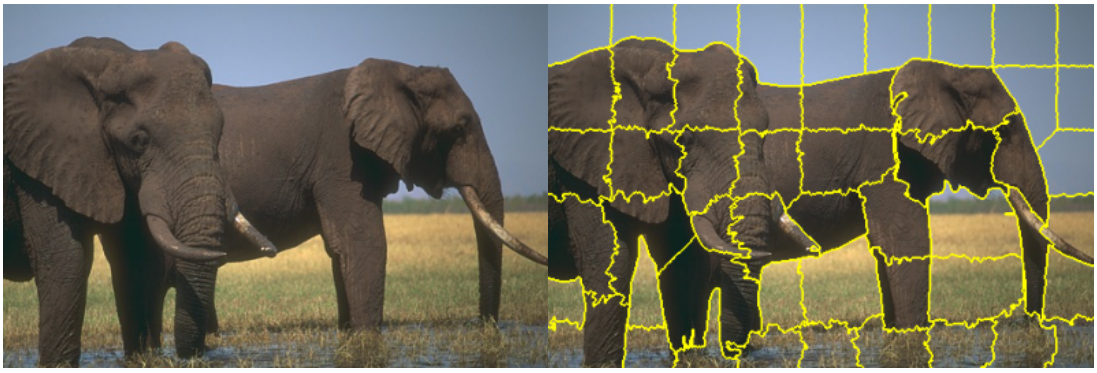
- Section A.1: An example graph from an image segmentation problem, providing some intuition on the practical quality of results.
- Section A.2: Shows statistics of the contributed training datasets IrisMP and RandomMP.
- Section A.3: Details on additional test datasets used.
- Section A.5: Experiments for domain specific finetuning of our models. While the domain specific training data is very scarce, these experiments show the promise of learnable multicut solvers.
- Section A.6: Additional embedding space visualizations (as in Figure 3.7).
- Section A.4: Training curves of training runs on RandomMP.

A.1 Multicut Segmentation Example

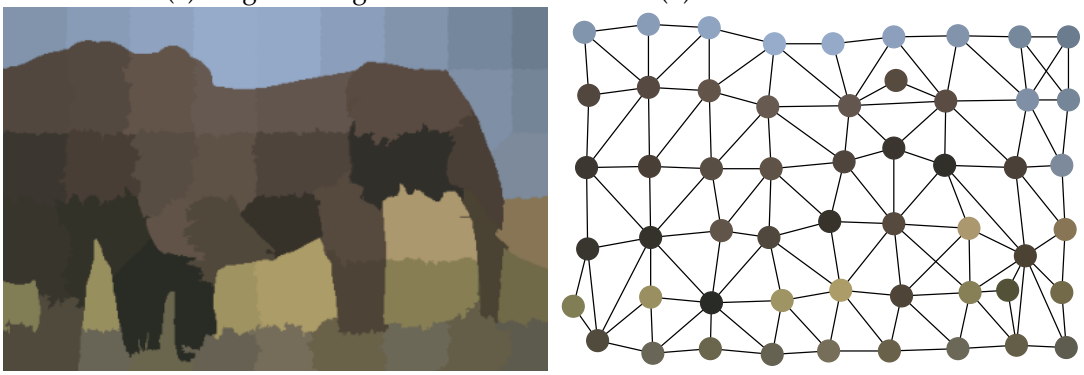
For visualization purposes, we generate a small graph based on image segmentation of a training sample from the Berkeley Segmentation Dataset, BSDS300 [Mar+01], given in Figure A.1(a). First, the gradients of the original image are computed using a Sobel filter. Then, the watershed transformation is computed with 50 desired markers, and a compactness of 0.0014. This results in the image consisting of 54 segments as shown in Figure A.1(b). Then the image is superpixelated computing the mean color of each resulting segment. E.g., $c_i = \sum_{x \in S_i} x/|S_i|$ is the mean color of superpixel i, where S_i is the set of pixels it contains. x is the color information of a pixel, yielding Figure A.1(c). From these superpixels the adjacency graph is constructed by connecting superpixels with a squared spatial distance less than 2. A positive weight w_{ij} between two superpixels i and j is calculated using color similarity by applying a Gaussian kernelwith $\sigma = 0.1$ on their color distance: $w_{ij} = e^{-|c_i - c_j|/\sigma}$. The resulting graph is shown in Figure A.2. From positive weights that represent superpixel similarity, positive and negative edge weights for the multicut problem can be derived using the logit function.

Since the graph is small, we can compute its optimal solution using an ILP solver (see Figure A.3(a)). The optimal objective value is denoted by c_{OPT}. The optimal solution yields an objective value of $c_{OPT} = -106.998$, while Greedy Additive Edge Contraction (GAEC) performs slightly worse with $c = -106.587$, thus providing optimality ratio $c_r = 0.9961$.

For comparison, we show multicuts computed by a two of our models in Figure A.4. While the resulting optimality ratios are lower, the results are still of comparable practical quality, especially when considering the model trained on RandomMP.

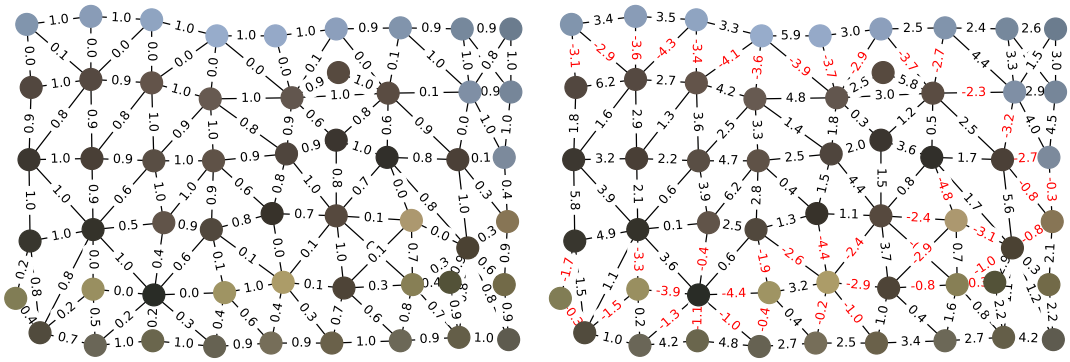


(a) Original image. (b) Watershed transformed.



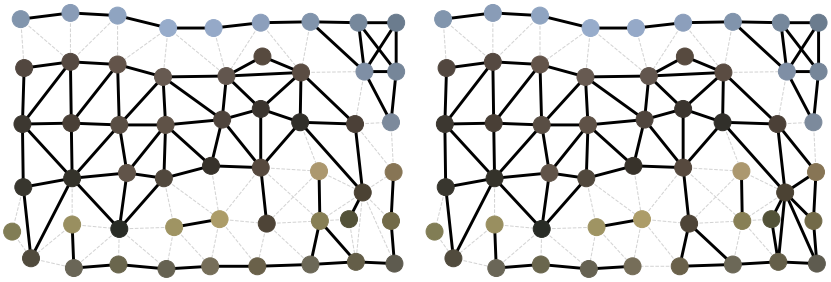
(c) Superpixel image. (d) Regional adjacency graph.

Figure A.1: Expressing image segmentation by superpixelization as a multicut problem instance.

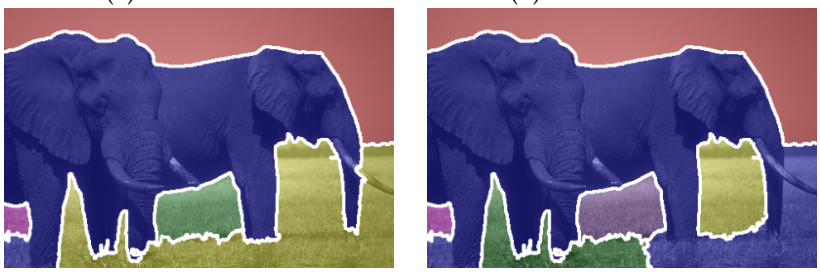


(a) Similarities (b) real-valued weights

Figure A.2: Example graph. **(a)** Weights are similarities based on a Gaussian kernel. **(b)** Weights are log-odds of (a) and define a multicut problem instance.

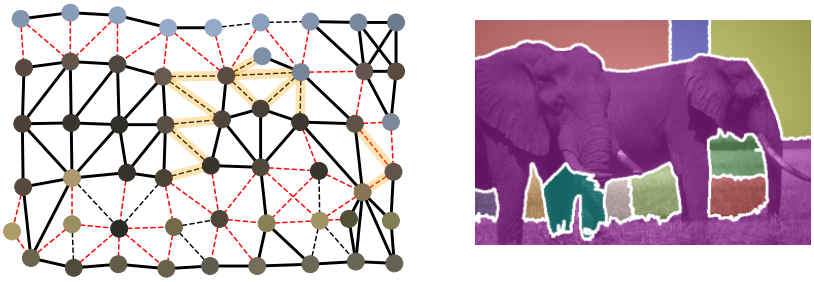


(a) ILP solution. (b) GAEC solution.

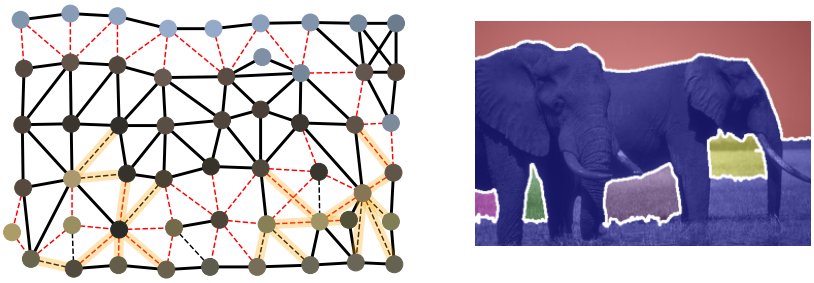


(c) ILP segmentation. (d) GAEC segmentation.

Figure A.3: Multicuts of the graph using different solvers. **(a)** shows the optimal cut, and **(b)** shows the cut computed with GAEC. Dotted, gray lines indicate that the edge is cut, and solid, black lines indicate otherwise. **(c)** and **(d)** depict the resulting segmentations.



(a) `GCN_W_BN`, IrisMP, depth $= 12$, $c_r = 0.8948$.



(b) `GCN_W_BN`, RandomMP, depth $= 20$, $c_r = 0.9899$.

Figure A.4: Multicuts and resulting segmentations of the example graph by two of our trained models. Highlighted edges are removed by the model without partitioning the corresponding nodes. Hence, those cuts violate cycle consistencies and the edges are reinstated after rounding.

A.2 Training Dataset Statistics

Table A.1 and Table A.2 show statistics of the IrisMP and RandomMP datasets. Measures denoted with an overline show the corresponding mean and standard deviation over all instances in the split. Cardinalities $|V|$ and $|E|$ denote the number of nodes and edges in the graph. We show the minimal weight denoted by w_{min}, the average weight denoted by w_{avg}, and the maximal weight denoted by w_{max}. The optimal objective value is denoted by $c(\tilde{\mathbf{y}})$, where $\tilde{\mathbf{y}}$ is the decision vector given by the optimal solution. Additionally, we show upper and lower bounds for the objective values. The lower bound is given by cutting all negative edges, denoted by $c(\mathbf{y}^-)$, where $\mathbf{y}^-$ is the decision vector containing $y_e = 1$ if the edge weight of e is negative, and $y_e = 0$ otherwise. The upper bound is, correspondingly, given by cutting all positive edges and denoted by $c(\mathbf{y}^+)$. We also include the value of the trivial solution (cutting all edges), denoted by $c(\mathbf{1})$. Statistics denoted by $c(\cdot)$ are normalized by the number of edges.

Table A.1: IrisMP statistics.

	Graph Stats		
Split	Train	Eval	Test
$\lvert\mathcal{D}\rvert$	20 000	1000	1000
$\overline{\lvert V\rvert}$	20 ± 3	20 ± 3	20 ± 3
$\overline{\lvert E\rvert}$	194 ± 50	196 ± 49	192 ± 50
	Weights		
$\overline{w_{\text{min}}}$	-4.51 ± 0.29	-4.51 ± 0.33	-4.51 ± 0.31
$\overline{w_{\text{avg}}}$	-0.41 ± 0.41	-0.42 ± 0.41	-0.40 ± 0.42
$\overline{w_{\text{max}}}$	4.57 ± 0.12	4.57 ± 0.12	4.57 ± 0.11
	Objective Values		
$c(\mathbf{y}^-)$	-1.169	-1.177	-1.168
$c(\tilde{\mathbf{y}})$	-1.103	-1.111	-1.100
$c(\mathbf{1})$	-0.407	-0.426	-0.405
$c(\mathbf{y}^+)$	0.762	0.751	0.762

Table A.2: RandomMP statistics.

Graph Stats			
Split	Train	Eval	Test
$\lvert\mathcal{D}\rvert$	20 000	1000	1000
$\overline{\lvert V\rvert}$	180 ± 30	180 ± 29	179 ± 31
$\overline{\lvert E\rvert}$	686 ± 115	685 ± 114	684 ± 118
Weights			
$\overline{w_{\text{min}}}$	-9.36 ± 1.09	-9.37 ± 1.08	-9.39 ± 1.12
$\overline{w_{\text{avg}}}$	-0.00 ± 0.10	0.00 ± 0.10	0.00 ± 0.10
$\overline{w_{\text{max}}}$	9.38 ± 1.09	9.37 ± 1.10	9.38 ± 1.10
Objective Values			
$c(\mathbf{y}^-)$	-1.196	-1.193	-1.195
$c(\tilde{\mathbf{y}})$	-0.844	-0.842	-0.841
$c(\mathbf{1})$	-0.001	0.002	0.002
$c(\mathbf{y}^+)$	1.196	1.195	1.197

A.3 Test Datasets

Table A.3 and Table A.4 show statistics of the additional test datasets used in our experiments.

Table A.3: BSDS300 statistics.

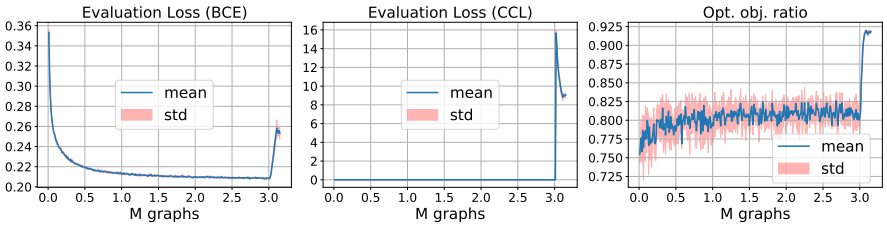
Graph Stats	
$\lvert\mathcal{D}\rvert$	100
$\overline{\lvert V\rvert}$	1551 ± 777
$\overline{\lvert E\rvert}$	4432 ± 2269
Weights	
$\overline{w_{\text{min}}}$	-10.77 ± 3.74
$\overline{w_{\text{avg}}}$	0.48 ± 0.57
$\overline{w_{\text{max}}}$	10.27 ± 2.32
Objective Values	
$c(\mathbf{y}^-)$	-0.738
$c(\tilde{\mathbf{y}})$	-0.669
$c(\mathbf{1})$	0.497
$c(\mathbf{y}^+)$	1.235

Table A.4: CREMI statistics.

Graph Stats	
$\lvert\mathcal{D}\rvert$	3
$\overline{\lvert V\rvert}$	$31\,988 \pm 4769$
$\overline{\lvert E\rvert}$	$212\,686 \pm 24\,767$
Weights	
$\overline{w_{\text{min}}}$	-6.91 ± 0.00
$\overline{w_{\text{avg}}}$	-0.15 ± 0.26
$\overline{w_{\text{max}}}$	79.52 ± 12.63
Objective Values	
$c(\mathbf{y}^-)$	-1.013
$c(\tilde{\mathbf{y}})$	-1.002
$c(\mathbf{1})$	-0.135
$c(\mathbf{y}^+)$	0.878

A.4 Training Curves on RandomMP

Figure A.5 shows mean evaluation plots of five training runs of GCN_W_BN on 3.15M instances of RandomMP. No Cycle Consistency Loss (CCL) was applied in the first 3M instances. Then, α was linearly increased over 100k instances to $\alpha = 0.01$. Afterwards, the training continued for 50k instances with $\alpha = 0.01$. The node embedding dimensionality was set to 128 and the number of Message Passing Neural Network (MPNN) layers to 20. The Multilayer Perceptron (MLP) edge classifier consists of 2 hidden layers with 256 neurons. Optimization was performed with Adam [KB15] (0.001 learning rate, $5 \cdot 10^{-4}$ weight decay, $(0.9, 0.999)$ betas) and a batch size of 200. Each training was performed on a MEGWARE Gigabyte G291-Z20 server on one NVIDIA Quadro RTX 8000 GPU and took 24hrs on average, whereof the training time of the last 150k (CCL) instances took around 18hrs.



(a) BCE (b) CCL (c) Objective

Figure A.5: Mean and standard deviation of **(a)** BCE evaluation loss, **(b)** CCL evaluation loss, and **(c)** evaluation optimal objective ratio of 5 training runs of GCN_W_BN on RandomMP.

A.5 Finetuning Experiments

Originally, generating synthetic training data for our method was mainly driven by the lack of suitable training data available. The benchmark datasets mentioned consist of only a few problem instances (BSDS300: 100, CREMI: 3, Knott: 24). Although BSDS300 consists of 100 training and 100 testing images, the multicut problem instances provided by the OpenGM benchmark are solely based on the test images. This is the case, because training images were used to train a model that derives edge weights for the testing images and are discarded afterwards. Nevertheless, we consider it interesting to see how the model behaves in this environment of scarce training data, and whether finetuning can help to boost model performance on a specific task. Therefore, we ran the following additional experiments:

1) We trained GCN_W_BN from scratch on training splits of these datasets.

2) We finetuned the best performing model of Table 1 (GCN_W_BN trained on RandomMP - referred to as RMP in the following) on training splits of these datasets.

We split BSDS300 into 70/20/10 (train/eval/test), CREMI into 2/1 (train/eval) and Knott into 18/6 (train/eval). The performance after training is evaluated on the eval split and compared to the performance as optimality ratio of the RandomMP trained GCN_W_BN (RMP) on the eval split. Results are given in Table A.5. Please note that these results can not be directly compared to Table 3.3 since the whole datasets were evaluated in Table 3.3.

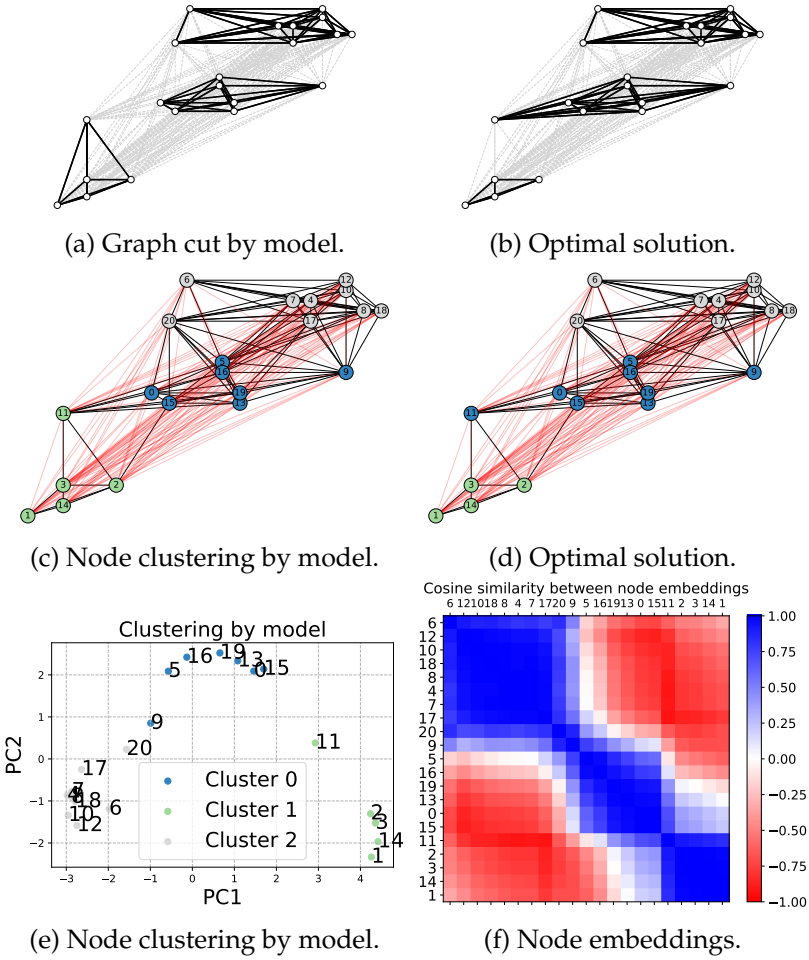
Table A.5: Domain specific training of GCN_W_BN from scratch and finetuned on the scarce available training data for each task, compared to the general purpose model trained on RandomMP (RMP). Results show that domain specific properties can be learned from few samples but pretraining can help in general.

	RMP	from scratch	RMP + finetuned
BSDS300	0.8818	0.8834	0.8818
Knott	0.8386	0.8335	0.9249
CREMI	0.9068	0.8081	0.9365

Still, the results indicate that when trained on BSDS300 from scratch, the model improves on the eval split from 0.8818 to 0.8834. We were not able to find models that outperform RMP trained on Knott and CREMI. For finetuning we tried three different settings: i) retraining all parameters (MPNN and edge classifier), ii) only retraining edge classifier parameters and iii) only retraining the last layer of the edge classifier. We found that setting iii) worked best overall. As shown in Table A.5, we found models that improved on Knott and CREMI. Yet, finetuning did not help to improve performance on BSDS300.

A.6 Embedding Space Visualizations

Figure A.6 depicts results of our model (GCN_W_BN) on an IrisMP graph (#0).



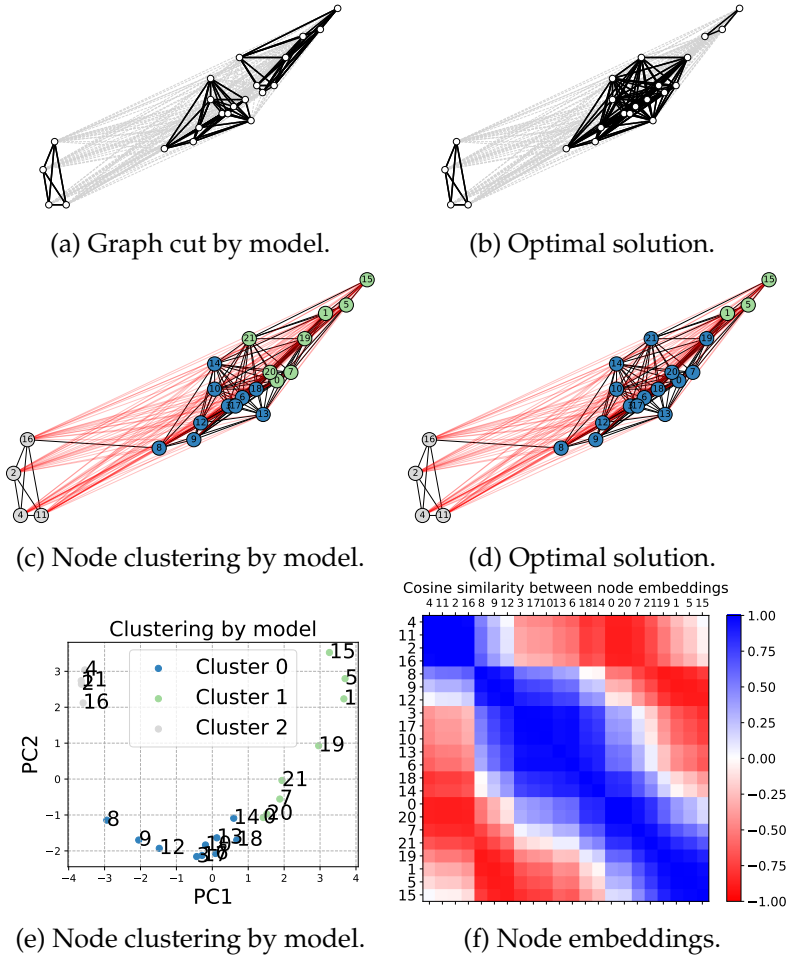
(a) Graph cut by model. (b) Optimal solution.

(c) Node clustering by model. (d) Optimal solution.

(e) Node clustering by model. (f) Node embeddings.

Figure A.6: **(a)** Graph cut solution computed by the model. **(b)** Graph cut of this graph according to the optimal solution. **(c)** Clustering of nodes according to the models' graph cut. **(d)** Clustering of nodes according to the optimal solution. **(e)** Node embeddings projected into a 2D space using PCA. Node colors according to the model prediction. **(f)** Cosine similarity between all node embeddings, ordered by similarity.

Figure A.7 depicts results of our model (GCN_W_BN) on an IrisMP graph (#1).



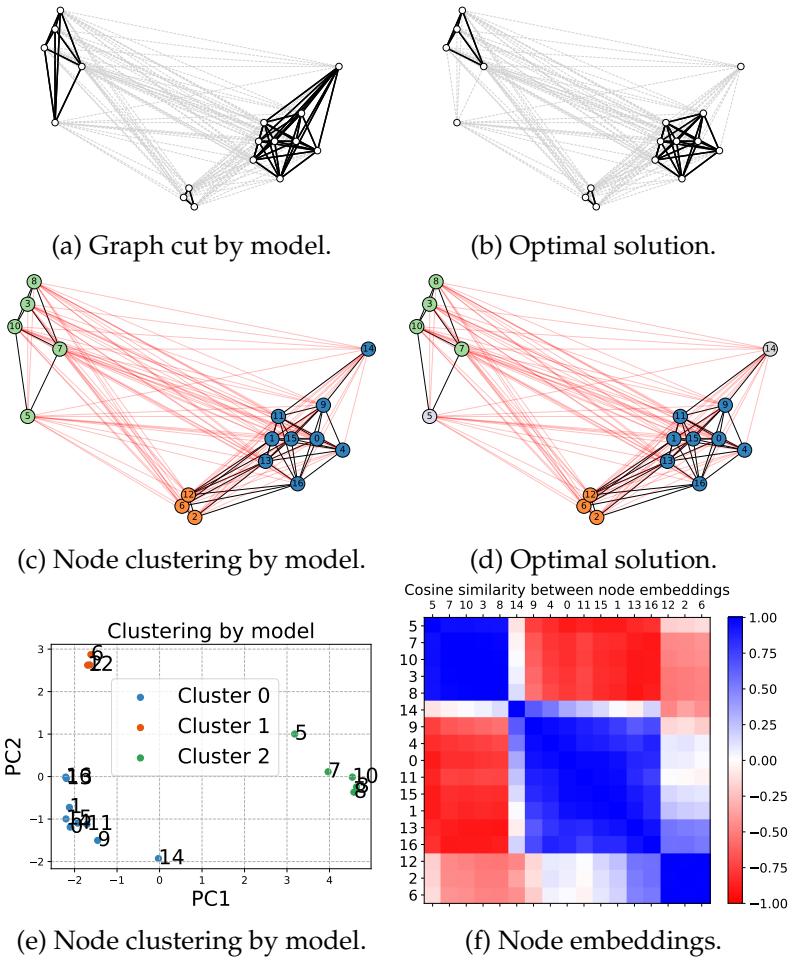
(a) Graph cut by model. (b) Optimal solution.

(c) Node clustering by model. (d) Optimal solution.

(e) Node clustering by model. (f) Node embeddings.

Figure A.7: **(a)** Graph cut solution computed by the model. **(b)** Graph cut of this graph according to the optimal solution. **(c)** Clustering of nodes according to the models' graph cut. **(d)** Clustering of nodes according to the optimal solution. **(e)** Node embeddings projected into a 2D space using PCA. Node colors according to the model prediction. **(f)** Cosine similarity between all node embeddings, ordered by similarity.

Figure A.8 depicts results of our model (GCN_W_BN) on an IrisMP graph (#16).



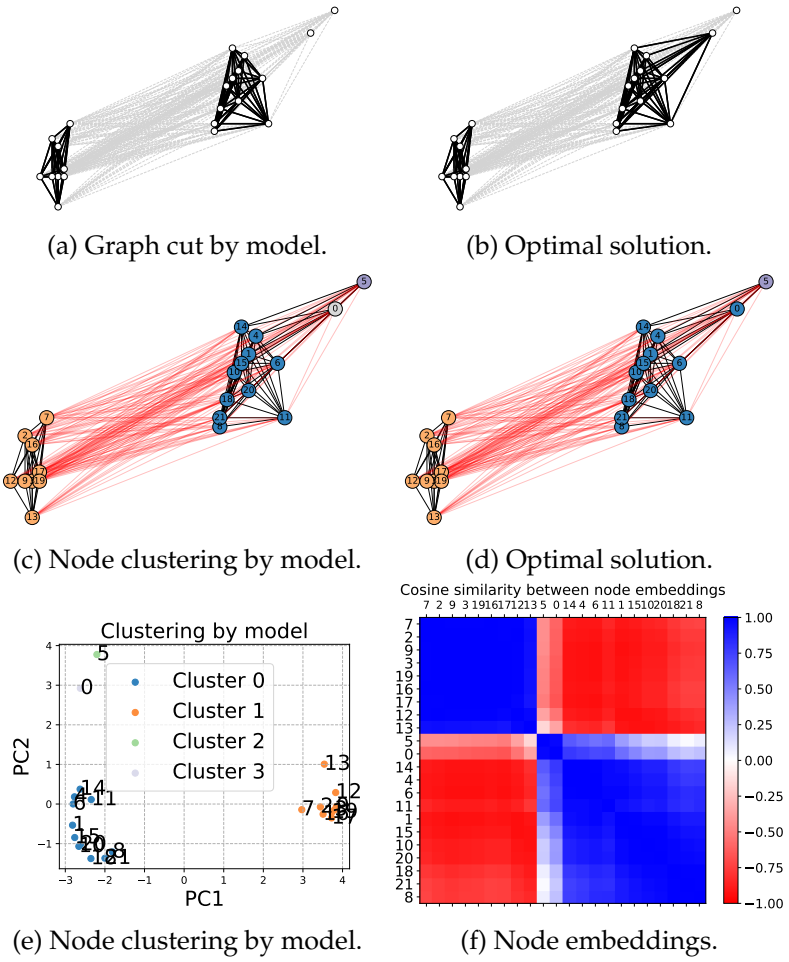
(a) Graph cut by model. (b) Optimal solution.

(c) Node clustering by model. (d) Optimal solution.

(e) Node clustering by model. (f) Node embeddings.

Figure A.8: **(a)** Graph cut solution computed by the model. **(b)** Graph cut of this graph according to the optimal solution. **(c)** Clustering of nodes according to the models' graph cut. **(d)** Clustering of nodes according to the optimal solution. **(e)** Node embeddings projected into a 2D space using PCA. Node colors according to the model prediction. **(f)** Cosine similarity between all node embeddings, ordered by similarity.

Figure A.9 depicts results of our model (GCN_W_BN) on an IrisMP graph (#17).



(a) Graph cut by model. (b) Optimal solution.

(c) Node clustering by model. (d) Optimal solution.

(e) Node clustering by model. (f) Node embeddings.

Figure A.9: **(a)** Graph cut solution computed by the model. **(b)** Graph cut of this graph according to the optimal solution. **(c)** Clustering of nodes according to the models' graph cut. **(d)** Clustering of nodes according to the optimal solution. **(e)** Node embeddings projected into a 2D space using PCA. Node colors according to the model prediction. **(f)** Cosine similarity between all node embeddings, ordered by similarity.

This page intentionally left blank.

Chapter B
Edge Detection with Discrete Constraints

In this supplementary material, we provide several additional details and visualizations:

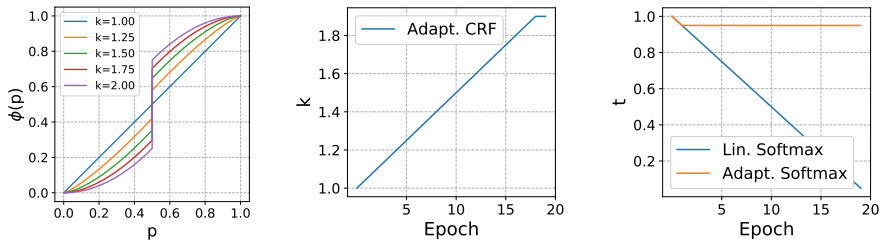
- Section B.1: Additional details to our training settings.
- Section B.2: Additional qualitative results on BSDS500.

B.1 Training Details

Adaptive CRF. By reformulating mean field updates from Equation 4.3 with our adaptive function ϕ (see Subsection 4.3.3), we get:

$$Q_i^t(y_i = l) = \frac{1}{Z_i} exp\Bigg\{ -\sum_{c \in C} \Bigg(\sum_{p \in P_{c|y_i=l}} \Big(\prod_{j \in c, j \neq i} \phi\left(Q_j^{t-1}(y_j = p_j), k\right) \Big) \gamma_p + \Big(1 - \Big(\sum_{p \in P_{c|x_i=l}} \Big(\prod_{j \in c, j \neq i} \phi\left(Q_j^{t-1}(y_j = p_j), k\right) \Big) \Big) \Big) \gamma_{max} \Bigg) \Bigg\}, \quad \text{(B.1)}$$

where function ϕ modifies probabilities such that their values are pushed closer to 0 or 1, depending on their current value. Figure B.1 (a) shows a plot of this function with different values of k. During training, k is initialized with 1.0 and then increased based on the cooling scheme defined in Equation 4.4. When training Adaptive CRF on BSDS500 for 20 epochs, we observe that k is increased each epoch with the exception of the last one (see Figure B.1 (b)).



(a) Cooling function (b) Cooling factor (c) Softmax temperature

Figure B.1: **(a)** Function ϕ plotted for different values of k. **(b)** Value of k during 20 epochs of training Adaptive CRF. **(c)** Values of t during 20 epochs of training Linear Softmax and Adaptive Softmax.

Linear Softmax and Adaptive Softmax. The purpose of ϕ is to push values towards 1 or 0, and therefore enforce multicut constraints more strictly. However, this should also be possible by considering temperature decay in the softmax function that is called after each mean field update iteration. Hence, instead of

$$Q_i^t(y_i = l) = \frac{\exp\{\tilde{Q}_i^t(y_i = l)\}}{\sum_l \exp\{\tilde{Q}_i^t(y_i = l)\}}$$

we compute

$$Q_i^t(y_i = l) = \frac{\exp\{\tilde{Q}_i^t(y_i = l)/t\}}{\sum_l \exp\{\tilde{Q}_i^t(y_i = l)/t\}},$$

where we initialize $t = 1.0$ and decay each epoch in the case of Linear Softmax by 0.05. In the case of Adaptive Softmax we use the same condition as for Adaptive CRF (see Equation 4.4), where t is only decayed by 0.05 if the average number of violated cycle inequalities is ≤ 500. Figure B.1 (c) shows the temperature decaying schemes during training of 20 epochs on BSDS500.

Training Runs. We train each model on one RTX8000 GPU with batch size 10, and optimize with SGD with learning rate 10^{-6}, momentum 0.9, weight decay $2 \cdot 10^{-4}$, stepsize 3, gamma 0.1 for 20 epochs.

B.2 Qualitative Results on BSDS500

We provide additional examples of BSDS500 images in Figure B.2-B.4, their respective edge maps as well as the corresponding UCM segmentation. We compare plain RCF results with models optimized with the basic CRF and the adaptive CRF which both encourage valid cycles that comply with the cycle inequality constraints of the Minimum Cost Multicut Problem. It can be seen that applying the CRFs during training results in cleaner edge maps where trailing edges are removed and contours are more likely to be closed. The effect is amplified for the adaptive CRF. The corresponding segmentations suffer less from oversegmented backgrounds and individual components are more precise compared to the plain RCF. The additional examples illustrate that promoting closed contours when learning edge maps yields more accurate segmentations.

(a) Example Image

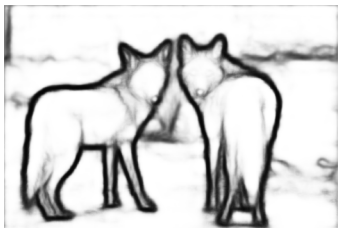
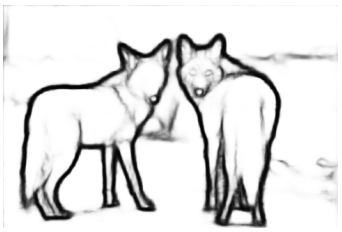
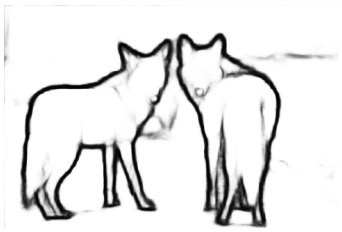
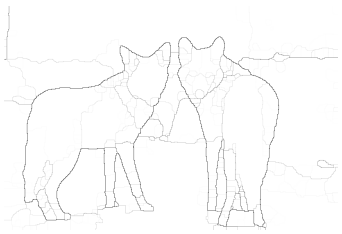
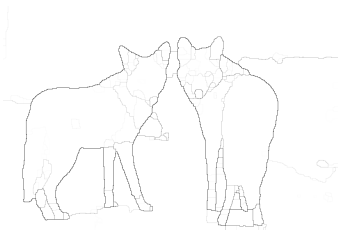
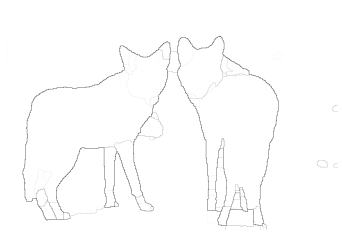
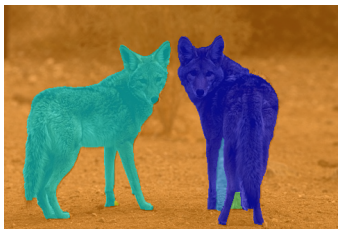
Baseline RCF	Baseline CRF	Adaptive CRF
		
(b) Edge maps		
		
(c) UCM Segmentations		
		
(d) UCM Segmentations (Threshold=0.5)		

Figure B.2: Example image 196062 from BSDS500, and resulting edge maps and segmentations for baseline models and the proposed adaptive CRF model.

(a) Example Image

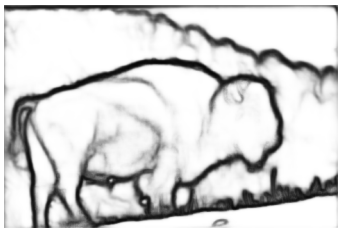
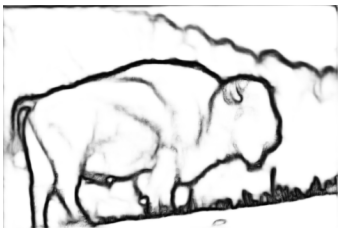
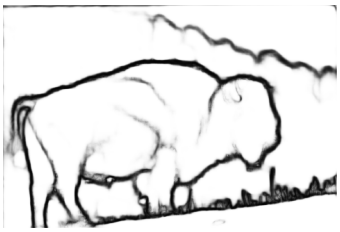
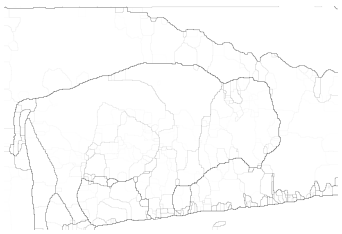
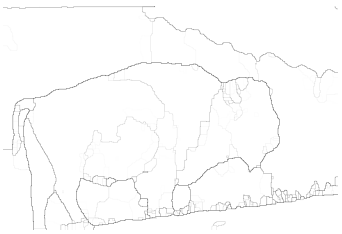
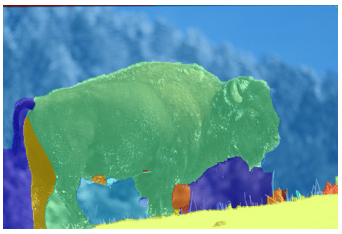
Baseline RCF	Baseline CRF	Adaptive CRF
		
(b) Edge maps		
		
(c) UCM Segmentations		
		
(d) UCM Segmentations (Threshold=0.5)		

Figure B.3: Example image 41029 from BSDS500, and resulting edge maps and segmentations for baseline models and the proposed adaptive CRF model.

(a) Example Image

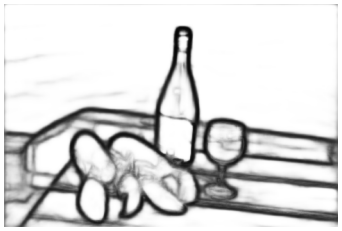
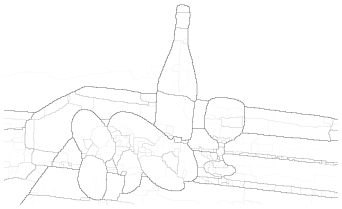
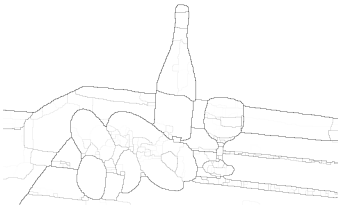
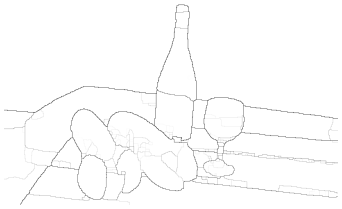
Baseline RCF	Baseline CRF	Adaptive CRF
		
(b) Edge maps		
		
(c) UCM Segmentations		
		
(d) UCM Segmentations (Threshold=0.5)		

Figure B.4: Example image 157032 from BSDS500, and resulting edge maps and segmentations for baseline models and the proposed adaptive CRF model.

Chapter C

Learned Representations to Penalize Image Synthesis

In this supplementary material, we provide several additional details, ablations and visualizations. We provide:

- Section C.1: Implementation details to minimizing FID.
- Section C.2: Visualizations of generated images by DCGAN under different training settings, trained on FFHQ64.
- Section C.3: Visualizations of generated images by SNGAN under different training settings, trained on FFHQ64.
- Section C.4: Visualizations of generated images by DCGAN under different training settings, trained on CIFAR10.
- Section C.5: Visualizations of generated images by SNGAN under different training settings, trained on CIFAR10.
- Section C.6: Overview of FID comparing different corruptions based on ImageNet-C, for all severities, when substituting different backbone networks.
- Section C.7: Implementation details on deep fake detection with FID.

C.1 Implementation Details to Minimizing FID

We train each of the models for 48 hours and report the best FID measured during training. FID is measured after each epoch. We train DCGAN with the minimax loss

$$\mathcal{L}_{\mathrm{G}} = \mathbb{E}_{z\sim N(0,1)}[\log(1 - D(G(z)))], \tag{C.1}$$

$$\mathcal{L}_{\mathrm{D}} = -\mathbb{E}_{x\sim \mathrm{data}}[\log(D(x))] \; - \; \mathbb{E}_{z\sim N(0,1)}[\log(1 - D(\hat{\mathbf{x}})], \tag{C.2}$$

and for optimization we use Adam with $\beta = (0.5, 0.999)$, $\epsilon = 10^{-8}$, learning rate $= 0.0002$, and weight decay $= 0$.

We train SNGAN with the hinge loss

$$\mathcal{L}_{\mathrm{G}} = -\mathbb{E}_{z\sim N(0,1)}[D(G(z))], \tag{C.3}$$

$$\mathcal{L}_{\mathrm{D}} = -\mathbb{E}_{x\sim \mathrm{data}}[\min(0, -1 + D(x))] - \mathbb{E}_{z\sim N(0,1)}[\min(0, -1 - D(G(z)))]. \tag{C.4}$$

and for optimization we use Adam with $\beta = (0.0, 0.9)$, $\epsilon = 10^{-8}$, learning rate $= 0.0002$, and weight decay $= 0$. We combine all results in the following table.

C.2 Generated Images (DCGAN/FFHQ)

Figure C.1: $\text{DCGAN}_{\text{G+D}}$ trained on FFHQ64. FID: 14.86.

Figure C.2: $\text{DCGAN}^{\text{FID}}_{\text{G+D}}$ trained on FFHQ64. FID: 5.38.

Figure C.3: $\text{DCGAN}_{\text{G}}^{\text{FID}}$ trained on FFHQ64. FID: 7.89.

Figure C.4: DCGAN-Up$_{\text{G}}^{\text{FID}}$ (upsampling instead of transpose convolutions) trained on FFHQ64. FID: 7.61.

C.3 Generated Images (SNGAN/FFHQ)

Figure C.5: $\text{SNGAN}_{\text{G+D}}$ trained on FFHQ64. FID: 9.44.

Figure C.6: $\text{SNGAN}^{\text{FID}}_{\text{G+D}}$ trained on FFHQ64. FID: 6.00.

Figure C.7: $\text{SNGAN}_{\text{G}}^{\text{FID}}$ trained on FFHQ64. FID: 7.74.

C.4 Generated Images (DCGAN/CIFAR10)

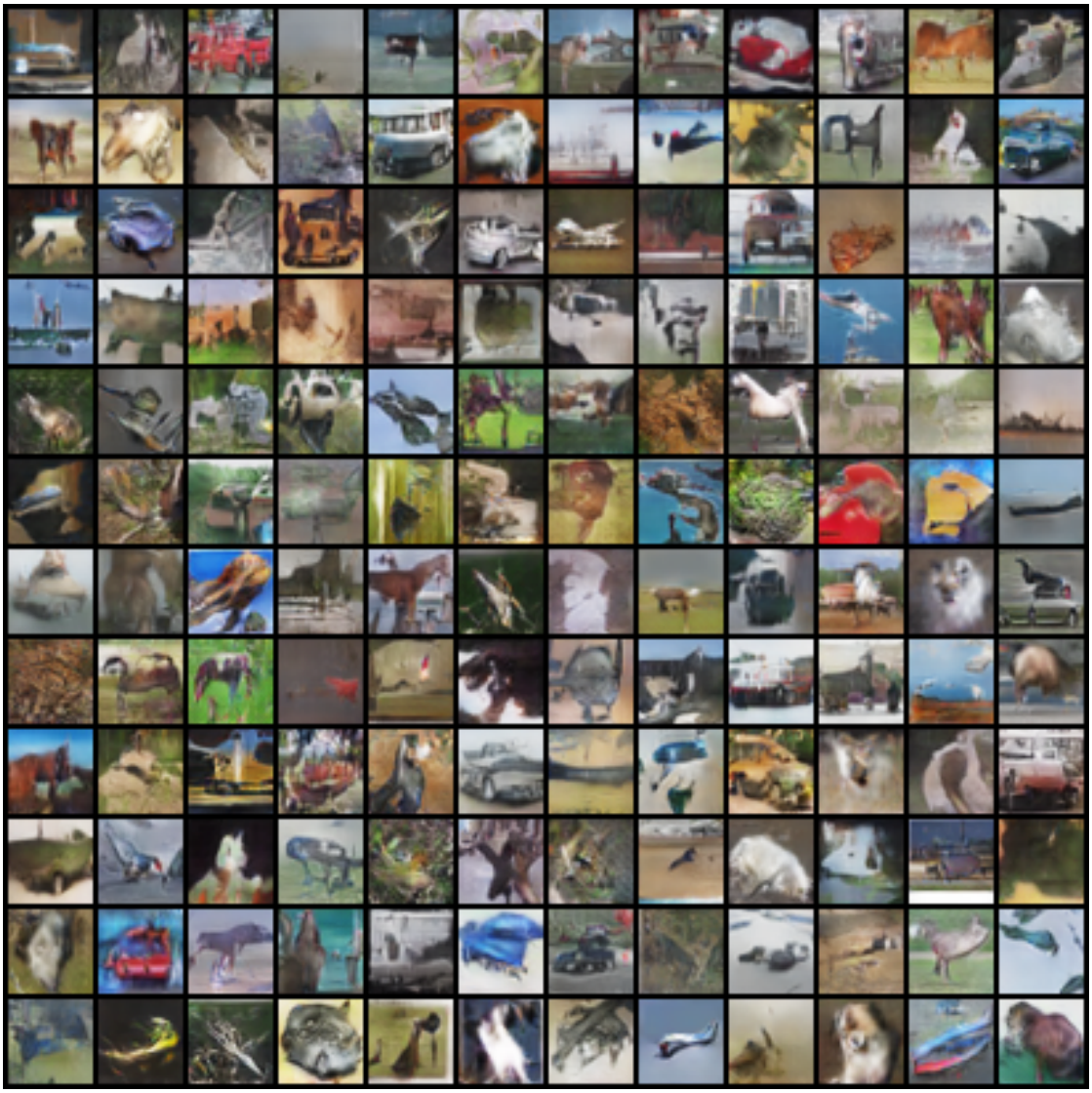


Figure C.8: $\text{DCGAN}_{\text{G+D}}$ trained on CIFAR10. FID: 28.72.

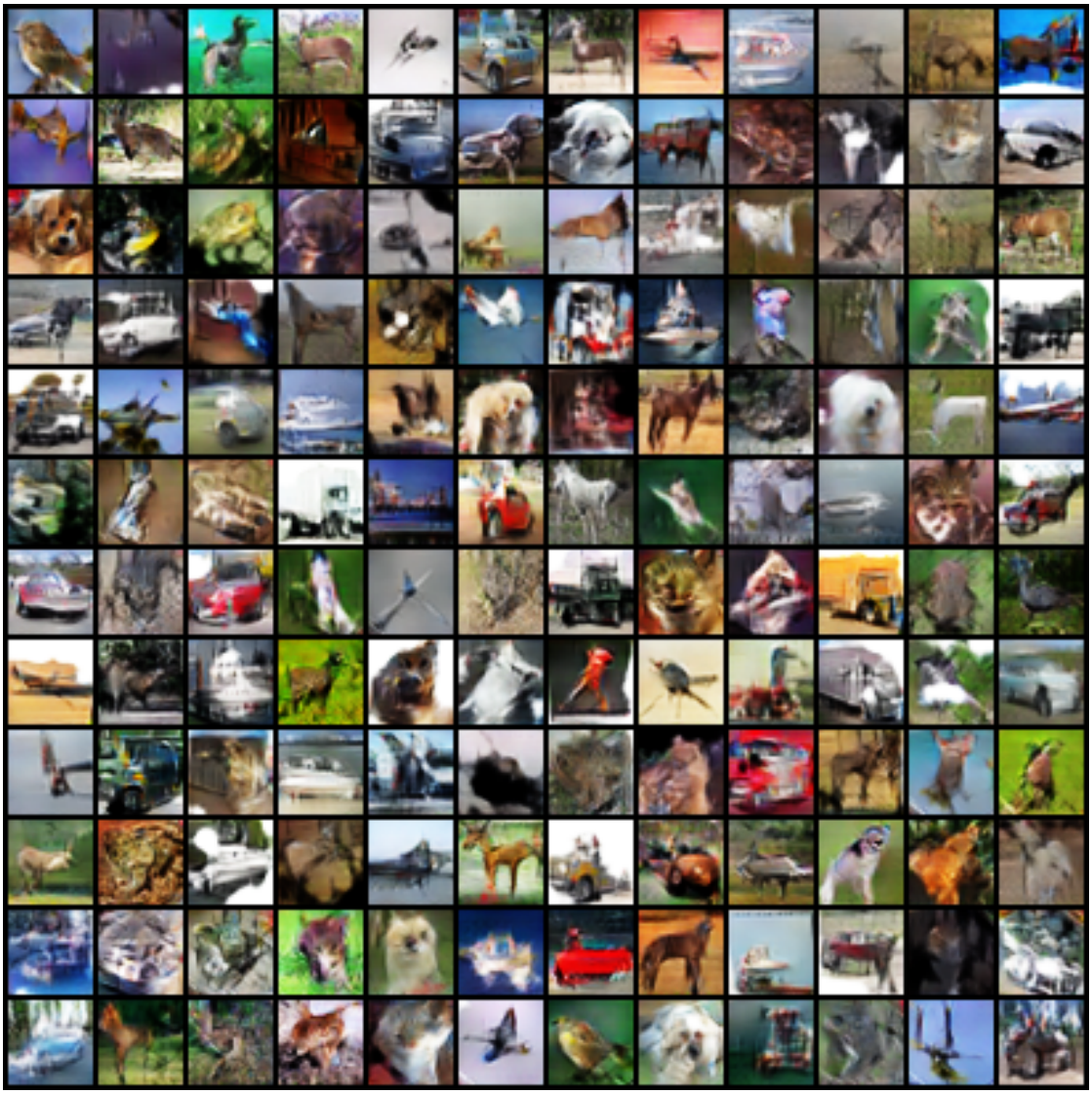


Figure C.9: $\text{DCGAN}^{\text{FID}}_{\text{G+D}}$ trained on CIFAR10. FID: 8.27.

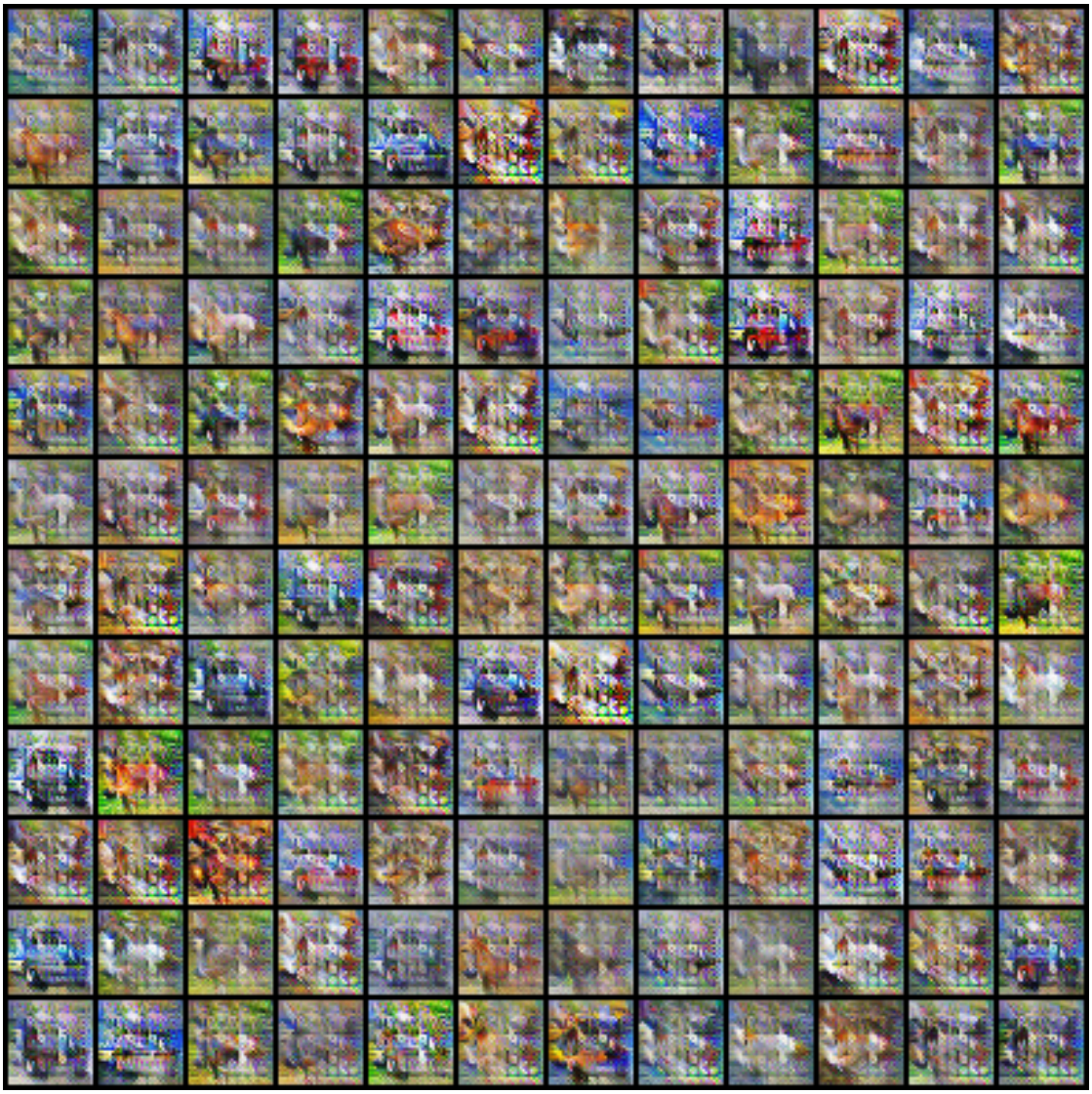


Figure C.10: $\text{DCGAN}_{\text{G}}^{\text{FID}}$ trained on CIFAR10. FID: 10.82.

Figure C.11: DCGAN-Up$_{G}^{FID}$ (upsampling instead of transpose convolutions) trained on CIFAR10. FID: 10.77.

C.5 Generated Images (SNGAN/CIFAR10)

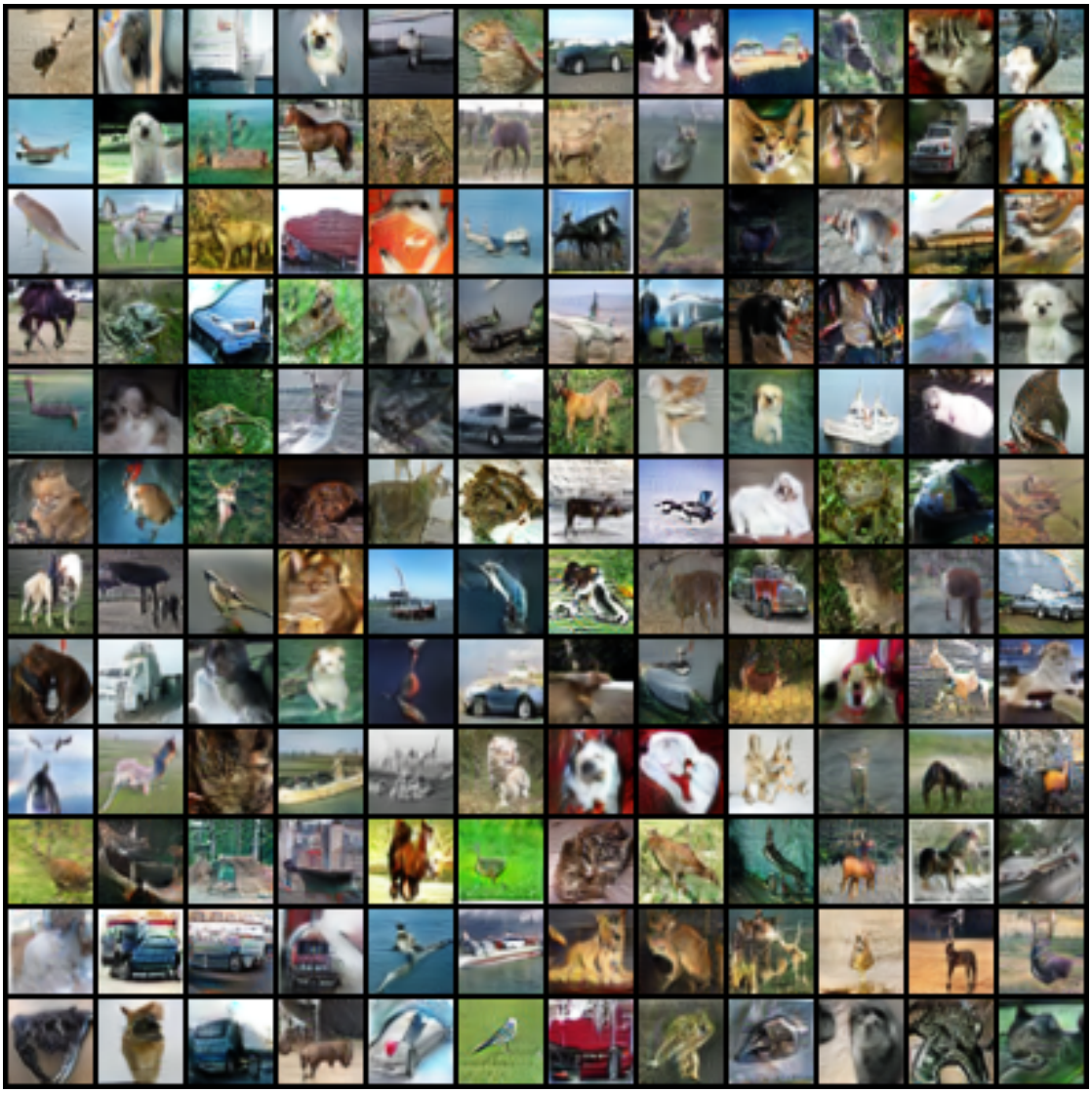


Figure C.12: $\text{SNGAN}_{\text{G+D}}$ trained on CIFAR10. FID: 17.85.

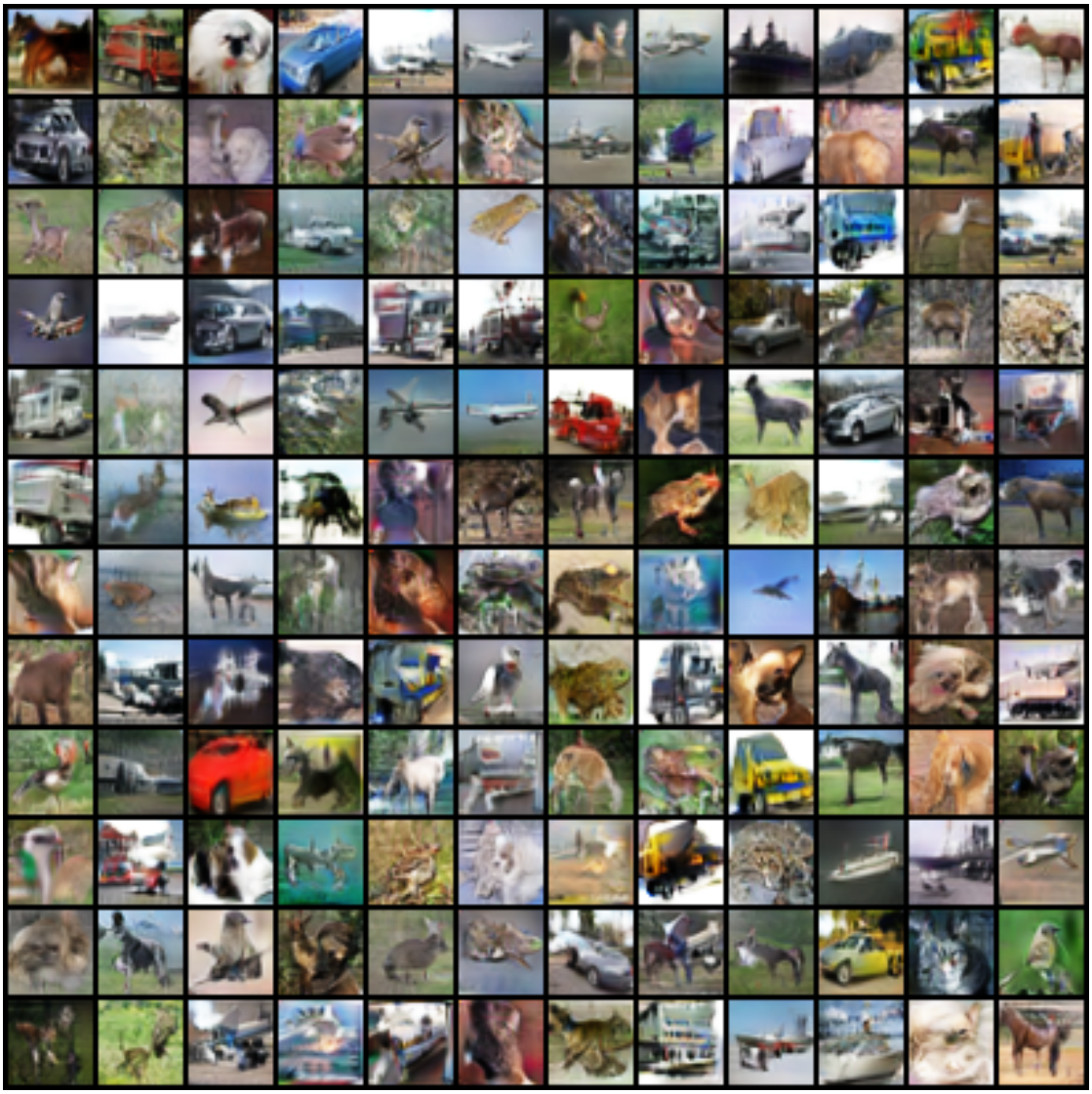


Figure C.13: $\text{SNGAN}^{\text{FID}}_{\text{G+D}}$ trained on CIFAR10. FID: 8.07.

Figure C.14: SNGAN$^{\text{FID}}_{\text{G}}$ trained on CIFAR10. FID: 11.66.

C.6 FIDs when substituting backbone networks on ImageNet-C

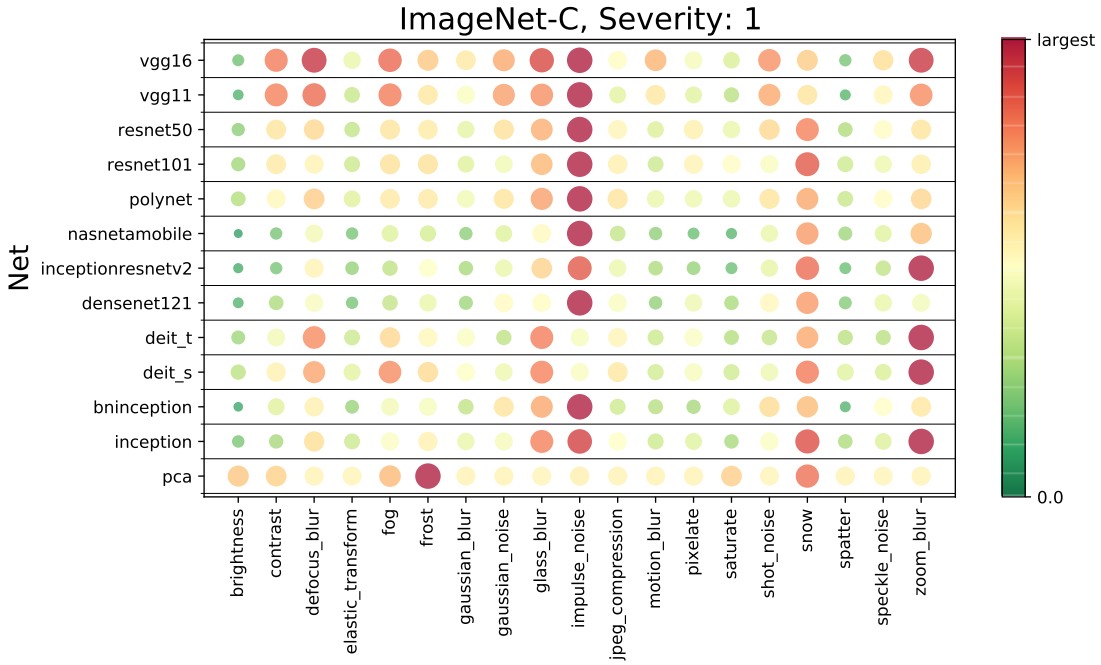


Figure C.15: Color- and size-coded FID between ImageNet validation images and 19 corrupted versions thereof provided by ImageNet-C [HD19]. Inception v3 is substituted by different classification networks [SZ15; He+16; Zha+17c; Zop+18; Sze+17; Hua+17; IS15; Sze+16] to investigate whether the ranking is affected by the feature extractor. All corruptions are at severity 1. Colors and circle sizes depend on the largest observed FID per network. Additionally, PCA features are shown, which provide descriptive features with different sensitivity to corruptions compared to image classification networks. We can see that rankings are inconsistent in-between different feature extractors.

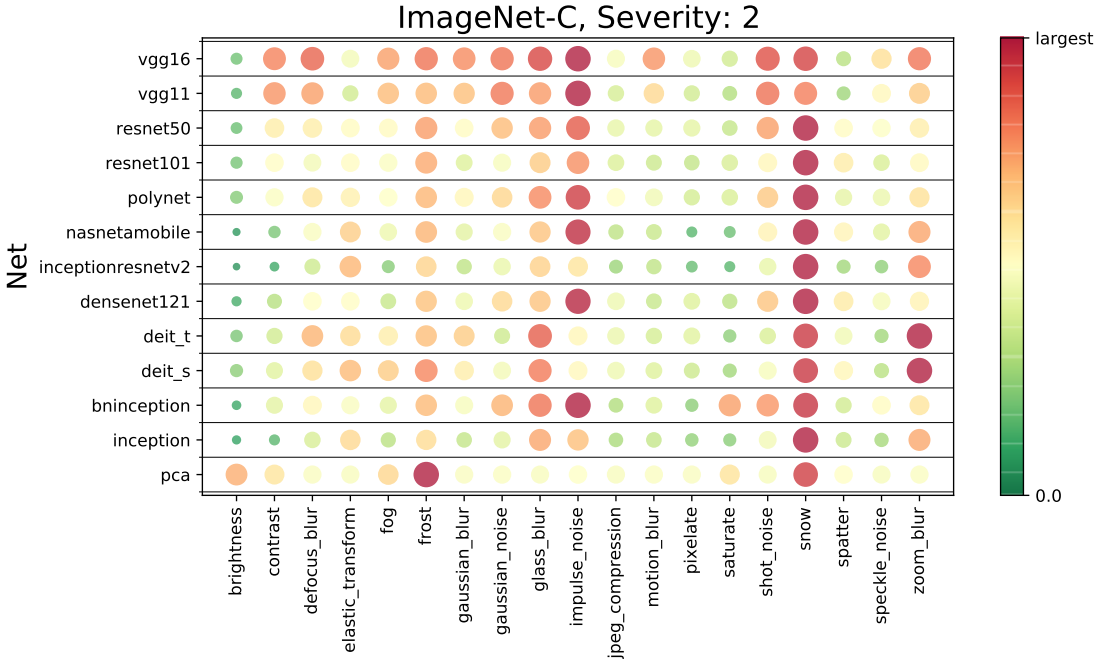


Figure C.16: Color- and size-coded FID between ImageNet validation images and 19 corrupted versions thereof provided by ImageNet-C [HD19]. Inception v3 is substituted by different classification networks [SZ15; He+16; Zha+17c; Zop+18; Sze+17; Hua+17; IS15; Sze+16] to investigate whether the ranking is affected by the feature extractor. All corruptions are at severity 2. Colors and circle sizes depend on the largest observed FID per network. Additionally, PCA features are shown, which provide descriptive features with different sensitivity to corruptions compared to image classification networks. We can see that rankings are inconsistent in-between different feature extractors.

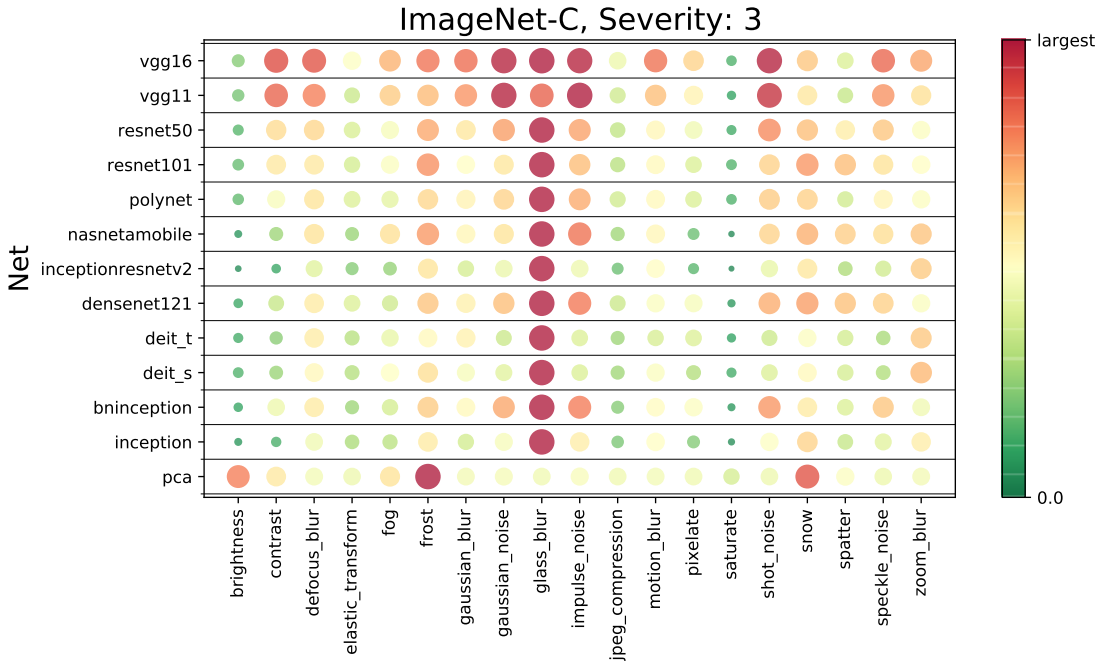


Figure C.17: Color- and size-coded FID between ImageNet validation images and 19 corrupted versions thereof provided by ImageNet-C [HD19]. Inception v3 is substituted by different classification networks [SZ15; He+16; Zha+17c; Zop+18; Sze+17; Hua+17; IS15; Sze+16] to investigate whether the ranking is affected by the feature extractor. All corruptions are at severity 3. Colors and circle sizes depend on the largest observed FID per network. Additionally, PCA features are shown, which provide descriptive features with different sensitivity to corruptions compared to image classification networks. We can see that rankings are inconsistent in-between different feature extractors.

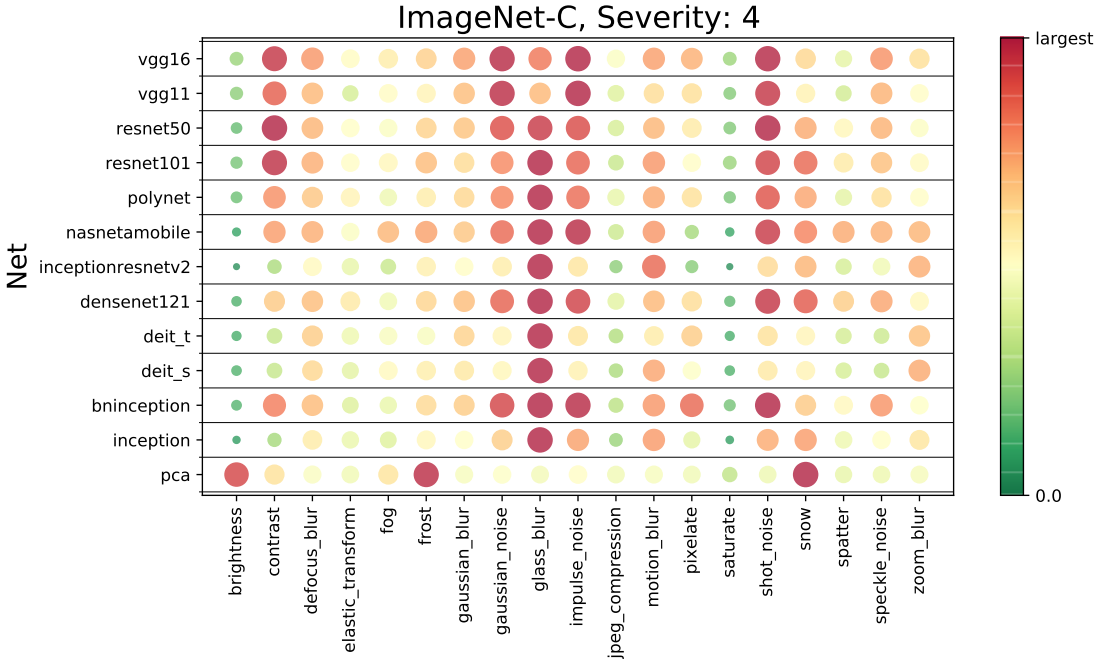


Figure C.18: Color- and size-coded FID between ImageNet validation images and 19 corrupted versions thereof provided by ImageNet-C [HD19]. Inception v3 is substituted by different classification networks [SZ15; He+16; Zha+17c; Zop+18; Sze+17; Hua+17; IS15; Sze+16] to investigate whether the ranking is affected by the feature extractor. All corruptions are at severity 4. Colors and circle sizes depend on the largest observed FID per network. Additionally, PCA features are shown, which provide descriptive features with different sensitivity to corruptions compared to image classification networks. We can see that rankings are inconsistent in-between different feature extractors.

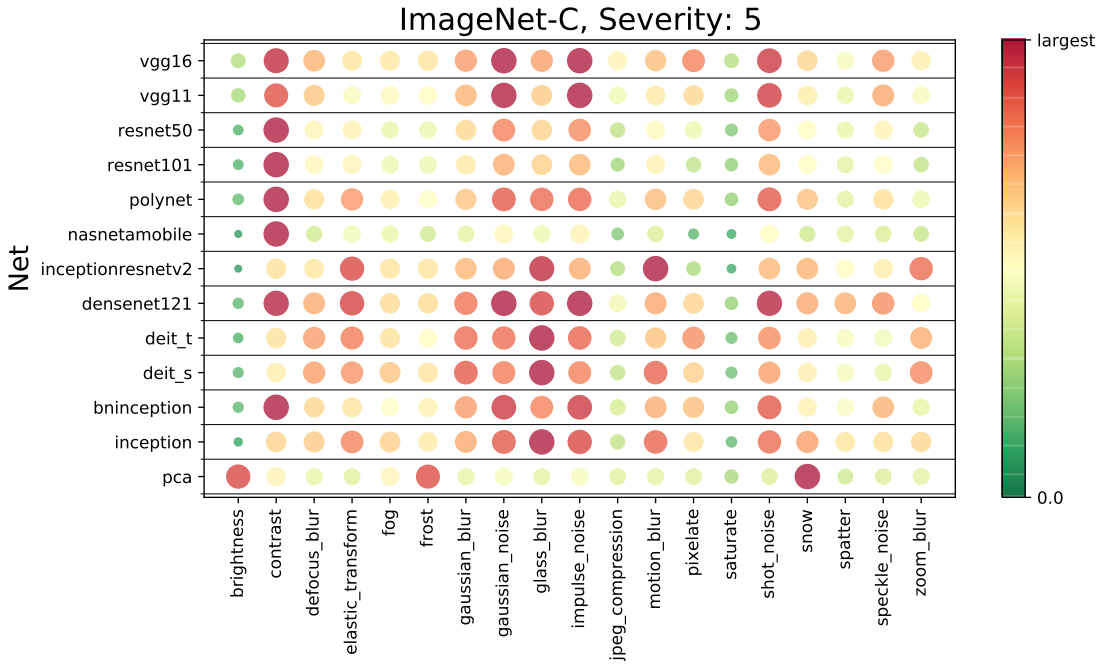


Figure C.19: Color- and size-coded FID between ImageNet validation images and 19 corrupted versions thereof provided by ImageNet-C [HD19]. Inception v3 is substituted by different classification networks [SZ15; He+16; Zha+17c; Zop+18; Sze+17; Hua+17; IS15; Sze+16] to investigate whether the ranking is affected by the feature extractor. All corruptions are at severity 5. Colors and circle sizes depend on the largest observed FID per network. Additionally, PCA features are shown, which provide descriptive features with different sensitivity to corruptions compared to image classification networks. We can see that rankings are inconsistent in-between different feature extractors.

C.7 Deep Fake Detection with FID

We map 70 000 images from FFHQ and 70 000 images generated by StyleGAN2 (with and without truncation) into the Inception v3 feature space by using the FID PyTorch implementation. We split each into 60 000 training and 10 000 testing images, and hence, end up with balanced training datasets containing 120 000 images and balanced test datasets containing 20 000 images. We train sklean logistic regression models and report the accuracy on the test dataset. A selection of corresponding images is shown in Figure 5.8. We argue that a meaningful metric should be visually aligned with human perception. Hence, if a human can be fooled by a generator network, then this generator should be considered superior to one that is not able to do so. We see that truncation decreases FID substantially, and consequently improves the ability of detecting StyleGAN2 generated images as fake. In contrast, we show images produced by StylegAN2 without and with truncation in Figure 5.8. By inspecting the images we observe that truncation removes textures (and also artifacts). We hypothesize that its bias towards textures facilitates Inception v3 to extract features that allow almost perfect detectability (98% when truncation is applied). However, we expect humans to be more easily fooled by truncated images than untruncated ones. Hence, we argue that this is a hint towards that FID is not aligned with human perception.

Chapter D

Spectral Distribution-Aware Image Synthesis

In this supplementary material, we provide several additional details and visualizations:

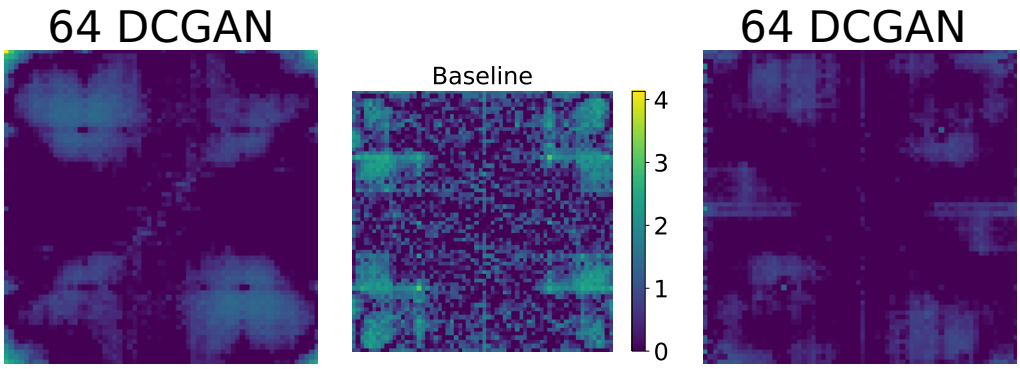
- Section D.1: An example showing high frequency artifacts generated by a model trained without spectral regularization.
- Section D.2: Further visualizations of differences in power spectra.
- Section D.3: Details about training the logistic regression involved in the proposed cloaking score.
- Section D.4: Samples of generated images by different models trained with our approach.

D.1 High Frequency Artifacts

In the proposed paper, we show that frequency artifacts in generated images can be substantially removed be using a simple spectral discriminator. Here, we want to emphasize again why a peak in the high frequency regime of the generated images' power spectra are undesired. We provide an example in Figure D.1, showing a face in front of what appears to be a mostly homogeneous background. Yet after applying a sharpening operation (Figure D.1 (b)), grid artifacts become obvious even in these supposedly flat regions. At a close look, they can also be seen in the plain images without sharpening (Figure D.1 (c)).

(a) DCGAN (b) sharpened (c) zoomed in

Figure D.1: Sample from DCGAN, 128^2. Peaks in the high frequencies which we measure in the power spectrum correspond to grid artifacts in the images. After applying an image sharpening operation on a generated image **(a)**, they become obvious **(b)** but at a close look, they can also be perceived in the original generated images **(c)**.



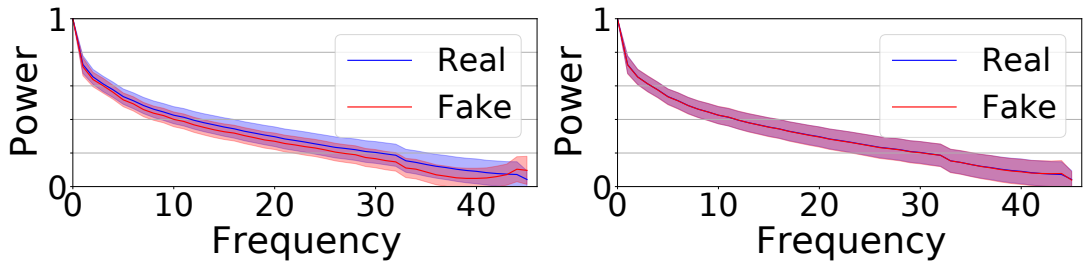
(a) DCGAN 64×64 (b) Durall et al. (c) DCGAN Spectral

Figure D.2: Average magnitude differences of the 2D FFT between real and generated images. We compare **(a)** DCGAN without additional loss or regularization, **(b)** DCGAN with the regularization proposed in [DKK20] and **(c)** the proposed model with spectral discriminator.

D.2 Evaluation of Generated Power Spectra

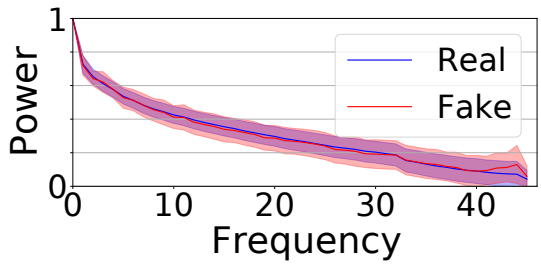
2D Power Spectra. Figure D.2 shows the mean absolute differences of the 2D power spectra of real and generated images. Here, we compare the spectra resulting from DCGAN (Figure D.2 (a)), from DCGAN with the spectral regularization from Durall, Keuper, and Keuper [DKK20] (Figure D.2 (b)) and our proposed model (Figure D.2 (c)). With the proposed model, the mean differences are substantially smaller even in the 2D power spectral, although the discriminator only operates on 1D projections of the spectra. The 2D spectra of the reference method [DKK20] still exhibit strong deviations.

1D Power Spectra. In Figures D.3, D.4, and D.5 we report the mean and standard deviation of the spectral profiles (azimuthal integrals) of the proposed model for higher resolution images. As shown in the main paper for images of resolution 64×64, the distributions fit almost perfectly when our proposed discriminator is use. The 1D projections of the power spectra are very similarly distributed when our model is used while they show obvious differences otherwise.

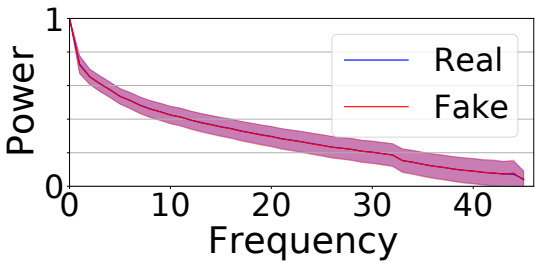


64 × 64 DCGAN

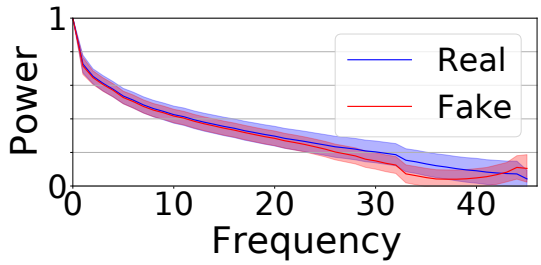
DCGAN Spectral



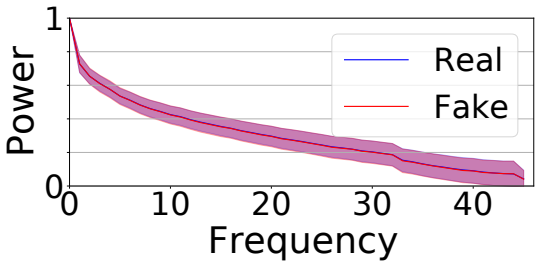
64 × 64 LSGAN



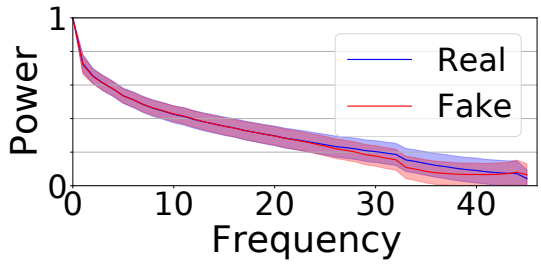
LSGAN Spectral



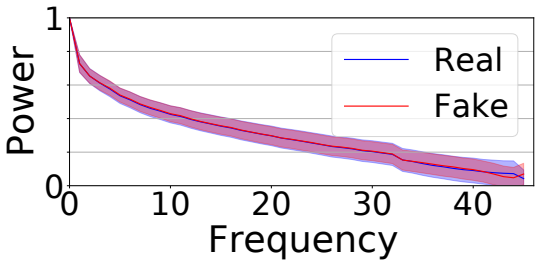
64 × 64 WGAN



WGAN Spectral

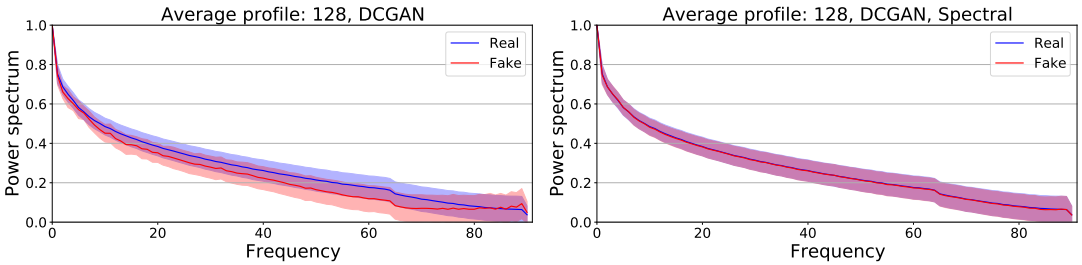


64 × 64 WGAN-GP

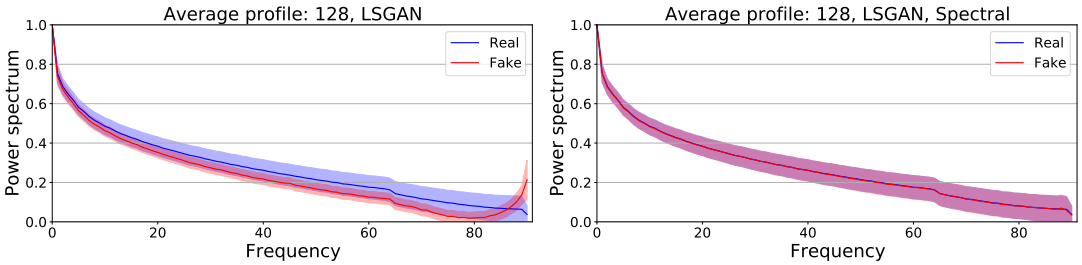


WGAN-GP Spectral

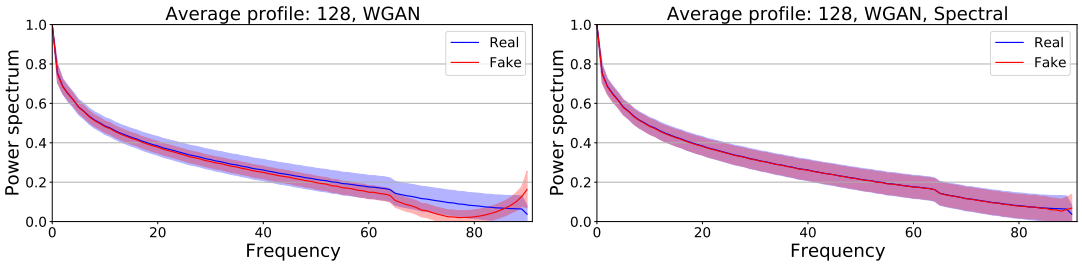
Figure D.3: Experiments with our models trained on FFHQ64. *Spectral* indicates that D_F was applied. Without the spectral discriminator, the spectral profiles of real and generated images are substantially different in their distribution. With the spectral discriminator, the mean and standard deviation of the spectral profiles fit almost perfectly.



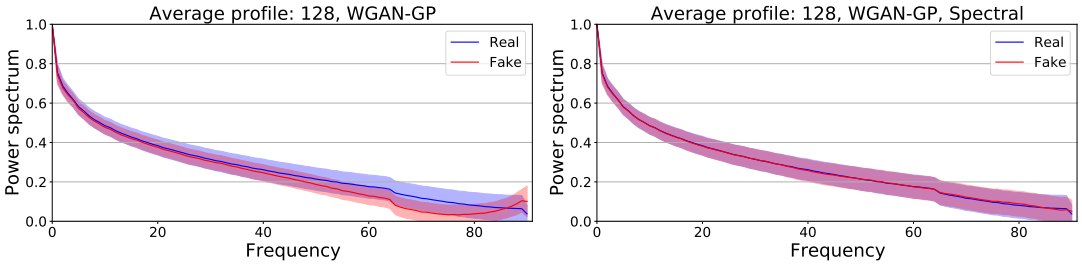
128 × 128 DCGAN DCGAN Spectral



128 × 128 LSGAN LSGAN Spectral

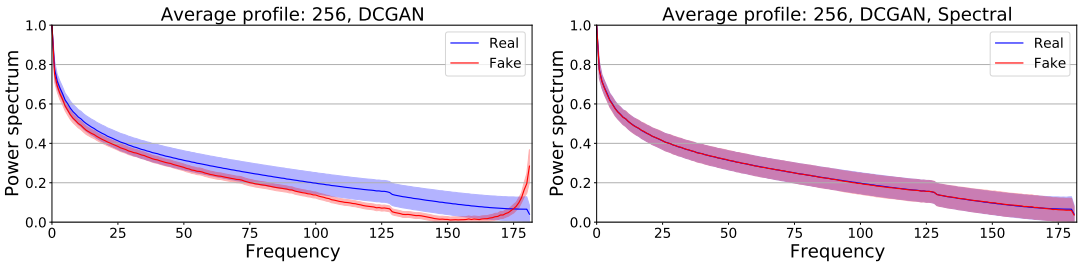


128 × 128 WGAN WGAN Spectral



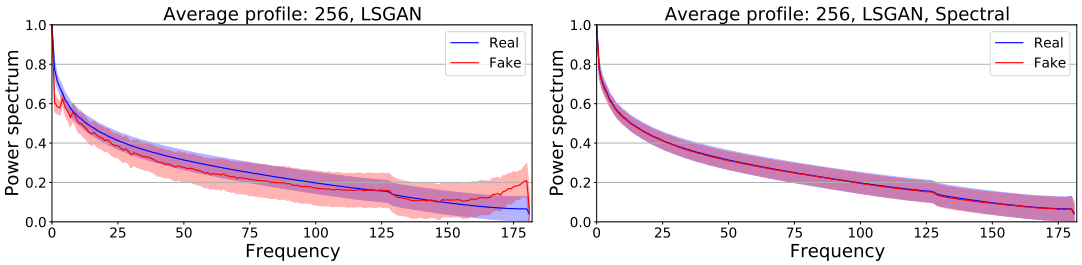
128 × 128 WGAN-GP WGAN-GP Spectral

Figure D.4: Experiments with our models trained on FFHQ128. *Spectral* indicates that D_F was applied. Without the spectral discriminator, the spectral profiles of real and generated images are substantially different in their distribution. With the spectral discriminator, the mean and standard deviation of the spectral profiles fit almost perfectly.



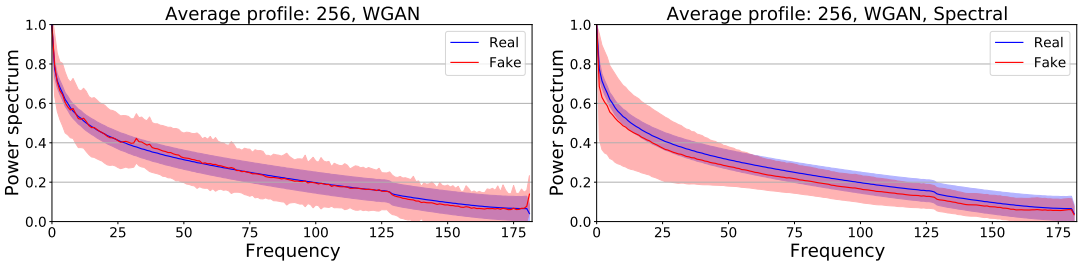
256 × 256 DCGAN

DCGAN Spectral



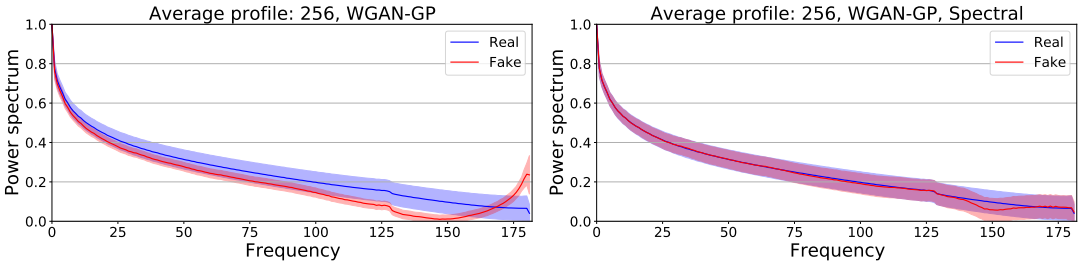
256 × 256 LSGAN

LSGAN Spectral



256 × 256 WGAN

WGAN Spectral



256 × 256 WGAN-GP

WGAN-GP Spectral

Figure D.5: Experiments with our models trained on FFHQ256. *Spectral* indicates that D_F was applied. Without the spectral discriminator, the spectral profiles of real and generated images are substantially different in their distribution. With the spectral discriminator, the mean and standard deviation of the spectral profiles fit almost perfectly.

D.3 Training Details for Cloaking Score

Figure D.6 shows the training progress of our regression model used in the cloaking score computation in the section *GAN Evaluation in the Frequency Domain* of the main paper. Training and testing accuracy are almost identical. Training accuracy increases progressively to 0.949 after 100 epochs, 0.979 after 1 000 epochs, 0.989 after 10 000 epochs, and 0.991 after 40 000 epochs. We stopped training at this point.

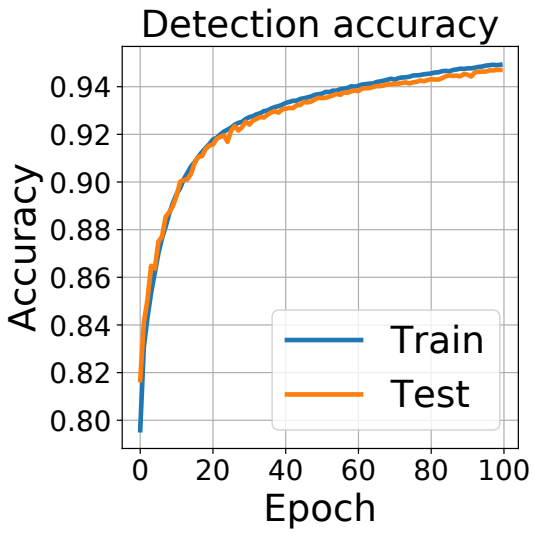


Figure D.6: Training and test accuracy of a Logistic Regression (LR) trained on 120k training images, 60k taken from FFHQ and 60k generated by StyleGAN2. 20k images are used for testing. The real and generated images can be distinguished to a large degree using LR.

D.4 Sample images generated from the Proposed Model and the Baselines

In Figures D.7 to D.14, we show examples of images generated with DCGAN and LSGAN without and with the proposed spectral discriminator for different resolutions. In all cases, the generated images are visually appealing and diverse.

Figure D.7: Generated images by DCGAN (64^2).

Figure D.8: Generated images by spectral DCGAN (64^2).

Figure D.9: Generated images by LSGAN (64^2).

Figure D.10: Generated images by spectral LSGAN (64^2).

Figure D.11: Generated images by DCGAN (128^2).

Figure D.12: Generated images by spectral DCGAN (128^2).

Figure D.13: Generated images by LSGAN (128^2).

Figure D.14: Generated images by spectral LSGAN (128^2).

Chapter E

Biasing Discrete Representations for Image Synthesis

In this supplementary material, we provide several additional details and visualizations. We provide:

- Section E.1: Details of the dataset used in our experiments.
- Section E.2: Encoder and decoder architecture as well as hyperparameters for experiments with Vector Quantized Variational Autoencoders (VQ-VAEs).
- Section E.3: Encoder and decoder architecture as well as hyperparameters for experiments with Variational Autoencoders (VAEs).

E.1 Details on Face Image Dataset

We focus on the facial part of CelebA images by using pretrained Multitask Cascaded Convolutional Networks (MTCNN) [Zha+16] to localize faces in the input plane, and cropping images according to the predicted bounding box. Furthermore, the data is resized to 64×64 via resampling using pixel area relation.

E.2 Details on VQ-VAE

Table E.1: VQ-VAE encoder architecture. ResBlock = ResNet Block, LReLU = Leaky ReLU.

	Layer	**Shape In** (H_0, W_0, D_0)	**Shape Out** (H_1, W_1, D_1)	**Filters** K	**Kernel** F	**Padding** P	**Stride** S
1	Conv LReLU	$(64, 64, 3)$	$(32, 32, 64)$	64	4	1	2
2	Conv LReLU	$(32, 32, 64)$	$(16, 16, 128)$	128	4	1	2
3	Conv LReLU	$(16, 16, 128)$	$(8, 8, 256)$	256	4	1	2
4	Conv LReLU	$(8, 8, 256)$	$(8, 8, 256)$	256	3	1	1
5	ResBlock	$(8, 8, 256)$	$(8, 8, 256)$				
6	ResBlock	$(8, 8, 256)$	$(8, 8, 256)$				
7	Conv LReLU	$(8, 8, 256)$	$(8, 8, 64)$	64	1	0	1

Table E.2: VQ-VAE decoder architecture. ResBlock = ResNet Block, LReLU = Leaky ReLU.

	Layer	**Shape In** (H_0, W_0, D_0)	**Shape Out** (H_1, W_1, D_1)	**Filters** K	**Kernel** F	**Padding** P	**Stride** S
1	Conv LReLU	$(8, 8, 64)$	$(8, 8, 256)$	256	3	1	1
2	ResBlock	$(8, 8, 256)$	$(8, 8, 256)$				
3	ResBlock	$(8, 8, 256)$	$(8, 8, 256)$				
4	ConvT LReLU	$(8, 8, 256)$	$(16, 16, 128)$	128	4	1	2
5	ConvT LReLU	$(16, 16, 128)$	$(32, 32, 64)$	64	4	1	2
6	ConvT	$(32, 32, 64)$	$(64, 64, 3)$	3	4	1	2

Table E.3: Training hyperparameters of VQ-VAE experiments.

Hyperparameter	**Value**
Latent Dimension	64
Embedding Dimension D	64
# Categories K	256
Batch Size	128
# Epochs	70
Optimizer	Adam
Learning Rate	0.001
β	0.25

E.3 Details on VAE

Table E.4: Encoder architecture of VAE. ResBlock = ResNet Block, LReLU = Leaky ReLU.

	Layer	**Shape In** (H_0, W_0, D_0)	**Shape Out** (H_1, W_1, D_1)	**Filters** K	**Kernel** F	**Padding** P	**Stride** S
1	Conv LReLU BatchNorm	$(64, 64, 3)$	$(32, 32, 32)$	32	3	1	2
2	Conv LReLU BatchNorm	$(32, 32, 32)$	$(16, 16, 64)$	64	3	1	2
3	Conv LReLU BatchNorm	$(16, 16, 64)$	$(8, 8, 128)$	128	3	1	2
4	Conv LReLU BatchNorm	$(8, 8, 128)$	$(4, 4, 256)$	256	3	1	2
5	Conv LReLU BatchNorm	$(4, 4, 256)$	$(2, 2, 512)$	512	3	1	2
6	Flatten Dense	$(2, 2, 512)$ 2048	2048 64				

Table E.5: Decoder architecture of VAE. ResBlock = ResNet Block, LReLU = Leaky ReLU.

	Layer	**Shape In** (H_0, W_0, D_0)	**Shape Out** (H_1, W_1, D_1)	**Filters** K	**Kernel** F	**Padding** P	**Stride** S
1	Dense LReLU BatchNorm	64	2048				
2	Unflatten ConvT LReLU BatchNorm	2048 $(2, 2, 512)$	$(2, 2, 512)$ $(4, 4, 256)$	256	3	1	2
3	ConvT LReLU BatchNorm	$(4, 4, 256)$	$(8, 8, 128)$	128	3	1	2
4	ConvT LReLU BatchNorm	$(8, 8, 128)$	$(16, 16, 64)$	64	3	1	2
5	ConvT LReLU BatchNorm	$(16, 16, 64)$	$(32, 32, 32)$	32	3	1	2
6	ConvT LReLU BatchNorm	$(32, 32, 32)$	$(64, 64, 32)$	32	3	1	2
7	Conv	$(64, 64, 32)$	$(64, 64, 3)$	3	3	1	1

Table E.6: Training hyperparameters of VAE experiments.

Hyperparameter	**Value**
Latent Dimension	64
Batch Size	64
# Epochs	30
Optimizer	Adam
Learning Rate	0.0001
β_{start}	10^{-6}
β_{final}	0.215
β Warm-up	5000
β Step	1.1
β Step Frequency	50

Chapter F
Biasing Generative Neural Architecture Search

In this supplementary material, we provide several additional details and visualizations:

- Section F.1: An overview of the graph representations for each search space we consider in the main paper.
- Section F.2: Additional ablation studies.
- Section F.3: Implementation details about the experimental settings.
- Section F.4: Details about the generator network.
- Section F.5: Hyperparameter settings of our experiments.
- Section F.6: Visual intuition of the latent space optimization technique.

F.1 Search Space Representations

In this section we give more details about the search spaces we consider.

F.1.1 NAS-Bench-101

NAS-Bench-101 is the first tabular benchmark designed for benchmarking Neural Architecture Search (NAS) methods. This search space is a cell-based search space and contains 423 624 unique neural networks. Each architecture is trained 3 times on CIFAR-10 [Kri09] for image classification. The cell topology is limited to the number of nodes $|V| \leq 7$ (including input and output nodes) and edges $|E| \leq 9$. The nodes represent the architecture layers and intermediate nodes can take any operation from the operation set $\mathcal{O} = \{ 1 \times 1 \text{ conv.}, 3 \times 3 \text{ conv.}, 3 \times 3 \text{ max pooling} \}$. For visualization purposes, we present in Figure F.1 exemplary a Directed Acyclic Graph (DAG) from the NAS-Bench-101 search space with its corresponding node attribute matrix and its adjacency matrix. Note, a concatenation of the flattened node attribute matrix and the flattened upper triangular adjacency matrix is the representation our generator model is trained to learn; this holds for all search spaces.

We show experiments on NAS-Bench-101 in Subsection 8.4.1.

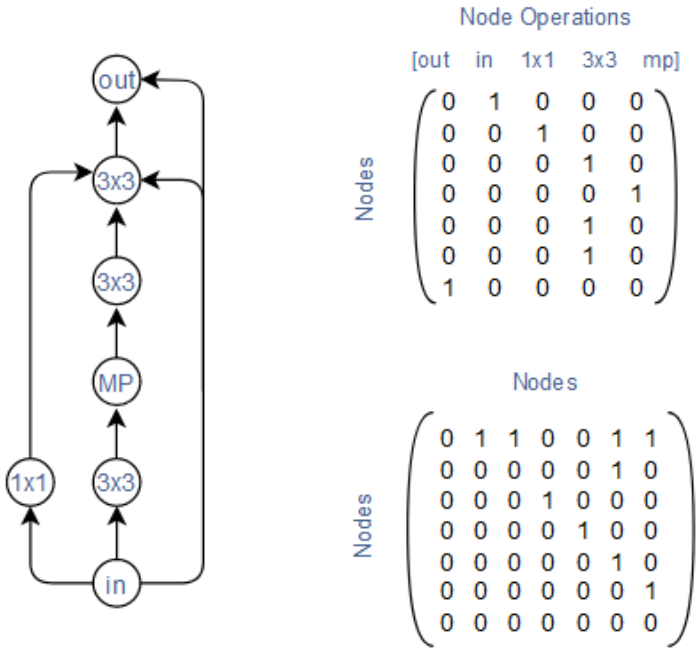


Figure F.1: Exemplary cell representation from the NAS-Bench-101 search space. (**left**) DAG representation of a graph with 7 nodes. (**right**) The top shows the node attribute matrix of the DAG and the bottom shows its adjacency matrix.

F.1.2 NAS-Bench-201

NAS-Bench-201 [DY20] is another cell-structured search space, which consists of 15 625 architectures. Each architecture is trained for 200 training epochs on CIFAR-10 [Kri09], CIFAR-100 [Kri09], and ImageNet16-120 [CLH17]. This benchmark provides validation and test accuracy information for each of the three datasets. The cell structure is different compared to NAS-Bench-101: Each cell has $|V| = 4$ nodes and $|E| = 6$ edges, where the former represent feature maps and the latter denote operations chosen from the set $\mathcal{O} = \{ 1 \times 1 \text{ conv.}, 3 \times 3 \text{ conv.}, 3 \times 3 \text{ avg. pooling}, \text{skip}, \text{zero} \}$. Figure F.2 visualizes a DAG of the NAS-Bench-201 search space in the provided variant with edge attributes, as well as our adapted representation where the edge attributes are changed to node attributes. This is similar to the representation in Yan et al. [Yan+20].

We show experiments on NAS-Bench-101 in Subsection 8.4.1.

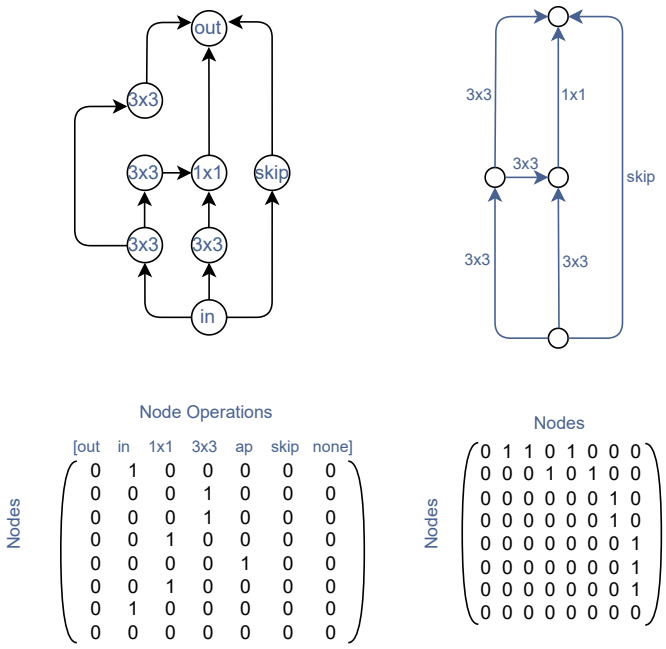


Figure F.2: Exemplary cell representation from the NAS-Bench-201 search space. (**top**) The left part visualizes the DAG representation with node attributes instead of edge attributes. The right part shows the true DAG representation in the NAS-Bench-201 search space. (**bottom**) The left part shows the node attribute matrix to the DAG and the right part shows its adjacency matrix.

F.1.3 DARTS Search Space

NAS-Bench-301 [Zel+22] is the first surrogate benchmark, which evaluates several surrogate models on in total 60 000 sampled architectures from the DARTS [LSY19] search space on the CIFAR-10 [Kri09] image classification task. The DARTS search space consists of 10^{18} neural networks, where each network consists of two cells; a normal cell and a reduction cell. Each cell is limited by the number of nodes $|N| = 7$ and the number of edges $|E| = 12$, where 4 of these edges connect the intermediate nodes (excluding the input nodes) to the output node. Each edge denotes an operation from the set $\mathcal{O} = \{$ 3×3 sep. conv., 5×5 sep. conv., 3×3 dil. conv., 5×5 dil. conv., 3×3 avg. pooling, 3×3 max pooling, identity, zero $\}$. Each intermediate edge is connected to two predecessor nodes. Each cell also contains two input nodes, which are the output nodes from the previous two cells. The overall network is created by stacking the normal and reduction cell.

In order to train our generative model to generate valid cells, we additionally randomly sample $500k$ architectures from the DARTS search space. We train our generative model to learn to generate valid cells independently of being a normal or reduction cell. In Figure F.3 we visualize the adapted node attribute matrix and the adapted adjacency matrix to an exemplary DAG in the DARTS search space [LSY19]. This is similar to the representation in Yan et al. [Yan+20].

We show experiments in the DARTS search space in Subsection 8.4.2.

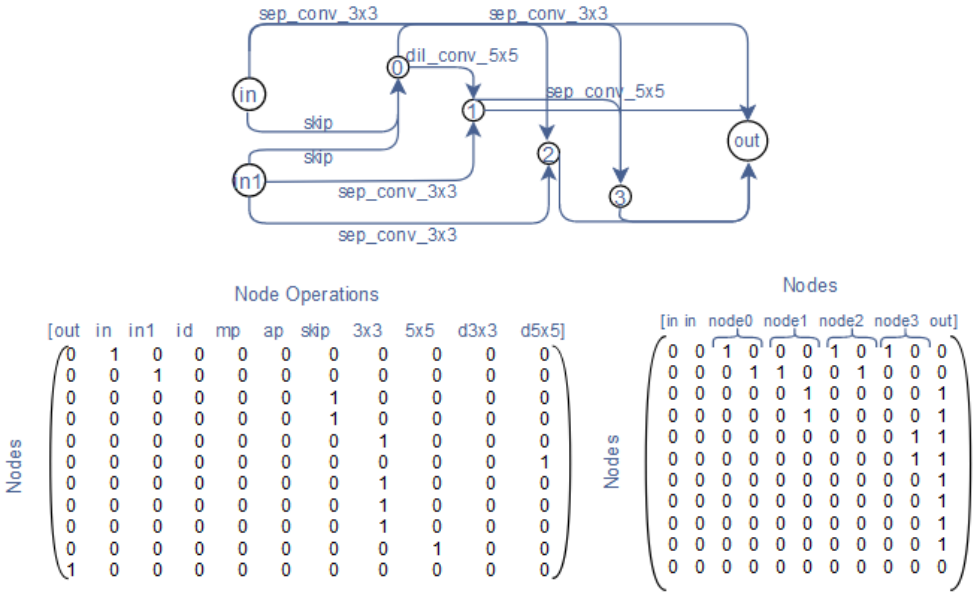


Figure F.3: Exemplary cell representation from the DARTS search space, where (**top**) visualizes the DAG representation in the DARTS search space, and (**bottom**) visualizes the node attribute matrix on the left and the respective adjacency matrix on the right.

F.1.4 NAS-Bench-NLP

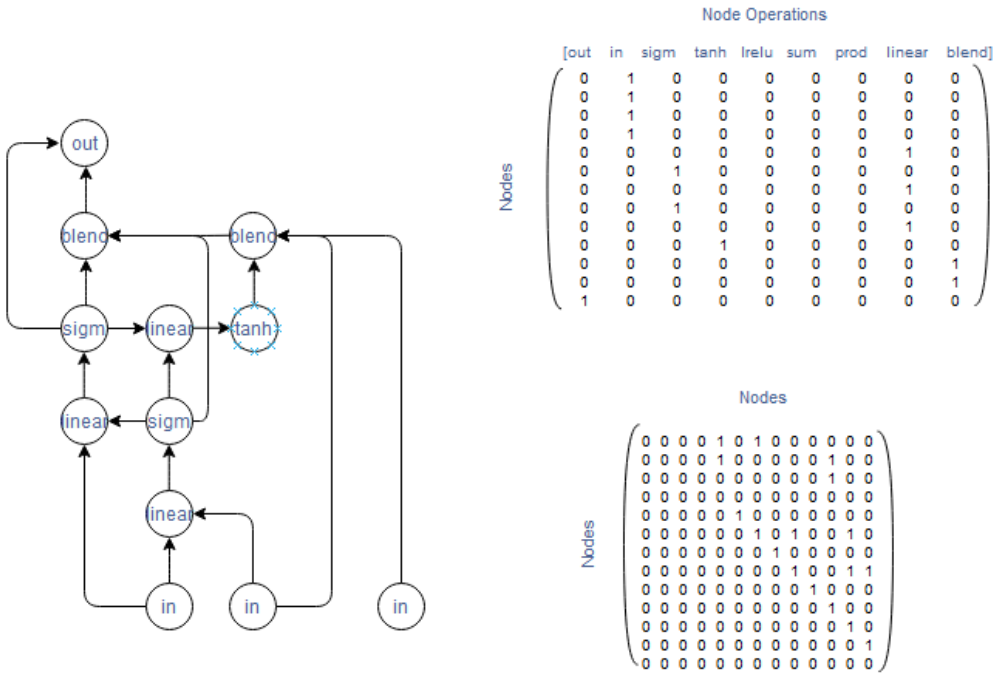


Figure F.4: Exemplary cell representation from NAS-Bench-NLP. (**left**) DAG representation of a graph with 12 nodes. (**right**) The top part shows the node attribute matrix to the DAG and the bottom part shows its adjacency matrix.

NAS-Bench-NLP [Kly+22] is the first Recurrent Neural Network (RNN)-derived benchmark for language modeling tasks. From the total 10^{53} possible architectures in the complete search space, 14 322 architectures are trained on Penn TreeBank [Mik+10] (PTB) and provided in this benchmark. The cell search space is constrained by the number of nodes $|V| \leq 24$, the number of hidden states $|H| \leq 3$ and the number of linear input vectors ≤ 3. The nodes represent the architecture operational layer and are chosen from the set $\mathcal{O} = \{$ linear, elementwise blending, elementwise product, elementwise sum, Tanh activation, Sigmoid activation, LeakyReLU activation $\}$.

For the experiments on NAS-Bench-NLP [Kly+22] we make use of the surrogate benchmark NAS-Bench-X11 [Yan+21] and the additional implementation in NAS-Bench-Suite [Meh+22]. Note, for the NAS-Bench-X11 evaluations, each architecture from the NAS-Bench-NLP search space must be trained for three epochs to use the surrogate model, whereas NAS-Bench-Suite provides the surrogate model for NAS-Bench-NLP without learning curve information, but also accompanying a lower Kendall Tau rank correlation. For fast evaluations we use the latter surrogate for our experiments. In order to use the surrogate benchmark, the architecture representation is the same used in Yan et al. [Yan+21] with the modification that each hidden node is connected to the output node. An exemplary architecture representation is visualized in Figure F.4. A next step is to analyse the 14 332 provided architectures on uniqueness, which leads

to 12 107 unique architectures. Furthermore, since Yan et al. [Yan+21] and Mehta et al. [Meh+22] only provide a surrogate model, which only considers architectures with up to 12 nodes, we also restrict our training data to this subset leading to a total of 7 258 architectures. We show experiments on NAS-Bench-NLP in Subsection 8.4.2.

F.1.5 Hardware-Aware-NAS-Bench

The recently introduced HW-NAS-Bench [Li+21] is the first public dataset for hardware NAS. It extends two representative NAS search spaces, NAS-Bench-201 [DY20] and FBNet [Wu+19], by providing measured and estimated hardware costs (i.e. latency and/or energy) for each device for all architectures in both search spaces. For this, HW-NAS-Bench considers six hardware devices: *Edge GPU* [NVI21], *Raspi 4* [Ras21], *Edge TPU* [Goo21b], *Pixel 3* [Goo21c], *ASIC-Eyeriss* [Che+16b] and *FPGA* [Xil21a; Xil21b]. In our experiments in Subsection 8.4.3, we consider the latency information on the NAS-Bench-201 search space.

F.2 Additional Ablation Studies

In this section we give an overview of ablation studies with respect to the proposed AG-Net.

F.2.1 Oracle Ablation

As we have seen in the previous section, our model AG-Net is able to find high-scoring architectures in various search spaces of different sizes and with different objectives. In addition, including the supposedly stronger predictor XGB [CG16] leads to improvements for the search on NAS-Bench-NLP [Kly+22]. In this section, we include an even stronger architecture accuracy evaluation model, i.e. the benchmark query input itself (oracle).

The comparison of the oracle benchmark (also including the ranking metric as for XGB in the main paper) to our AG-Net and XGB modifications are visualized in Figure F.5. This figure demonstrates the high performance of our model in the low query area.

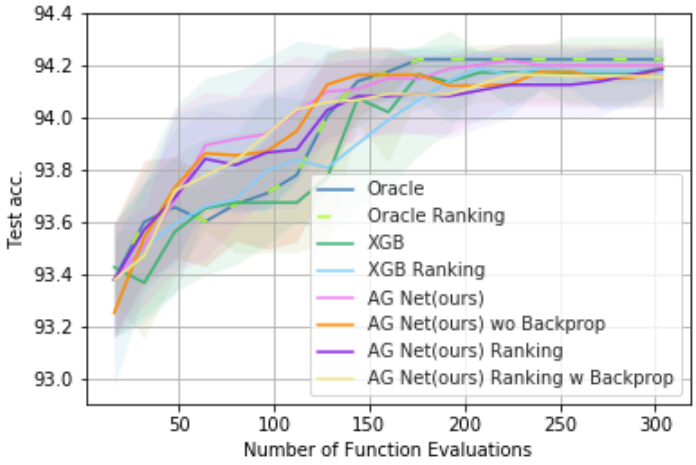


Figure F.5: Architecture search on NAS-Bench-101. Reported is the mean over 10 trials for the search of the best architecture in terms of validation accuracy on the CIFAR-10 image classification task compared to strong predictor models.

F.2.2 Latent Space Ablations

As we showed in Subsection 8.4.1, AG-Net improves over state-of-the art methods. For additional comparisons, we investigate different search methods in the latent space of the generative model, with samples $\mathbf{h}$ from a grid and also include baselines using the Latent Space Optimization (LSO) approach. For the first experiment we use the generator solely as a data sampler from the generator's latent space without any retraining, for the latter baseline we retrain the generator during the search. For the optimization, we use Bayesian optimization, local search, and random search.

Bayesian Optimization. We use DNGO [Sno+15] as our uncertainty prediction model for the Bayesian optimization search strategy, with the basis regression network being a one-layer Multilayer Perceptron (MLP) with a hidden dimensionality of 128, which is trained for 100 epochs and Expected Improvement (EI) [Moc74] as our acquisition function, which is mostly used in NAS. We set the best function value for the EI evaluation as the best validation accuracy of the training data. We sample 16 initial random latent space variables $\mathbf{h} \sim \mathcal{U}[-3, 3]$ and decode them to graphs using our pretrained generative model. These latent space variables and their corresponding validation architecture performances are then inputs for the DNGO model for training. Again, the best 16 architectures are selected using EI in each round to be evaluated and added to the training data. This search ends when the total query amount of 300 is reached.

Random and Local Search. In addition to Bayesian Optimization as a comparison, we also include a random search [LT19] and local search investigation. Recently, White, Nolen, and Savani [WNS21b] show that local search is a powerful NAS baseline, resulting in competitive results. Local search [WNS21b] evaluates samples and their neighborhood uniformly at random. An option to define the neighborhood is the set of architectures which differ from a sampled architecture by one node or edge. This can be done only in the discrete search space, given for example by the tabular NAS-Benchmarks. We have to adapt the neighborhood definition in our latent space for local search in this space. We sample a latent space variable $\mathbf{h} \sim \mathcal{U}[-3, 3]$, decode it and evaluate the generated neural architecture. Here, we define neighborhood as the Euclidean space around the sampled latent variable $U_\epsilon(z) = \{y \sim \mathcal{U}[-3, 3] | d(z, y) < \epsilon\}$, with ϵ being sufficiently small. This neighborhood is then investigated until a local optimum in terms of validation accuracy is reached. Furthermore, we include a random search and local search comparison using weighted retraining. Here, we retrain the generative model in each search iteration for 1 epoch with the weighted objective function, ceteris paribus.

To compare with weight-sharing approaches, we also compare to the supernet from Huang and Chu [HC21] for the NAS-Bench-201 search space. To compare our AG-Net with SGNAS, we use the supernet as our surrogate model to predict the architectures performance while retraining the generative model in the weighted manner. The results of our ablation studies are reported in Table F.1. AG-Net improves over search methods on the latent space with and without LSO on both benchmarks, demonstrating that our generator in combination with our MLP surrogate model learns to adapt the distribution shift constructed by the weighted retraining best.

For further visualizations we also plot different ablation search methods over different query numbers in Figure F.6 for NAS-Bench-101 and Figures F.7, F.8, and F.9 for NAS-Bench-201. These figures demonstrates the high any-time performance of our method on both search spaces. For any number of available queries, our model is better in finding high-performing architectures from the latent space than other latent space based methods.

F.2.3 Predictor Ablation – Local Solution

Our proposed method, consisting of the generative and surrogate model combined with latent space optimization encourages the architecture search to focus on promising regions in the search space. This method could be trapped in local solutions, which we investigate experimentally in the following. First, the previous section points out that AG-Net improves over both local search methods with and without the latent space optimization approach. Thus, we assume that the latent space optimization learns properties of high-scoring architectures without being easily trapped in poor local solutions. The amount of samples drawn in each search iteration also provides a trade-off between diversity versus specificity. To investigate further how easily AG-Net could be trapped in a local solution, we test our method when it only uses in total the best k (predicted) architectures from our test samples and the training data as a new training set for the next search iteration (degenerative) and is thereby encouraged to forget about worse performing architectures. Figure F.10 shows the search behavior of the degenerative model with $k = 16$ and $k = 32$. Even in this case, AG-Net is not easily trapped in poor solutions.

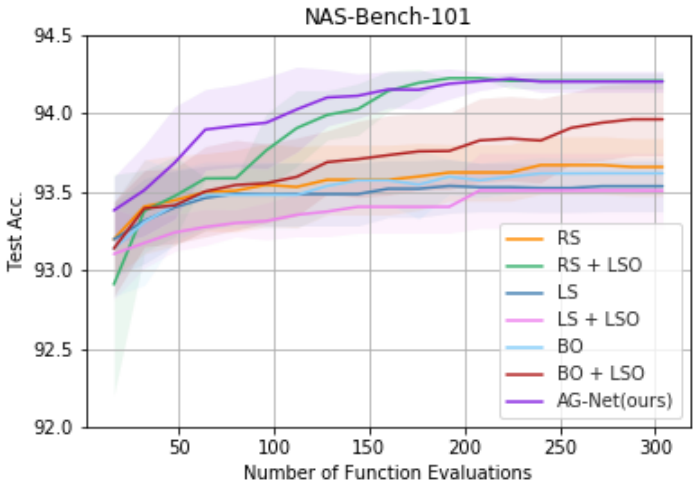


Figure F.6: Ablation: NAS on NAS-Bench-101 over 10 trials. We compare AG-Net to Random Search (RS), Local Search (LS), and Bayesian Optimization (BO) with and without additional Latent Space Optimization (LSO).

Table F.1: Ablation: Search results on NAS-Bench-101 and NAS-Bench-201 on the AG-Net latent space (mean over 10 trials with a query budget of 300).

	NAS-Bench-101		**NAS-Bench-201**					
	CIFAR-10		CIFAR-10		CIFAR-100		ImageNet16-120	
	Val. Acc	Test Acc	Val. Acc	Test Acc	Val. Acc	Test Acc	Val. Acc	Test Acc
Optimum*	95.06	94.32	91.61	94.37	73.49	73.51	46.77	47.31
Random Search	94.27	93.65	91.37	93.92	72.55	72.49	46.09	46.05
Local Search	94.31	93.66	91.28	94.01	72.52	72.59	45.89	46.07
Bayesian Optimization	94.27	93.62	91.30	93.99	72.23	72.35	46.09	46.01
Random Search + LSO	94.64	**94.20**	**91.61***	**94.37***	**73.49***	**73.51***	**46.77***	45.47
Local Search + LSO	94.17	93.50	91.30	93.96	72.43	72.58	45.83	45.95
Bayesian Optimization +LSO	94.50	93.96	91.43	94.17	72.64	72.67	46.30	45.91
SGNAS [HC21] + LSO	-	-	**91.61***	**94.37***	73.04	73.12	46.56	**46.32**
AG-Net (ours)	**94.96**	**94.20**	**91.61***	**94.37***	**73.49***	**73.51***	46.67	46.22

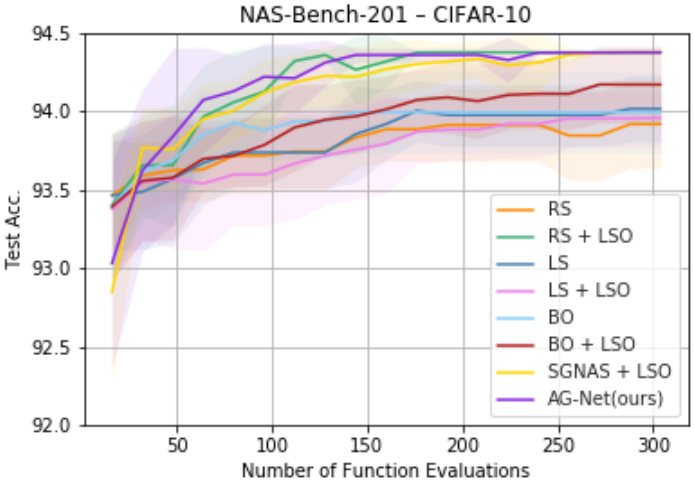


Figure F.7: Ablation: NAS on NAS-Bench-201 (CIFAR-10) over 10 trials. We compare AG-Net to Random Search (RS), Local Search (LS), Bayesian Optimization (BO), and SGNAS [HC21] with (+ LSO) and without retraining.

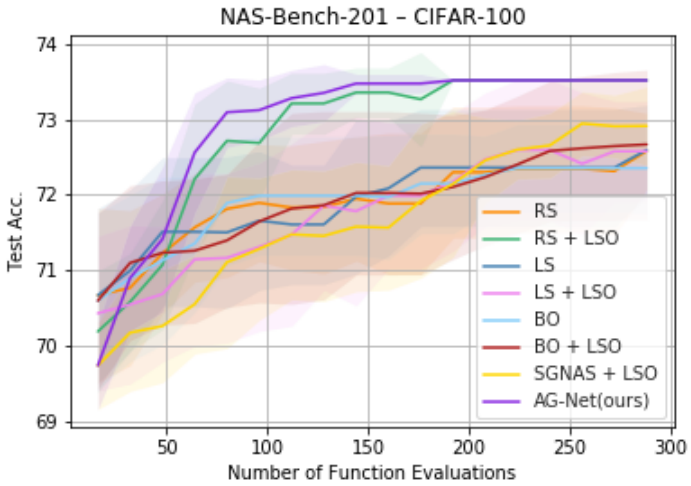


Figure F.8: Ablation: NAS on NAS-Bench-201 (CIFAR-100) over 10 trials. We compare AG-Net to Random Search (RS), Local Search (LS), Bayesian Optimization (BO), and SGNAS [HC21] with (+ LSO) and without retraining.

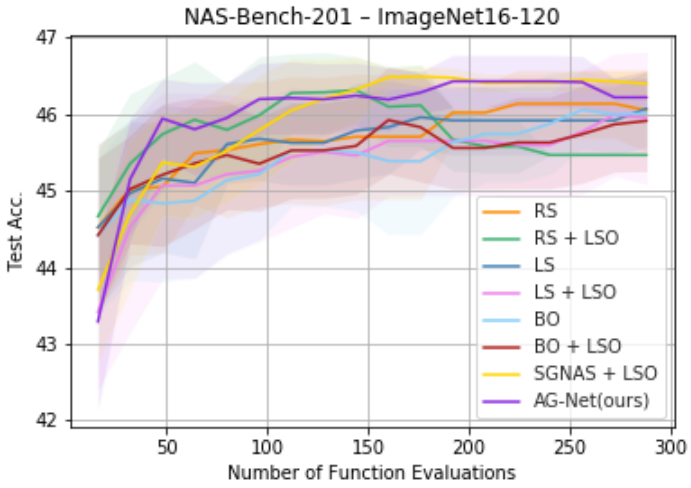


Figure F.9: Ablation: NAS on NAS-Bench-201 (ImageNet16-120) over 10 trials. We compare AG-Net to Random Search (RS), Local Search (LS), Bayesian Optimization (BO), and SGNAS [HC21] with (+ LSO) and without retraining.

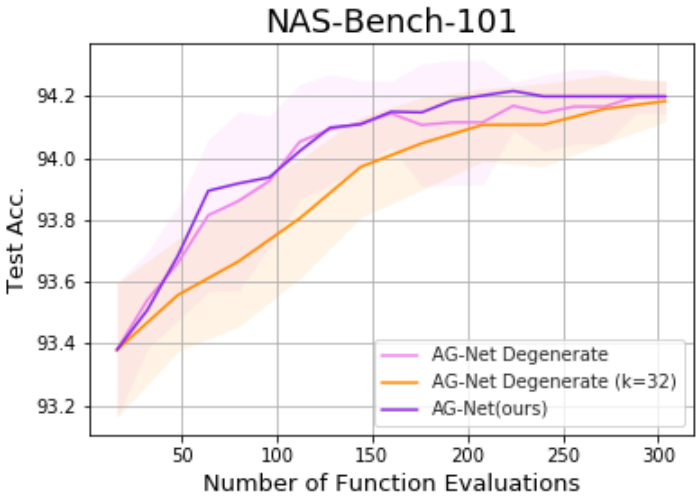


Figure F.10: Architecture search on NAS-Bench-101 in the degenerate setting. Reported is the mean over 10 trials.

F.3 Experiments: Implementation Details

F.3.1 Surrogate Model

In this section, we present details about the surrogate models used in the main paper. The MLP surrogate model used for our AG-Net is a 4-layer MLP with ReLU activation functions. The hidden size equals the input size. The input to the MLP surrogate model is the vector representation $\in \mathbb{R}^n$ of our graphs: a concatenation of the flattened node attribute matrix and flattened upper triangular matrix of the adjacency matrix, which presents the edge scores, see Section F.1 for visualizations. Note, the vector dimension n differs across the search spaces due to the different maximal amount of nodes. Our AG-Net passes the output of our generator, i.e. a generated vector representation, as the direct input to our MLP surrogate model.

We consider XGB [CG16] as an alternative surrogate prediction model. The input to this prediction model is the vector representation of the architecture.

F.3.2 Search Algorithm

High-level descriptions of the unconstrained (Subsection 8.4.1) and constrained (Subsection 8.4.3) versions of our search algorithm are depicted in Algorithm 1 and Algorithm 2 respectively.

F.3.3 NAS-Bench-101

Table F.2 is the detailed version of Table 8.1 including standard deviation.

F.3.4 NAS-Bench-201

Table F.4 is the version of Table 8.2 and Table 8.3 including standard deviation.

F.3.5 DARTS Search Space

Table F.3 is the detailed version of Table 8.4 including standard deviation.

Search Process using NAS-Bench-301. For experiments in the DARTS [LSY19] search space, we first train our generative model on generating valid cells, as visualized in Figure F.3; here we do not distinguish between generating a normal or a reduction cell. Having a pretrained generative model for generating valid cell representations in the DARTS search space allows for searching well-performing architectures. Here we describe the search process for architectures evaluated on CIFAR-10 using the surrogate benchmark NAS-Bench-301 [Zel+22]. Since the DARTS search space is defined by a normal and reduction cell, we have to adapt the search process, compared to the search in the tabular benchmark search spaces, where the architectures differ between the DAG. We begin the search by randomly sampling 16 architectures from NAS-Bench-301. Next, we generate one normal cell. This cell is used to search for the best reduction cell in terms of the accuracy given by the surrogate benchmark NAS-Bench-301, in combination with the randomly sampled cell. This search procedure then follows the same steps as for the tabular benchmarks and stops after we reach a query amount of 155. Now, we can use the best found reduction cell as a fixed starting point to search for the best normal cell in the same manner as before. The overall search stops after a maximal amount of 310 queries. The search outcome differs between starting

Algorithm 1: Unconstrained Search Algorithm.

Input: (i) Search space p_{D}
Input: (ii) Pretrained generator G
Input: (iii) Untrained performance predictor P
Input: (iv) Query budget b
Input: (v) e epochs to train G and P
▷ `Initialize training data`
$\mathbf{D} \leftarrow \{\}$
while $|\mathbf{D}| < 16$ **do**
  $\mathbf{D} \leftarrow \mathbf{D} \cup \{d \sim p_{\mathrm{D}}\}$
end
▷ `Evaluate architectures (get accuracies on target image dataset)`
$\mathbf{D} \leftarrow \mathrm{eval}(\mathbf{D})$
▷ `Randomly initialize predictor weights`
$P \leftarrow \mathrm{init}(P)$
▷ `Search loop`
while $|\mathbf{D}| < b$ **do**
  ▷ `Weight training data by performance`
  $\mathbf{D}_{\mathrm{w}} \leftarrow \mathrm{weight}(\mathbf{D})$
  ▷ `Train generator and predictor`
  $\mathrm{train}(G, P, \mathbf{D}_{\mathrm{w}}, e)$
  ▷ `Generate 100 candidates`
  $\mathbf{D}_{\mathrm{cand}} \leftarrow \{\}$
  while $|\mathbf{D}_{cand}| < 100$ **do**
    $\mathbf{h} \sim \mathcal{U}[-3, 3]$
    $\mathbf{D}_{\mathrm{cand}} \leftarrow \mathbf{D}_{\mathrm{cand}} \cup G(\mathbf{h})$
  end
  ▷ `Select top-16 candidates with P`
  $\mathbf{D}_{\mathrm{cand}} \leftarrow \mathrm{select}(\mathbf{D}_{\mathrm{cand}}, P, 16)$
  ▷ `Evaluate and add to data`
  $\mathbf{D} \leftarrow \mathbf{D} \cup \mathrm{eval}(\mathbf{D}_{\mathrm{cand}})$
end

Algorithm 2: Constrained Search Algorithm.

Input: (i) Search space p_{D}
Input: (ii) Pretrained generator G
Input: (iii) Untrained performance predictor P_{a}
Input: (iv) Set of constraint predictors P_c
Input: (v) Query budget b
Input: (vi) e epochs to train G and P
Input: (vii) Set of constraints C

▷ `Initialize training data`
$\mathbf{D} \leftarrow \{\}$
while $|\mathbf{D}| < 16$ **do**
  $\mathbf{D} \leftarrow \mathbf{D} \cup \{d \sim p_{\mathrm{D}}\}$
end
▷ `Evaluate architectures (get accuracies and constraints on target image dataset)`
$\mathbf{D} \leftarrow \mathrm{eval}(\mathbf{D})$
▷ `Randomly initialize predictor weights`
$P_{\mathrm{a}} \leftarrow \mathrm{init}(P_{\mathrm{a}})$
foreach $P \in P_c$ **do**
  $P \leftarrow \mathrm{init}(P)$
end
▷ `Search loop`
while $|\mathbf{D}| < b$ **do**
  ▷ `Weight train data by performance and constraints`
  $\mathbf{D}_{\mathrm{w}} \leftarrow \mathrm{weight}(\mathbf{D}, C)$
  ▷ `Train generator and predictors`
  $\mathrm{train}(G, P_{\mathrm{a}}, P_c, \mathbf{D}_{\mathrm{w}}, e)$
  ▷ `Generate 100 candidates`
  $\mathbf{D}_{\mathrm{cand}} \leftarrow \{\}$
  while $|\mathbf{D}_{cand}| < 100$ **do**
    $\mathbf{h} \sim \mathcal{U}[-3, 3]$
    $\mathbf{D}_{\mathrm{cand}} \leftarrow \mathbf{D}_{\mathrm{cand}} \cup G(\mathbf{h})$
  end
  ▷ `Select top16 candidates with` P_{a} `and` P_c
  $\mathbf{D}_{\mathrm{cand}} \leftarrow \mathrm{select}(\mathbf{D}_{\mathrm{cand}}, P_{\mathrm{a}}, P_c, 16)$
  ▷ `Evaluate and add to data`
  $\mathbf{D} \leftarrow \mathbf{D} \cup \mathrm{eval}(\mathbf{D}_{\mathrm{cand}})$
end

Table F.2: Architecture search on NAS-Bench-101. Reported is the mean ± standard deviation over 10 trials for the search of the best architecture in terms of validation accuracy on the CIFAR-10 image classification task compared to state-of-the-art methods.

NAS Method	Val. Acc	Test Acc	Queries
Optimum*	95.06*	94.32*	–
Arch2vec + RL [Yan+20]	–	94.10	400
Arch2vec + BO [Yan+20]	–	94.05	400
NAO ‡[Luo+18]	94.66 ± 0.14	93.49 ± 0.59	192
BANANAS† [WNS21a]	94.73 ± 0.17	94.09 ± 0.19	192
Bayesian Optimization† [Sno+15]	94.57 ± 0.20	93.96 ± 0.21	192
Local Search† [WNS21b]	94.57 ± 0.15	93.97 ± 0.13	192
Random Search† [LT19]	94.31 ± 0.15	93.61 ± 0.27	192
Regularized Evolution* [Rea+19]	94.47 ± 0.11	93.89 ± 0.20	192
WeakNAS [Wu+21]	–	**94.18** ± 0.14	200
XGB (ours)	94.61 ± 0.04	94.13 ± 0.11	192
XGB + Ranking (ours)	94.60 ± 0.08	94.14 ± 0.19	192
AG-Net (ours)	**94.90** ± 0.22	**94.18** ± 0.10	192

Table F.3: Results on NAS-Bench-301 (mean ± standard deviation over 50 trials) for the search of the best architecture in terms of validation accuracy compared to state-of-the-art methods.

NAS Method	Val. Acc	Queries
BANANAS† [WNS21a]	94.77 ± 0.10	192
Bayesian Optimization† [Sno+15]	94.71 ± 0.10	192
Local Search† [WNS21b]	**95.02** ± 0.10	192
Random Search† [LT19]	94.31 ± 0.12	192
Regularized Evolution† [Rea+19]	94.75 ± 0.11	192
XGB (ours)	94.79 ± 0.13	192
XGR + Ranking (ours)	94.76 ± 0.14	192
AG-Net (ours)	94.79 ± 0.12	192

Table F.4: Architecture Search on NAS-Bench-201. We report the mean ± standard deviation over 10 trials for the architecture with the highest validation accuracy. For comparable numbers of queries, AG-Net performs similarly or better than the previous state of the art.

NAS Method	**CIFAR-10**		**CIFAR-100**		**ImageNet16-120**		**Queries**
	Val. Acc	Test Acc	Val. Acc	Test Acc	Val. Acc	Test Acc	
Optimum*	91.61*	94.37*	73.49*	73.51*	46.73*	47.31*	-
SGNAS [HC21]	90.18 ± 0.31	93.53 ± 0.12	70.28 ± 1.20	70.31 ± 1.09	44.65 ± 2.32	44.98 ± 2.10	-
Arch2vec + BO [Yan+20]	91.41 ± 0.22	94.18 ± 0.24	$\mathbf{73.35} \pm 0.32$	$\mathbf{73.37} \pm 0.30$	46.34 ± 0.18	46.27 ± 0.37	100
AG-Net (ours)	$\mathbf{91.55} \pm 0.08$	$\mathbf{94.24} \pm 0.19$	73.20 ± 0.34	73.12 ± 0.40	46.31 ± 0.33	46.20 ± 0.47	96
AG-Net (ours with topk=1)	91.41 ± 0.30	94.16 ± 0.31	73.14 ± 0.56	73.15 ± 0.54	$\mathbf{46.42} \pm 0.14$	$\mathbf{46.43} \pm 0.30$	100
BANANAS† [WNS21a]	$\mathbf{91.56} \pm 0.14$	94.3 ± 0.22	$\mathbf{73.49}^* \pm 0.00$	73.50 ± 0.00	$\mathbf{46.65} \pm 0.13$	$\mathbf{46.51} \pm 0.11$	192
BO† [Sno+15]	91.54 ± 0.06	94.22 ± 0.18	73.26 ± 0.19	73.22 ± 0.27	46.43 ± 0.35	46.40 ± 0.35	192
RS † [LT19]	91.12 ± 0.26	93.89 ± 0.27	72.08 ± 0.53	72.07 ± 0.61	45.87 ± 0.39	45.98 ± 0.41	192
XGB (ours)	91.54 ± 0.09	94.34 ± 0.10	73.10 ± 0.51	72.93 ± 0.74	46.48 ± 0.13	46.08 ± 0.79	192
XGB + Ranking (ours)	91.48 ± 0.12	94.25 ± 0.15	73.20 ± 0.36	73.24 ± 0.34	46.40 ± 0.28	46.16 ± 0.64	192
AG-Net (ours)	$\mathbf{91.60} \pm 0.02$	$\mathbf{94.37}* \pm 0.00$	$\mathbf{73.49}^* \pm 0.00$	$\mathbf{73.51}^* \pm 0.00$	46.64 ± 0.12	46.43 ± 0.34	192
GANAS [Rez+21]	-	94.34 ± 0.05	-	73.28 ± 0.17	-	$\mathbf{46.80} \pm 0.29$	444
AG-Net (ours)	$\mathbf{91.61}* \pm 0.00$	$\mathbf{94.37}* \pm 0.00$	$\mathbf{73.49}^* \pm 0.00$	$\mathbf{73.51}^* \pm 0.00$	$\mathbf{46.73}^* \pm 0.00$	46.42 ± 0.00	400

Table F.5: Results on NAS-Bench-NLP (mean ± standard deviation over 100 trials) for the search of the best architecture in terms of validation perplexity compared to state-of-the-art methods.

NAS Method	Val. Perplexity	Queries
BANANAS† [WNS21a]	95.68 ± 0.16	304
Local Search† [WNS21b]	95.69 ± 0.18	304
Random Search† [LT19]	95.64 ± 0.19	304
Regularized Evolution† [Rea+19]	95.66 ± 0.21	304
XGB (ours)	**95.95** ± 0.20	304
XGR + Ranking (ours)	95.92 ± 0.19	304
AG-Net (ours)	95.86 ± 0.18	304

with a reduction or the normal cell. The search procedure starting with a random reduction cell is analogous. In the main paper, we report the search outcome for NAS-Bench-301 [Zel+22] starting with a random reduction cell.

Search Process using TENAS. As we described in the previous section, the search in the DARTS [LSY19] search space needs adaptions in the search procedure. Here we describe the further adaption of using training free measurements instead of the NAS-Bench-301 prediction. The training free measurements are based on the recent paper TE-NAS [CGW21], which ranks architectures by analysing the neural tangent kernel, by its condition number (KN), and the number of linear regions (NLR) of each architecture. Concretely, for the search on ImageNet [Den+09] we search for architectures in terms of their KN value and their number of linear regions instead of their validation accuracy. In the beginning of our search we generate three random normal cells. These cells are used to search for an optimal reduction cell optimizing both KN and NLR measurements. In each search iteration we generate reduction cells and calculate the KN and NLR for each combination of normal cell and reduction cell. The reduction cells are ranked according to their mean KN and their mean NLR (mean in terms of all three normal cells). The 16 best ranked reduction cells are then used for the next iteration of reduction cell search. The reduction cell search stops, when a maximum of 104 queries is reached. After that we use the best found reduction cell in terms of the lowest KN and the highest NLR for the next search for a normal cell. The next steps use this best found reduction cell as a starting point and searches for the best normal cell in the same manner as before. The search stops after a total of 208 queries and outputs an overall normal and reduction cell combination, leading to a DARTS [LSY19] architecture, which we train on ImageNet [Den+09] using the same training pipeline as Chen, Gong, and Wang [CGW21].

F.3.6 NAS-Bench-NLP

Table F.5 is the detailed version of Table 8.5 including standard deviation.

F.3.7 Hardware-Aware NAS-Bench

In comparison to the experiments for NAS-Bench-101 [Yin+19b] and NAS-Bench-201 [DY20], the search on the Hardware-Aware NAS-Bench [Li+21] changes to be a multi-objective learning procedure. We compare two different objective settings: i) a joint constrained optimization in Equation 8.4 and ii) a constrained optimization in Equation 8.5. For both settings we need to adapt the surrogate model by including an additional predictor $g(\cdot)$ for latency. We implement $g(\cdot)$ equally to the performance predictor $f(\cdot)$, whereas both predictors share weights in our experiments. We give a detailed overview of the hyperparameter settings in Section F.5. Since we include an additional predictor, the training objective needs to be updated, as seen in Equation 8.6 with multiple targets. The risk of including multiple targets to the training objective is an exploding loss leading to reduced valid generation ability of our generative network. In order to overcome this problem, we scale each loss term by the largest one, such that each term is at most 1. This way, we have a more stable training.

Exemplary Searches for Other Devices. In Figure 8.7 we showed an exemplary search result comparing random search with both of our constrained algorithm settings in the case of different latency constraints on a Pixel3. In the following, we show more examples on different devices in Figure F.11. These plots show that both methods *Joint=1* and *Joint=0* outperform the random search baseline in all different device experiments. The same results as in the main paper holds therefore for all other devices too; *Joint=1* is able to find better performing architectures compared to *Joint=0* if the latency constraint L restricts the feasible search space strongly.

Search Progress and Baselines. Local search [WNS21b] is considered a strong baseline in NAS. In the case of constrained searches (as in HW-NAS-Bench), we noticed that it cannot perform well without adaptation. The vanilla local search algorithm expects as input a single randomly drawn architecture. However, this architecture is not guaranteed to be feasible as its latency can be larger than the latency constraint. To circumvent this, we performed local search in the following settings: (a) local search vanilla setting with one randomly drawn architecture, and (b) local search initialized with 16 randomly drawn architectures. In each setting, local search continues to search the neighborhood of the next best architecture that satisfies the latency constraint. We noticed that initializing local search with 16 randomly drawn architectures improves its performance substantially, however, it is still not on par with random search [LT19] in this setting. Consequently, we only show random search as the baseline in Table 8.7 to improve readability. In Figure F.12 we show the progress of our algorithms (*Joint=0* and *Joint=1*) compared to random search and local search.

F.4 Generator Details

Generator Evaluation. We examine the generation ability of our generator [Yan+20; Luk+21]. The training on the surrogate benchmarks is a priori only on a subset of the overall dataset. For that we train on 90% of the overall dataset and keep a hold-out dataset of 10%. Then, we sample 10 000 random variables $\mathbf{z} \sim \mathcal{N}(0, 1)$ and decode them to graphs. We report the results of this investigation in Table F.6. Here, validity describes the ratio of valid graphs our generator model generates, uniqueness describes the portion of unique graphs from the valid generated graphs, and novelty is the portion of generated graphs *not in the training set*. It is not surprising for the NAS-Bench-301 and NAS-Bench-NLP search spaces that our model is able to generate 100% unique and novel graphs given the large size of both search spaces. This demonstrates that our generator model is able to generate valid graphs with high novelty and, consequently, is able to cover a substantial part of the search space.

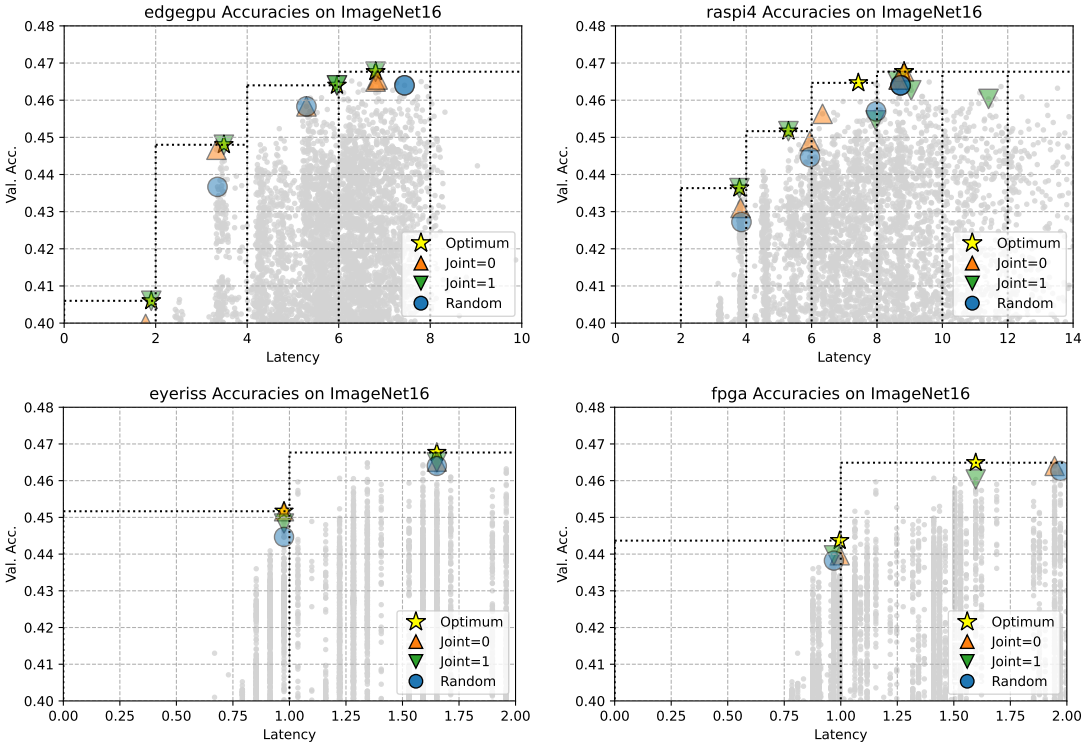


Figure F.11: Exemplary searches on HW-NAS-Bench for image classification on ImageNet16 with 192 queries on Edge GPU, Raspi4, Eyeriss, FPGA and latency conditions $L \in \{2, 4, 6, 8, 10\}$, $L \in \{2, 4, 6, 8, 10, 12, 14\}$ and $L \in \{1, 2\}$ (y-axis zoomed for visibility).

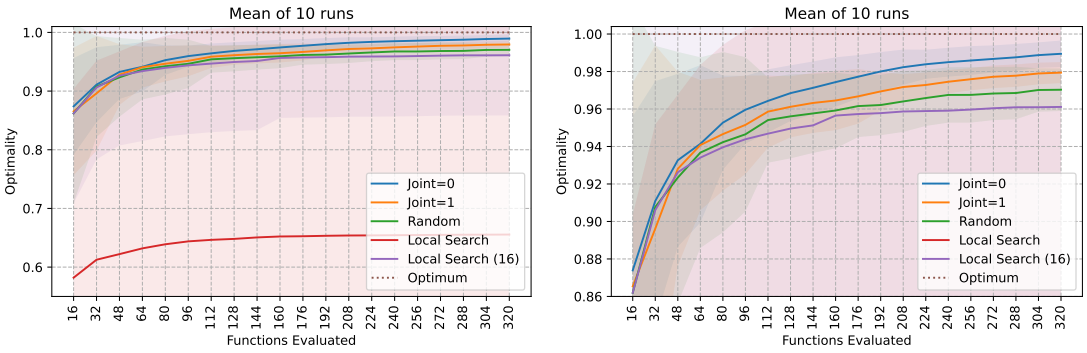


Figure F.12: (**left**) Optimality for all search parameters in Table 8.7 at any time during the search progress in terms of the number of evaluated architectures (up to 320). Optimality is the mean validation accuracy of 10 runs per algorithm, normalized by the optimal value for each parameter setting (hence, optimum is at 1.0). (**right**) zoomed y-axis

Table F.6: Generator Abilities and training costs. The proposed generator generates architectures with high validity and uniqueness scores. The novelty scores are in a similar range as for previous methods [Luk+21].

Search Space	Validity %	Uniqueness %	Novelty %	Training Time (GPU days)
NAS-Bench-101	71.69	97.92	62.30	0.4
NAS-Bench-201	99.97	73.61	10.03	0.3
NAS-Bench-301	42.27	100.00	100.00	0.9
NAS-Bench-NLP	57.95	100.00	100.00	0.7

Generator Implementation Details. In this section we present more details about the generation model SVGe from Lukasik et al. [Luk+21]. The pseudo algorithm is described in Algorithm 3. The modules f_{initNode}, f_{addNode}, f_{addEdges}, $f_{\text{Embedding}}$ used in this code are two-layer MLPs with ReLU activation functions. Note, in contrast to SVGe, we don't sample within the generation process, in order to allow for end-to-end learning with the prediction model for AG-Net.

Algorithm 3: Graph Generation.

Input: $\mathbf{h}_{\text{in}} \sim \mathcal{N}(0,1)$
Output: random sampled reconstructed graph $\widetilde{G} = (\widetilde{V}, \widetilde{E})$

initialize one-hot encoded InputNode v_0, with embedding $\mathbf{h}_0 \leftarrow f_{\text{initNode}}(\mathbf{h}_{\text{in}}, f_{\text{Embedding}})$
$V \leftarrow \{v_0\}$, $E \leftarrow \emptyset$, $\mathbf{h}_G \leftarrow \mathbf{h}_{\text{in}}$,
while $|V| \leq$ *Max Number of Nodes* **do**
  $v_{t+1} \leftarrow f_{\text{addNode}}(\mathbf{h}_{\text{in}}, \mathbf{h}_G)$
  $V \leftarrow V \cup \{v_{t+1}\}$
  $\mathbf{h}_{t+1} \leftarrow f_{\text{initNode}}(\mathbf{h}_{\text{in}}, \mathbf{h}_G, f_{\text{Embedding}}(v_{t+1})])$
  for $v_j \in V \setminus v_{t+1}$ **do**
    $s_{\text{addEdges}}(j, t+1) \leftarrow f_{\text{addEdges}}(\mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{h}_G, \mathbf{h}_{\text{in}})$
    $e_{(j,t+1)} \sim \text{Eval}(s_{\text{addEdges}}(j, t+1))$; ▷ `evaluate whether to add edge`
    if $e_{(j,t+1)} = 1$ **then**
      $E \leftarrow E \cup \{e_{(j,t+1)} = (v_j, v_{t+1})\}$
    end
  end
  $\mathbf{h}_t \leftarrow \text{concat}(\mathbf{h}_t, \mathbf{h}_{t+1})$
  $G \leftarrow (V, E)$
  $\mathbf{h}_t \leftarrow (\mathbf{h}_t, G)$; ▷ `update node embeddings`
  $\mathbf{h}_G \leftarrow \text{aggregate}(\mathbf{h}_t)$; ▷ `update graph embedding`
  $t \leftarrow t + 1$
end
$V \sim \text{Categorical}(V)$; ▷ `Sample node types`
$E \sim \text{Ber}(E)$; ▷ `Sample edges`
$\widetilde{G} = (V, E)$

F.5 Hyperparameters

In this section we give a detailed overview about the hyperparameter settings for our generative network. We use Pytorch [Pas+19] and Pytorch Geometric [FL19] for all our implementations.

F.5.1 Generator

Table F.7 presents all used hyperparameters for the generator training. We train our generator in a ticked manner; after every 5 000 training data samples, we evaluate our generator for validity. The used pretrained checkpoint for our search is then the one, which has the highest validity (defined by randomly sampling 10 000 latent vectors $\mathbf{z} \in \mathbb{R}^{32}$ and decoding architectures). The training is the same for all different search spaces.

Table F.7: Hyperparameters of the generator model.

Hyperparameter	Default Value
Node Embedding	32
Latent Vector	32
MLP Node Embedding layer	2
GNN layer	2
Batch Size	32
Optimizer	Adam [KB15]
Learning Rate	0.0002
Betas	(0.5, 0.999)
Ticks	500
Tick Size	5,000

F.5.2 Surrogate Model

The overall surrogate is an MLP with ReLU activations. Table F.8 and Table F.9 list all hyperparameters for the search experiments in the main paper for the simple performance surrogate model and the multi-objective surrogate model for the additional hardware objective. The hyperparameters for XGB [CG16] are the same as in Mehta et al. [Meh+22].

Table F.8: Hyperparameters for the performance surrogate model $f(\cdot)$.

Hyperparameter	Dataset			
	NB101	NB201	NB301	NBNLP
α	0.9			
MLP Layers	4			
MLP Hidden	56	84	176	559
Epochs	15	30	15	30
Optimizer	Adam [KB15]			
LR	0.001			
Betas	(0.5, 0.999)			
weight factor	0.001			
batch size	16			
loss	L2			

Table F.9: Hyperparameters for both surrogate models $f(\cdot)$ and $g(\cdot)$ for the multi-objective search in the Hardware-Aware Benchmark.

Hyperparameter	Hardware-Aware NASBench
α	0.95
λ	0.5
MLP Layers	4
MLP Hidden	82
Epochs	30
Optimizer	Adam [KB15]
LR	0.002
Betas	(0.5, 0.999)
weight factor	0.001
penalty term	1000
batch size	16
loss	L2

F.6 Latent Space Optimization Visualization

A more descriptive visualization of the latent space optimization technique used for our AG-Net neural architecture search is displayed in Figure F.13.

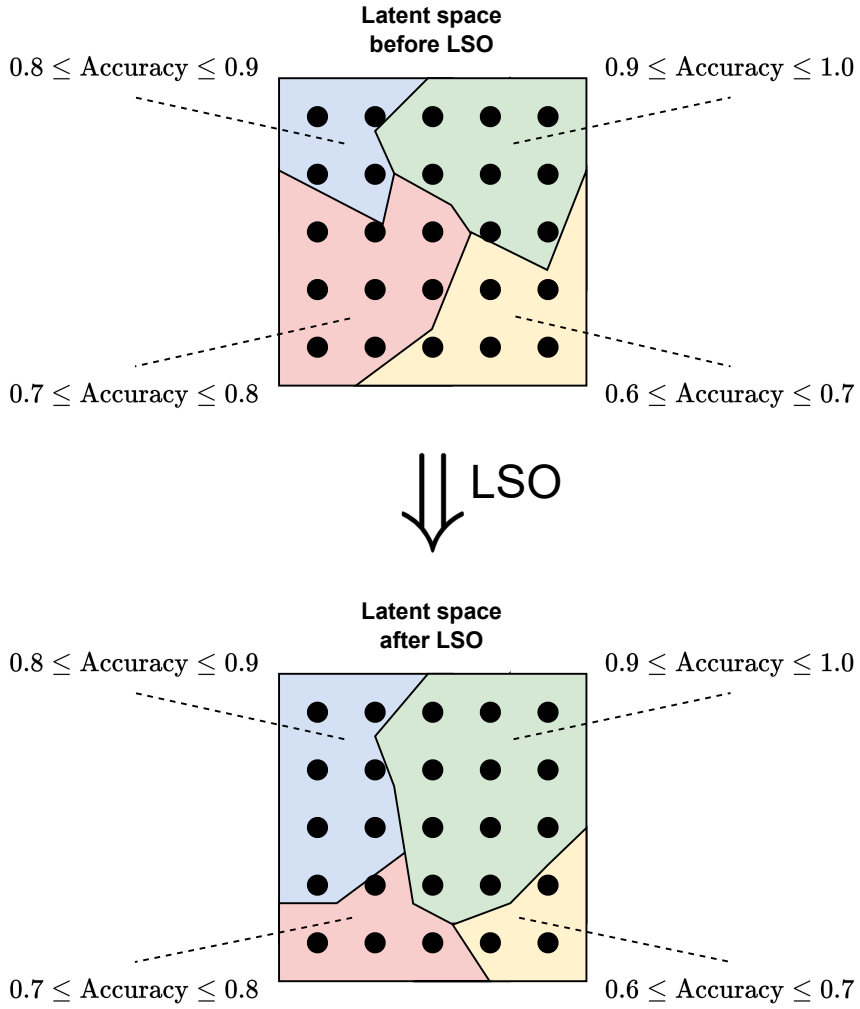


Figure F.13: The latent space is reshaped in a way that promotes desired properties of generated architectures (here: accuracy). Consequently, it becomes more likely for the generator to generate architectures satisfying this property.

Chapter G
Data for Robust Neural Architecture Design

In this supplementary material, we provide several additional details and visualizations:

- Section G.1: Details about the generation and structure of the dataset.
- Section G.2: Figure depicting the correlations between evaluated image datasets.
- Section G.3: Example images for the corruptions in CIFAR-10-C.
- Section G.4: Figures that are presented in the main chapter for CIFAR-10 are here shown for CIFAR-100 and ImageNet16-120.
- Section G.5: Analysis of the effect of architectural design choices on robustness of a network.

G.1 Dataset Generation

G.1.1 NAS-Bench-201

We base our evaluations on the NAS-Bench-201 [DY20] search space. It is a cell-based architecture search space. Each cell has in total 4 nodes and 6 edges. The nodes in this search space correspond to the architecture's feature maps and the edges represent the architectures operation, which are chosen from the operation set $\mathcal{O} = \{$ 1×1 conv., 3×3 conv., 3×3 avg. pooling, skip, zero $\}$ (see Figure 9.1). This search space contains in total $5^6 = 15\,625$ architectures, from which only 6466 are unique, since the operations skip and zero can cause isomorphic cells (see Figure G.1), where the latter operation zero stands for dropping the edge. Each architecture is trained on three different image datasets for 200 epochs: CIFAR-10 [Kri09], CIFAR-100 [Kri09] and ImageNet16-120 [CLH17]. For our evaluations, we consider all unique architectures in the search space and test splits of the corresponding datasets. Hence, we evaluate $3 \cdot 6466 = 19\,398$ pretrained networks in total.

G.1.2 Dataset Gathering

We collect evaluations for our dataset for different corruptions and adversarial attacks (as discussed in Subsection 9.3.2 and Subsection 9.3.3) following Algorithm 4. This process is also depicted in Figure G.2. Hyperparameter settings for adversarial attacks are listed in Table G.1. Due to the heavy load of running all these evaluations, they are performed on several clusters. These clusters are comprised of either (i) compute nodes with Nvidia A100 GPUs, 512 GB RAM, and Intel Xeon IceLake-SP processors, (ii) compute nodes with NVIDIA Quadro RTX 8000 GPUs, 1024 GB RAM, and AMD EPYC 7502P processors, (iii) NVIDIA Tesla A100 GPUs, 2048 GB RAM, Intel Xeon Platinum 8360Y processors, and (iv) NVIDIA Tesla A40 GPUs, 2048 GB RAM, Intel Xeon Platinum 8360Y processors.

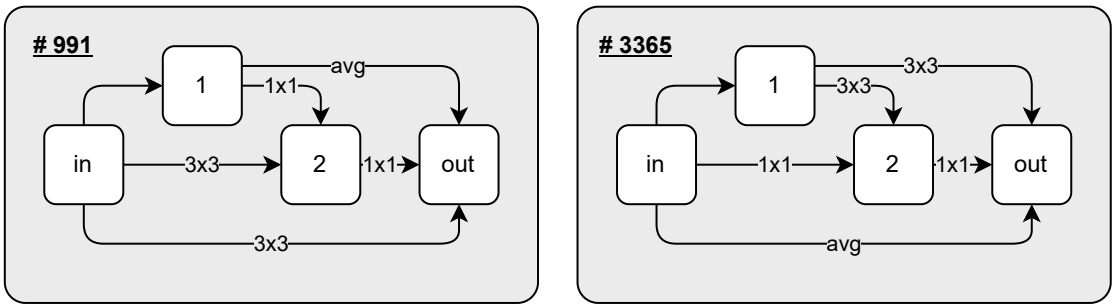


Figure G.1: Example of two isomorphic graphs in NAS-Bench-201. Due to the skip connection from node `in` to node `1`, both computational graphs are equivalent, but their identification in the search space is different. For this dataset, we evaluated all non-isomorphic graphs (`#991` was evaluated and `#3365` was not).

Table G.1: Hyperparameter settings of adversarial attacks evaluated.

Attack	**Hyperparameters**
FGSM	$\epsilon \in \{.1, .5., 1, 2, 3, 4, 5, 6, 7, 8, 255\}/255$
PGD	$\epsilon \in \{.1, .5., 1, 2, 3, 4, 8, 255\}/255$ $\alpha = 0.01/0.3$ 40 attack iterations
APGD	$\epsilon \in \{.1, .5., 1, 2, 3, 4, 8, 255\}/255$ 100 attack iterations
Square	$\epsilon \in \{.1, .5., 1, 2, 3, 4, 8, 255\}/255$ 5 000 search iterations

Algorithm 4: Robustness Dataset Gathering.

Input: (i) Architecture space A (NAS-Bench-201).
Input: (ii) Test datasets D (CIFAR-10, CIFAR-100, ImageNet16-120).
Input: (iii) Set of attacks and/or corruptions C.
Input: (iv) Robustness Dataset R.

for $a \in A$ **do**
  ▷ `Load pretrained weights for` a`.`
  a.load_weights(d)
  for $d \in D$ **do**
    for $c(\cdot, \cdot) \in C$ **do**
      ▷ `Corrupt dataset` d`.`
      $d_c \leftarrow c(a, d)$
      ▷ `Evaluate architecture` a `with` d_c`.`
      Accuracy, Confidence, ConfusionMatrix $\leftarrow eval(a, d_c)$
      ▷ `Extend robustness dataset with evaluations.`
      $R[d][c]$["accuracy"]$[a] \leftarrow$ Accuracy
      $R[d][c]$["confidence"]$[a] \leftarrow$ Confidence
      $R[d][c]$["cm"]$[a] \leftarrow$ ConfusionMatrix
    end
  end
end

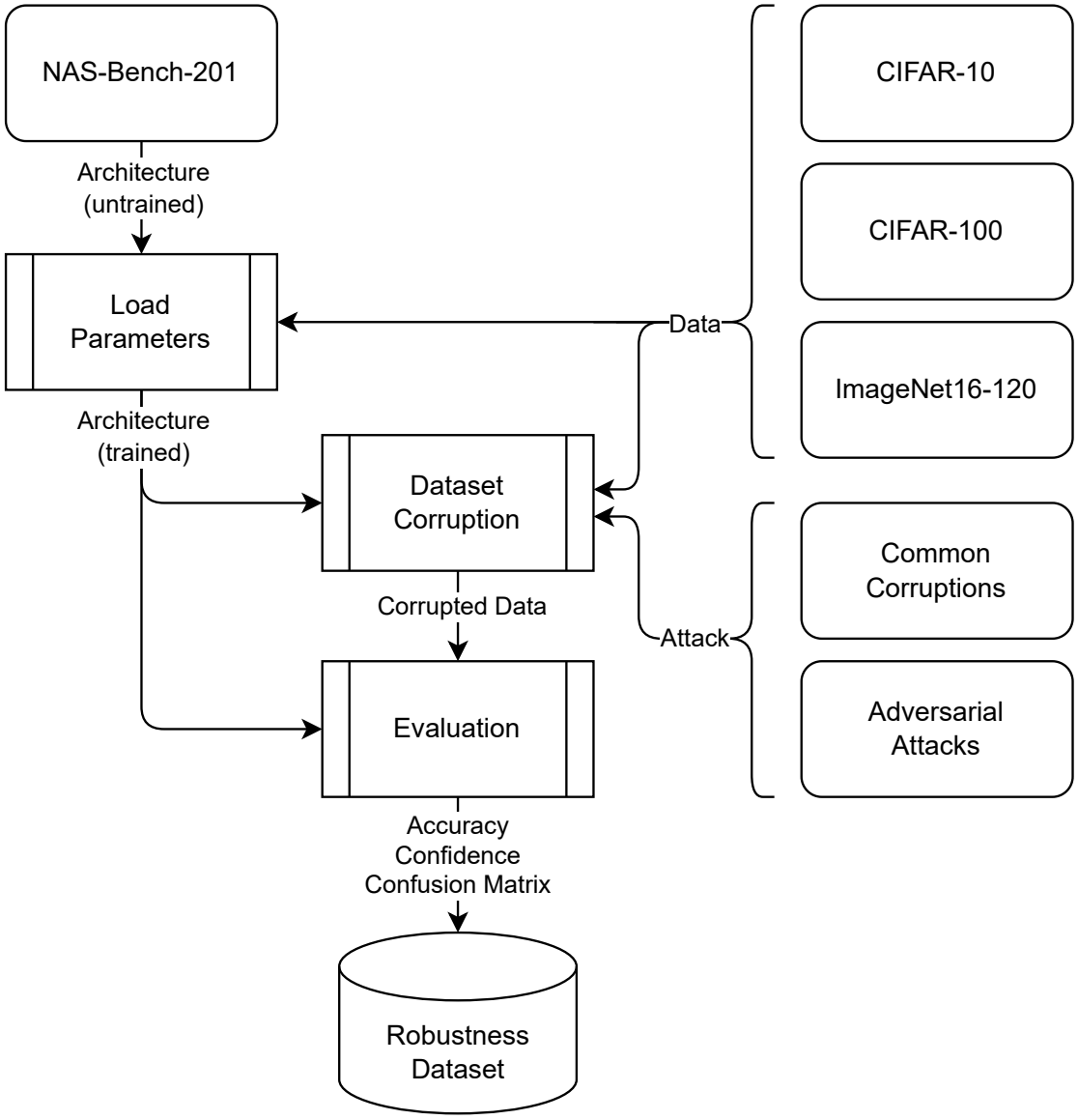


Figure G.2: Diagram showing the gathering process for our robustness dataset. (i) An non-isomorphic architecture contained in NAS-Bench-201 is created and its parameters are loaded from a provided checkpoint, dependent on the dataset evaluated. (ii) Given the evaluation dataset, an attack or corruption, and the trained network, the evaluation dataset is corrupted and (iii) the resulting corrupted data is used to evaluate the network. (iv) The evaluation results are stored in our robustness dataset.

Table G.2: Keys for attacks and corruptions evaluated.

Clean	Adversarial	Common Corruptions
clean	aa_apgd-ce	brightness
	aa_square	contrast
	fgsm	defocus_blur
	pgd	elastic_transform
		fog
		frost
		gaussian_noise
		glass_blur
		impulse_noise
		jpeg_compression
		motion_blur
		pixelate
		shot_noise
		snow
		zoom_blur

G.1.3 Dataset Structure, Distribution, and License

Files are provided in `json` format to ensure platform-independence and to reduce dependency on external libraries (e.g. Python has built-in `json`-support). We publish code that accompanies our dataset on GitHub. The dataset itself will be linked from GitHub and is hosted on an institutional cloud service. This ensures longtime availability and the possibility to version the dataset. Dataset and code are published under GNU GPLv3.

G.1.4 Structure

The dataset consists of 3 folders, one for each dataset evaluated. Each folder contains one `json` file for each combination of key and measurement. Keys refer to the sort of attack or corruption used (Table G.2 lists all keys). Measurements refer to the collected evaluation type (`accuracy`, `confidence`, `cm`). Clean and adversarial evaluations are performed on all datasets, while common corruptions are evaluated on `cifar10` and `cifar100`.

Metadata. The `meta.json` file contains information about each architecture in NAS-Bench-201. This includes, for each architecture identifier, the corresponding string defining the network design (as per [DY20]) as well as the identifier of the corresponding non-isomorphic architecture from [DY20] that we evaluated. The file also contains all ϵ values that we evaluated for each adversarial attack. An excerpt of this file is shown in Figure G.3.

Files. All files are named according to `"{key}_{measurement}.json"`. An exemplary excerpt of such file is shown in Figure G.4. Each file contains nested dictionaries stating the dataset, evaluation key and measurement type. For evaluations with multiple measurements, e.g. in the case of adversarial attacks for multiple ϵ values, the results are concatenated into a list. Files and their possible contents are described in Table G.3. We showed some analysis and possible use-cases on accuracies in the main paper. In the following, we elaborate on and show `confidence` and confusion matrix (`cm`) measurements.

Table G.3: Files and their possible content.

File	Description
clean_accuracy	one accuracy value for each evaluated network
clean_confidence	one confidence matrix for each evaluated network and collection scheme
clean_cm	one confusion matrix for each evaluated network
{attack}_accuracy	list of accuracies, where each element corresponds to the respective ϵ value
{attack}_confidence	list of confidence matrices, where each element corresponds to the respective ϵ value
{attack}_cm	list of confusion matrices, where each element corresponds to the respective ϵ value
{corruption}_accuracy	list of accuracies, where each element corresponds to the respective corruption severity
{corruption}_confidence	list of confidence matrices, where each element corresponds to the respective corruption severity
{corruption}_cm	list of confusion matrices, where each element corresponds to the respective corruption severity

```
{
  "ids": {
    ...,
    "21": {
      "nb201-string": "|nor_conv_1x1~0|+|none~0|none~1|+|nor_conv_1x1~0|nor_conv_3x3~1|none~2|",
      "isomorph": "21"
    },
    ...,
    "1832": {
      "nb201-string": "|nor_conv_1x1~0|+|nor_conv_1x1~0|none~1|+|nor_conv_1x1~0|skip_connect~1|none~2|",
      "isomorph": "309"
    },
    ...
  },
  "epsilons": {
    "aa_apgd-ce": [0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 8.0],
    "aa_square": [0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 8.0],
    "fgsm": [0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 255.0],
    "pgd": [0.1, 0.5, 1.0, 2.0, 3.0, 4.0, 8.0]
  }
}
```


Figure G.3: Excerpt of `meta.json` showing meta information of architectures #21 and #1832, as well as ϵ values for each attack. Architecture #21 is non-isomorphic and points to itself, while architecture #1832 is an isomorphic instance of #309.


```
{
  "cifar10": {
    "clean": {
      "accuracy": {
        "0": 0.856,
        ...
    }
  }
}
```

```
{
  "cifar10": {
    "pgd": {
      "accuracy": {
        "0": [0.812, 0.582, 0.295, 0.034, 0.002, 0.0, 0.0],
        ...
    }
  }
}
```


Figure G.4: Excerpt of files containing results for **(left)** `clean_accuracy.json` and **(right)** `pgd_accuracy.json` for dataset `cifar10` for the architecture #0. Numbers are rounded to improve readability.


```
{
  "cifar10": {
    "clean": {
      "confidence": {
        "0": {
          "label": [[...]],
          "argmax": [[...]],
          "prediction": [...]
        }
    }
  }
}
```


Figure G.5: Excerpt of `clean_confidence.json` for `cifar10`. Numbers are not shown to improve readability.

G.1.5 Confidence

We collect the mean confidence after softmax for each network over the whole (attacked) test dataset evaluated. We used 3 schemes to collect confidences (see Figure G.6). First, confidences for each class are given by true labels (called `label`). In case of `cifar10`, this results in a 10×10 confidence matrix, for `cifar100` a 100×100 confidence matrix, and `ImageNet16-120` a 120×120 confidence matrix. Second, confidences for each class are given by the class predicted by the network (called `argmax`). This again results in matrices of sizes as mentioned. Third, confidences for correctly classified images as well as confidences for incorrectly classified images (called `prediction`). For all image datasets, this results in a vector with 2 dimensions. Each result is saved as a list (or list of list), see Figure G.5.

Figure G.7 shows a progression of `label` confidence values for class label 0 on `cifar10` from `clean` to `fgsm` with increasing values of ϵ. Figure G.8 shows how `prediction` confidences of correctly and incorrectly classified images correlate with increasing values of ϵ when attacked with `fgsm`.

G.1.6 Confusion Matrix

For each evaluated network, we collect the confusion matrix (key: `cm`) for the corresponding (attacked) test dataset. The result is a 10×10 matrix in case of `cifar10`, a 100×100 matrix in case of `cifar100`, and a 120×120 matrix in case of `ImageNet16-120`. See Figure G.9 for an example, where we summed up confusion matrices for all networks on `cifar10`.

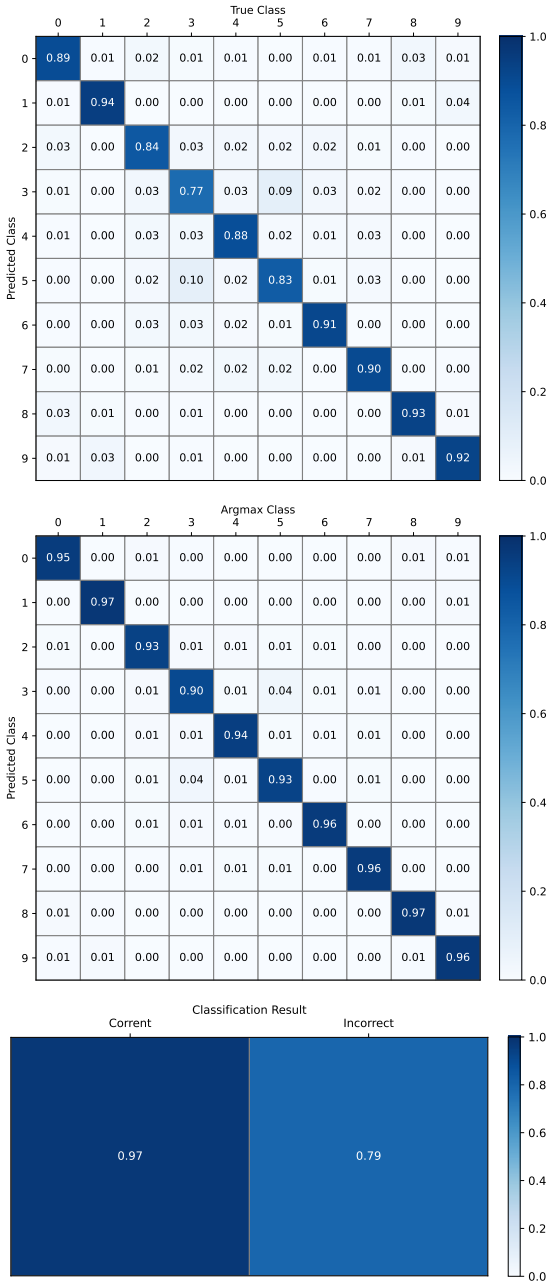


Figure G.6: Mean confidence scores on clean CIFAR-10 images for all non-isomorphic networks in NAS-Bench-201. (**top**: `label`) For each true class label. (**middle**: `argmax`) For each predicted class label. (**bottom**: `prediction`) For correct and incorrect classifications.

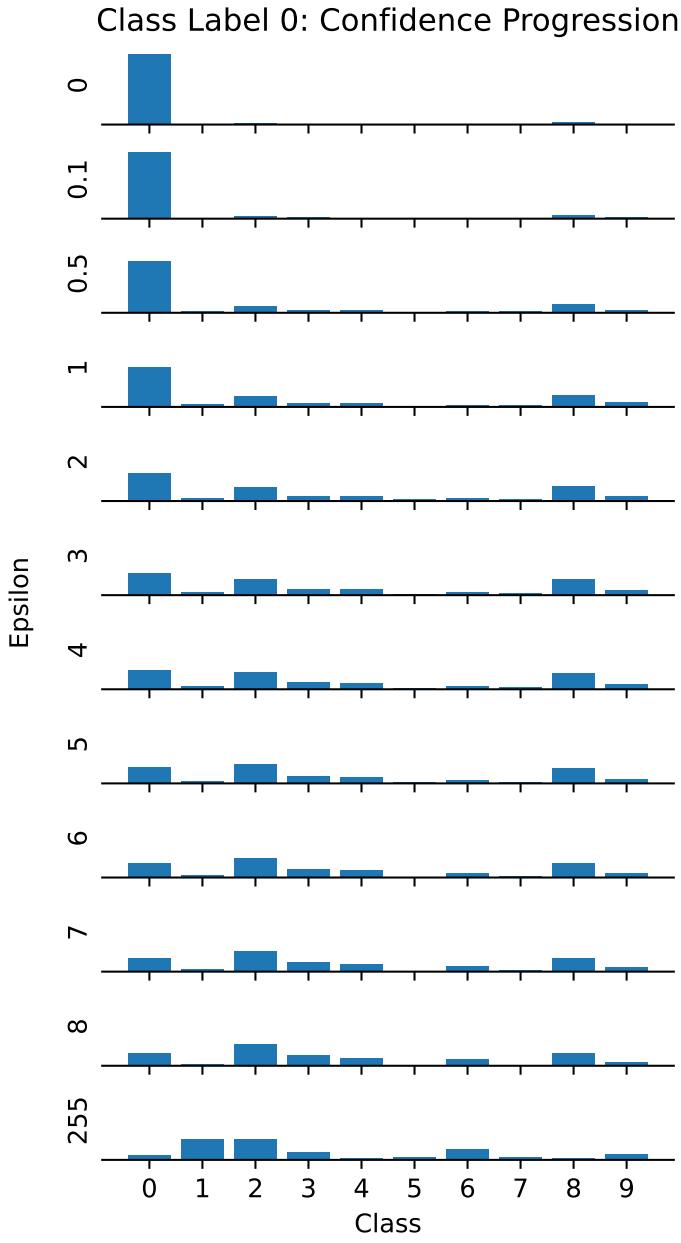


Figure G.7: Mean `label` confidence scores on FGSM-attacked CIFAR-10 images for different ϵ for all non-isomorphic networks in NAS-Bench-201. Only confidence scores for class label 0 are shown. Networks lose prediction confidence for the true label when ϵ increases.

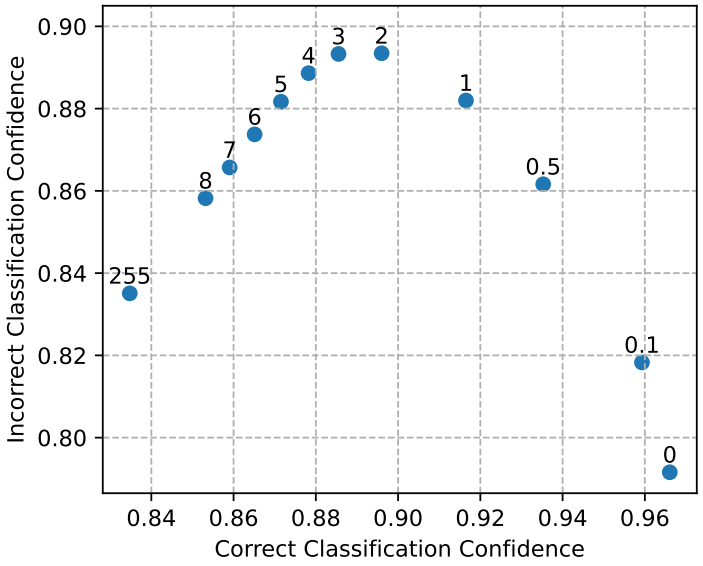


Figure G.8: Mean `prediction` confidence scores on FGSM-attacked CIFAR-10 images for different ϵ (on top of points) for all non-isomorphic networks in NAS-Bench-201. Networks become less confident in their prediction if their prediction is correct when ϵ increases. Networks become more confident in their prediction if their prediction is incorrect, however, only up to a certain ϵ value. When ϵ further increases, confidence drops again.

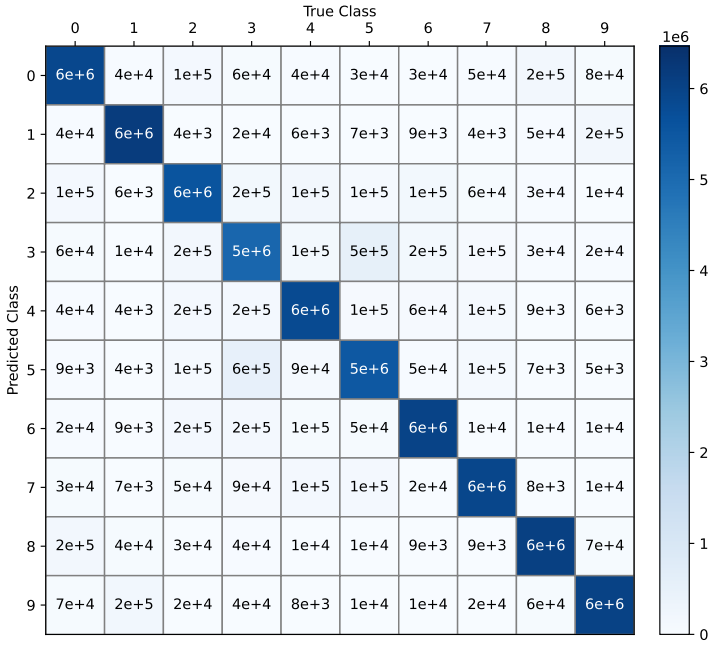


Figure G.9: Aggregated confusion matrices on clean CIFAR-10 images for all non-isomorphic networks in NAS-Bench-201.

G.2 Correlations between Image Datasets

In Figure G.10 we show the correlation between all clean and adversarial accuracies over all datasets collected. This plot shows a positive correlation between the image datasets for the one-step FGSM attack, whereas for all other multi-step attacks, the correlation becomes close to zero or even negative.

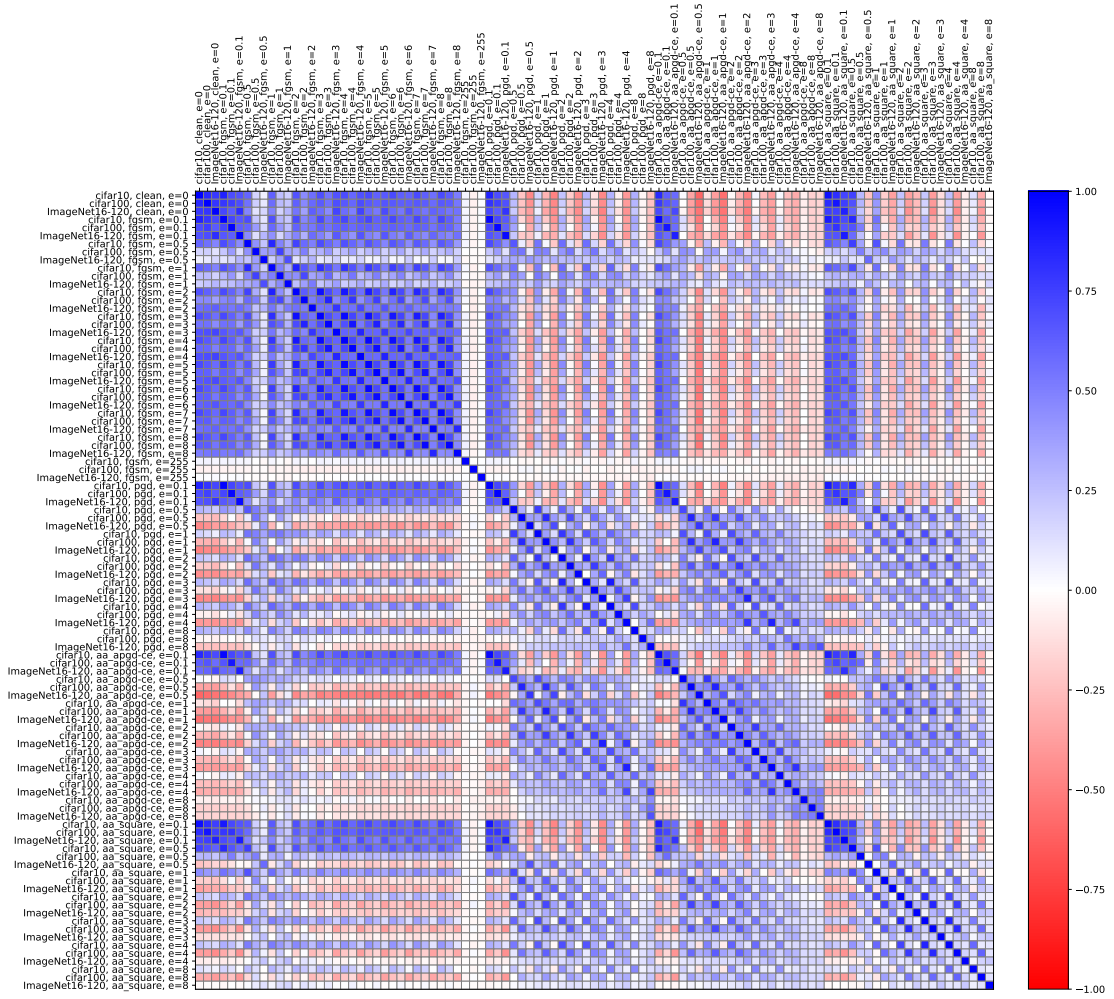


Figure G.10: Kendall rank correlation coefficient between all clean and adversarial accuracies that are evaluated in our dataset.

G.3 Example image of corruptions in CIFAR-10-C

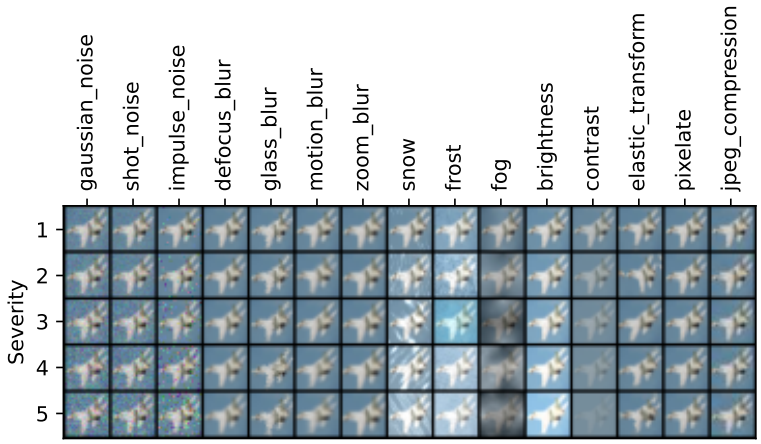


Figure G.11: An example image of CIFAR-10-C [HD19] with different corruption types at different severity levels. CIFAR-100-C [HD19] consists of images with the same corruption types and severity levels.

G.4 Main Paper Figures for other Image Datasets

CIFAR-100 Adversarial Attack Accuracies (Figure 9.2).

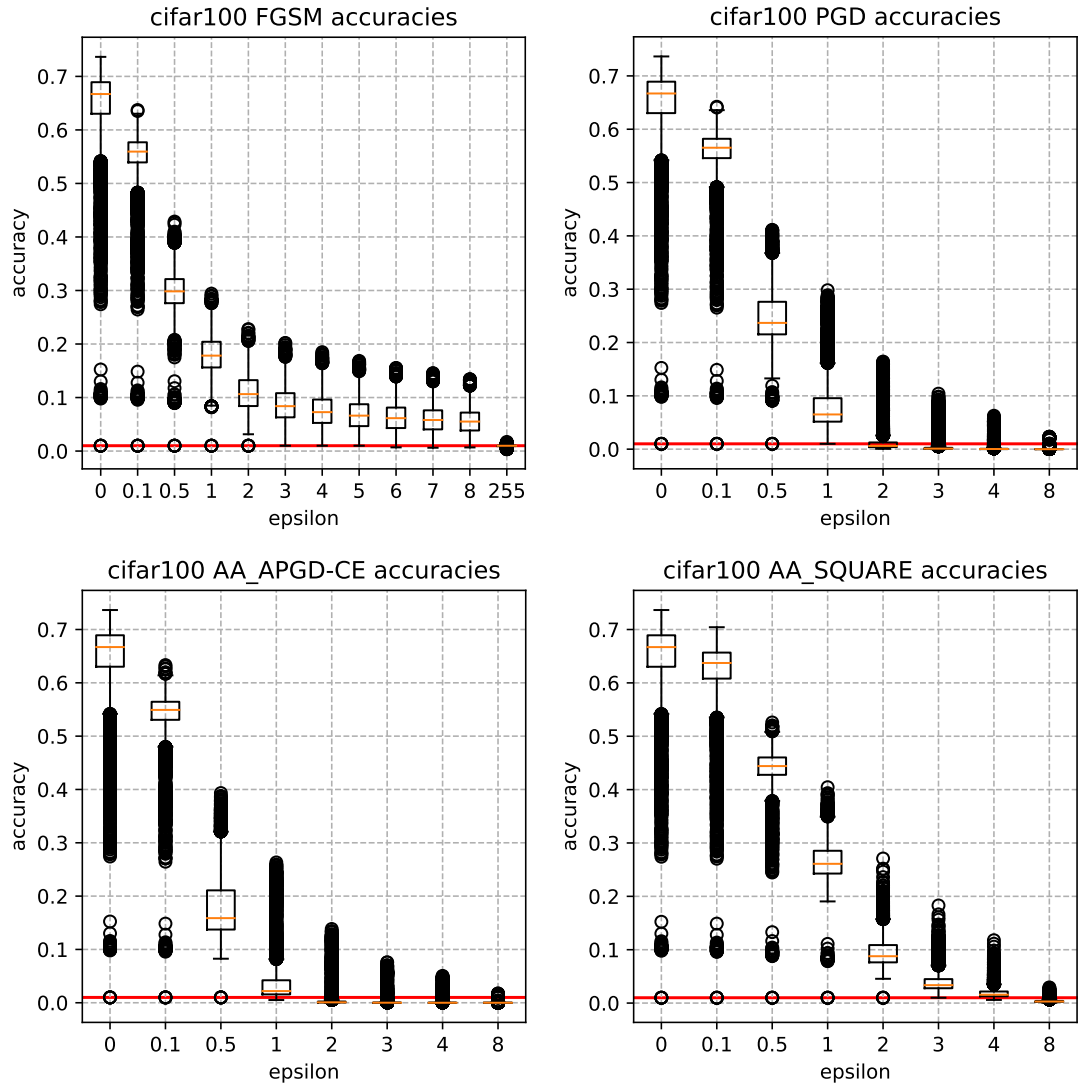


Figure G.12: Accuracy boxplots over all unique architectures in NAS-Bench-201 for different adversarial attacks (FGSM [GSS15], PGD [KGB17], APGD [CH20], Square [And+20]) and perturbation magnitude values ϵ, evaluated on CIFAR-100. Red line corresponds to guessing.

ImageNet16-120 Adversarial Attack Accuracies (Figure 9.2).

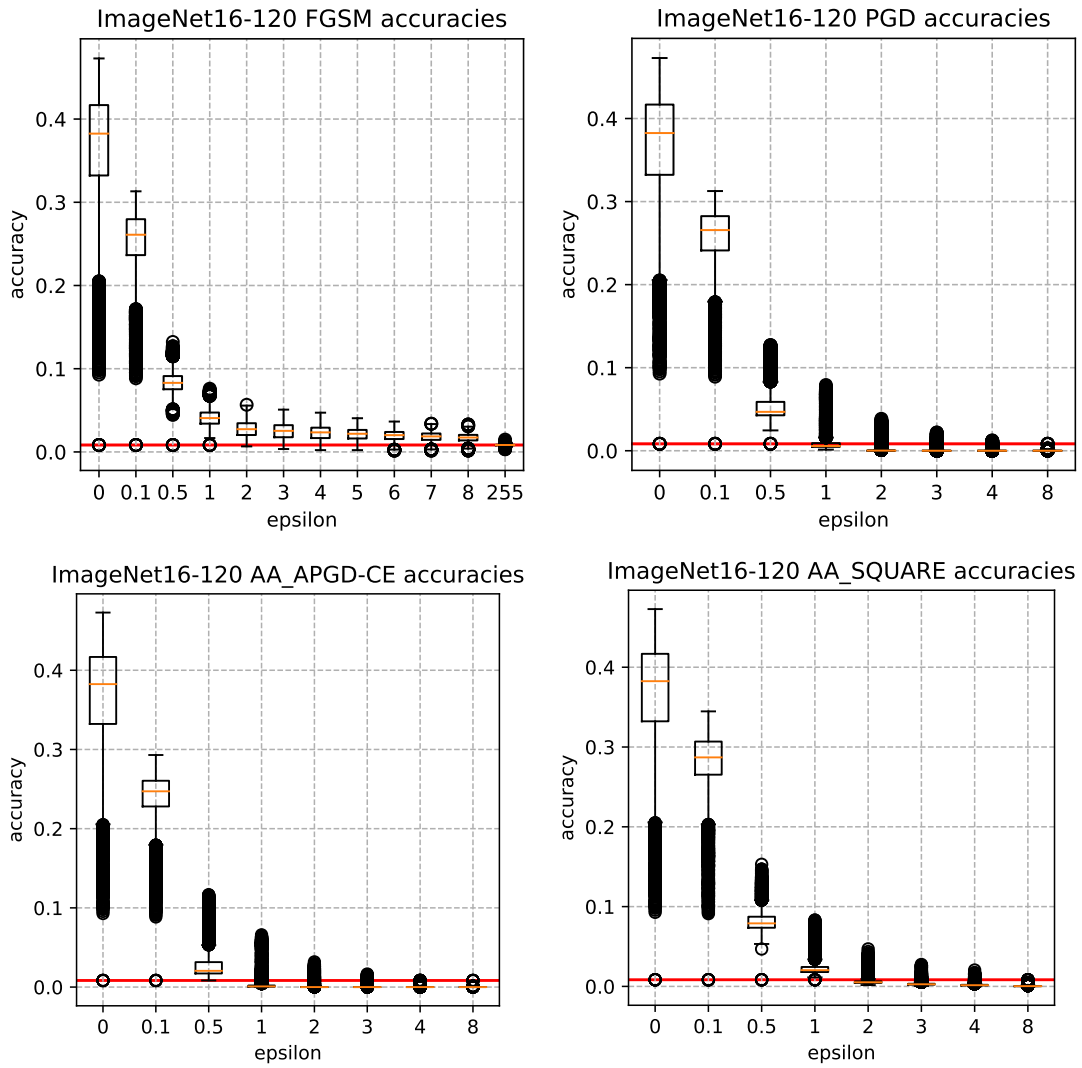


Figure G.13: Accuracy boxplots over all unique architectures in NAS-Bench-201 for different adversarial attacks (FGSM [GSS15], PGD [KGB17], APGD [CH20], Square [And+20]) and perturbation magnitude values ϵ, evaluated on ImageNet16-120. Red line corresponds to guessing.

CIFAR-10-C Common Corruption Accuracies (Figure 9.4).

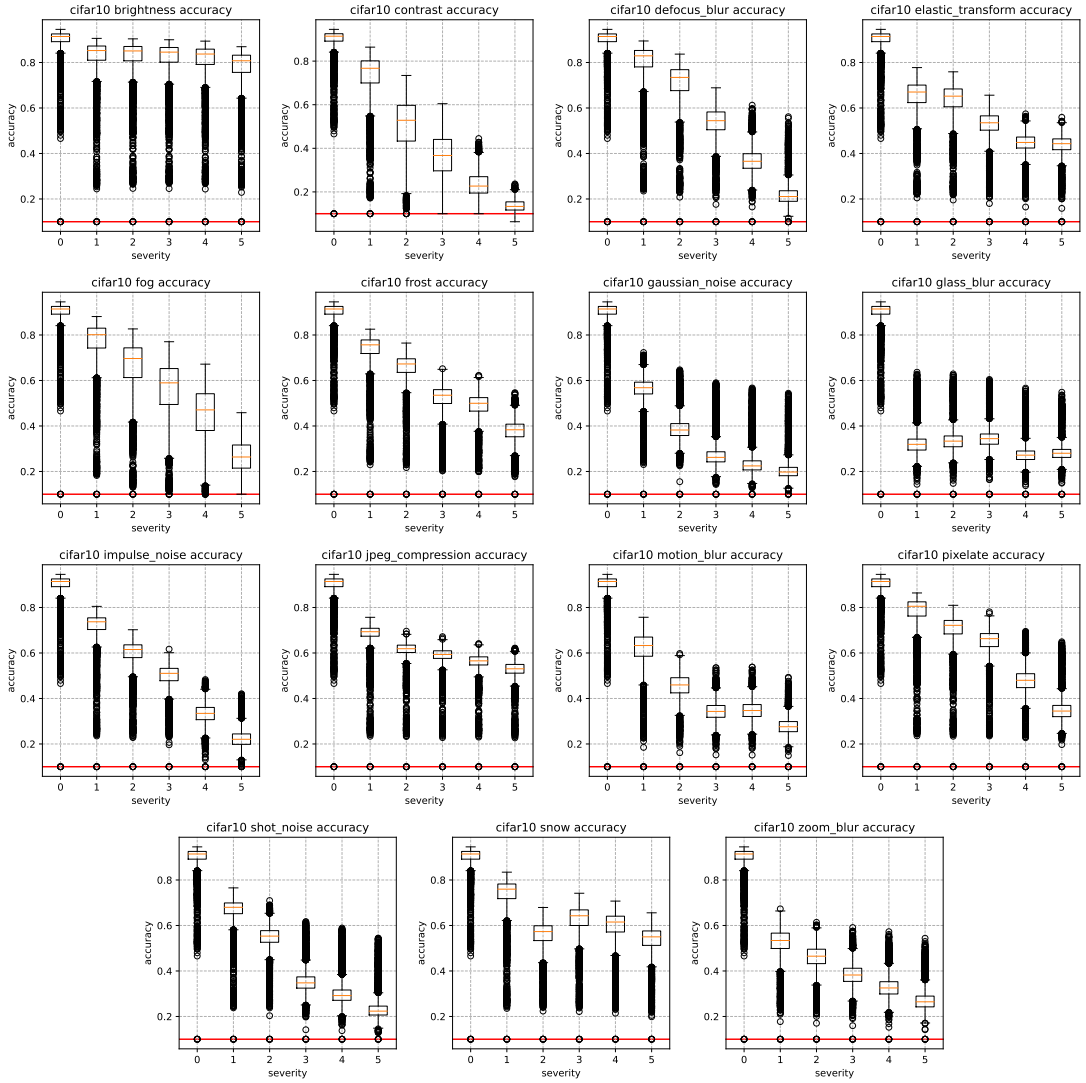


Figure G.14: Accuracy boxplots over all unique architectures in NAS-Bench-201 for different corruption types at different severity levels, evaluated on CIFAR-10-C. Red line corresponds to guessing.

CIFAR-100-C Common Corruption Accuracies (Figure 9.4).

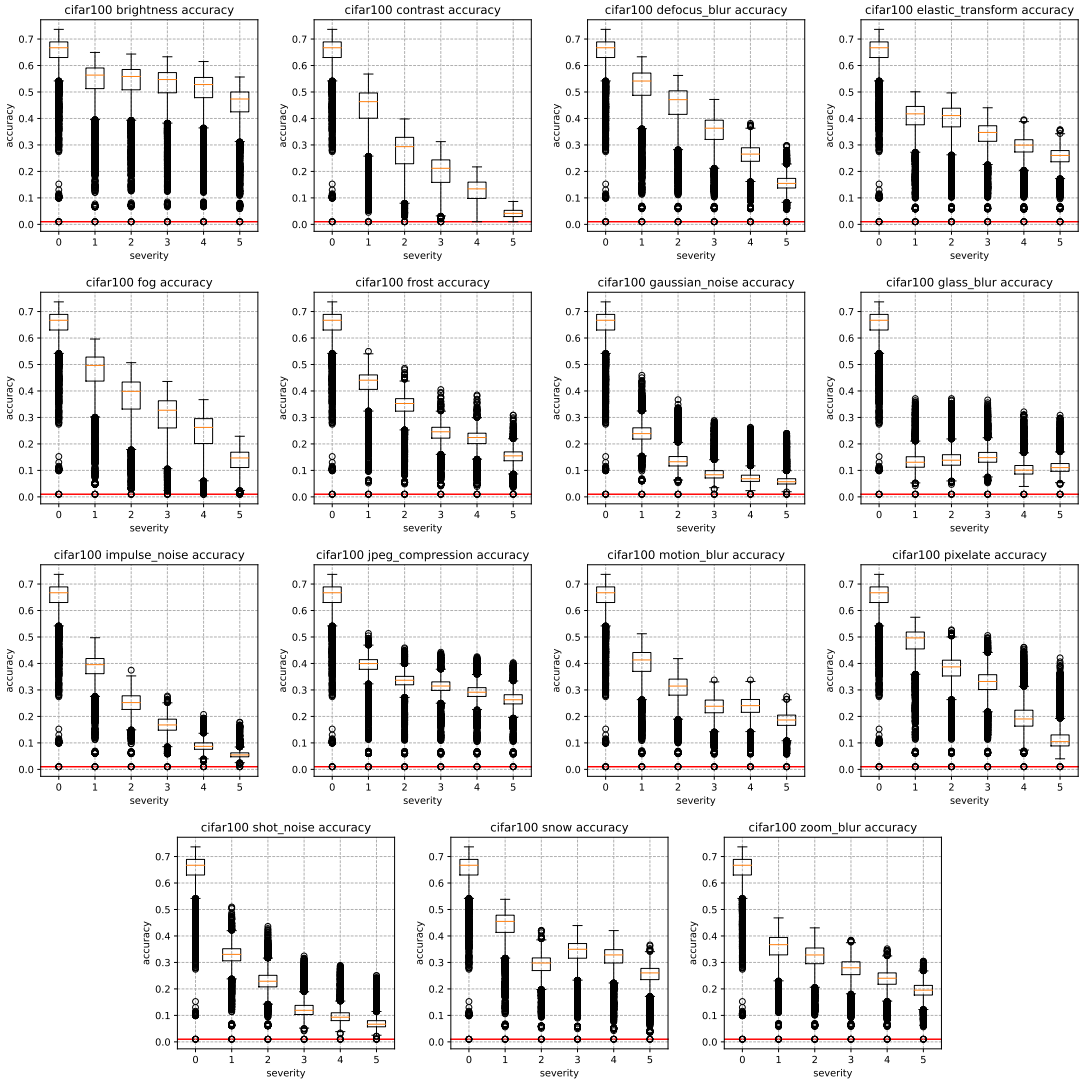


Figure G.15: Accuracy boxplots over all unique architectures in NAS-Bench-201 for different corruption types at different severity levels, evaluated on CIFAR-100-C. Red line corresponds to guessing.

CIFAR-100 Adversarial Attack Correlations (Figure 9.3).

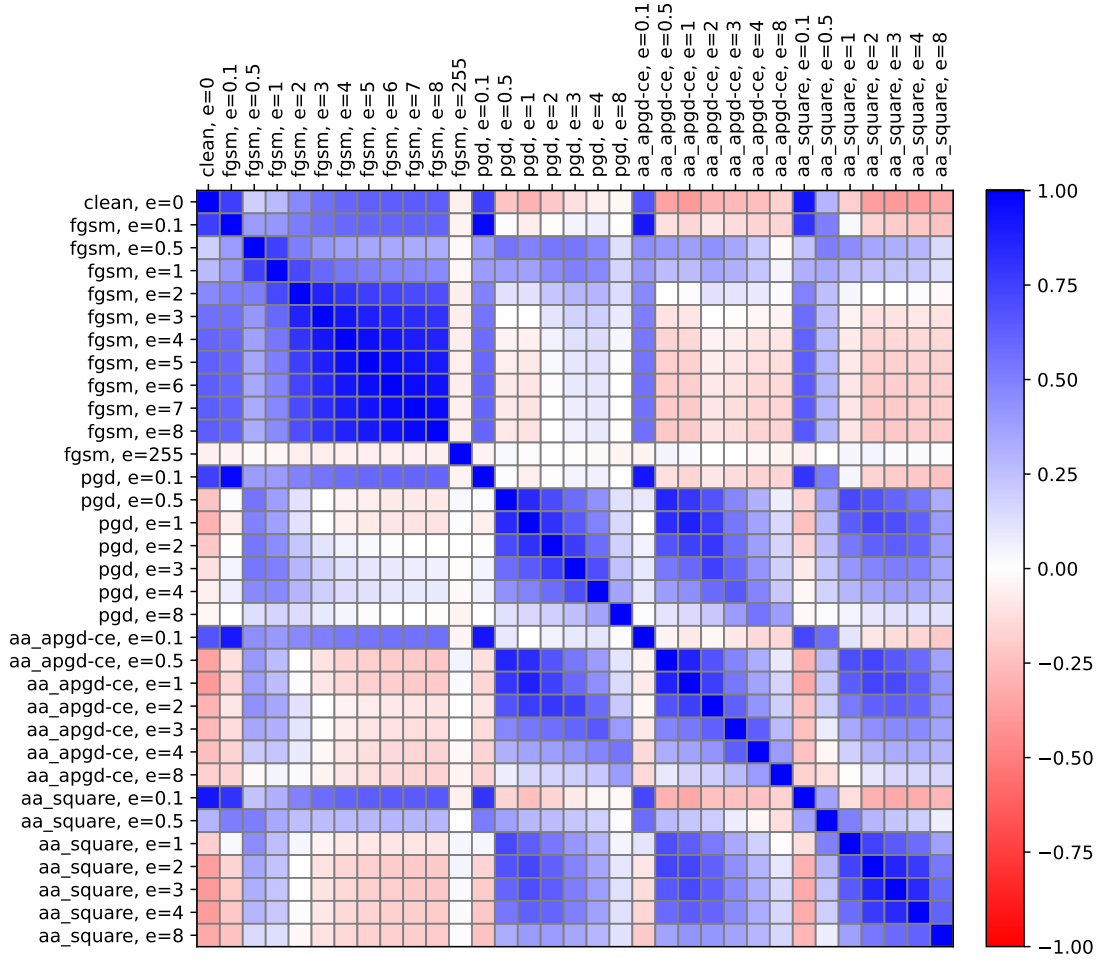


Figure G.16: Kendall rank correlation coefficient between clean accuracies and robust accuracies on different attacks and magnitude values ϵ on CIFAR-100 for all unique architectures in NAS-Bench-201.

ImageNet16-120 Adversarial Attack Correlations (Figure 9.3).

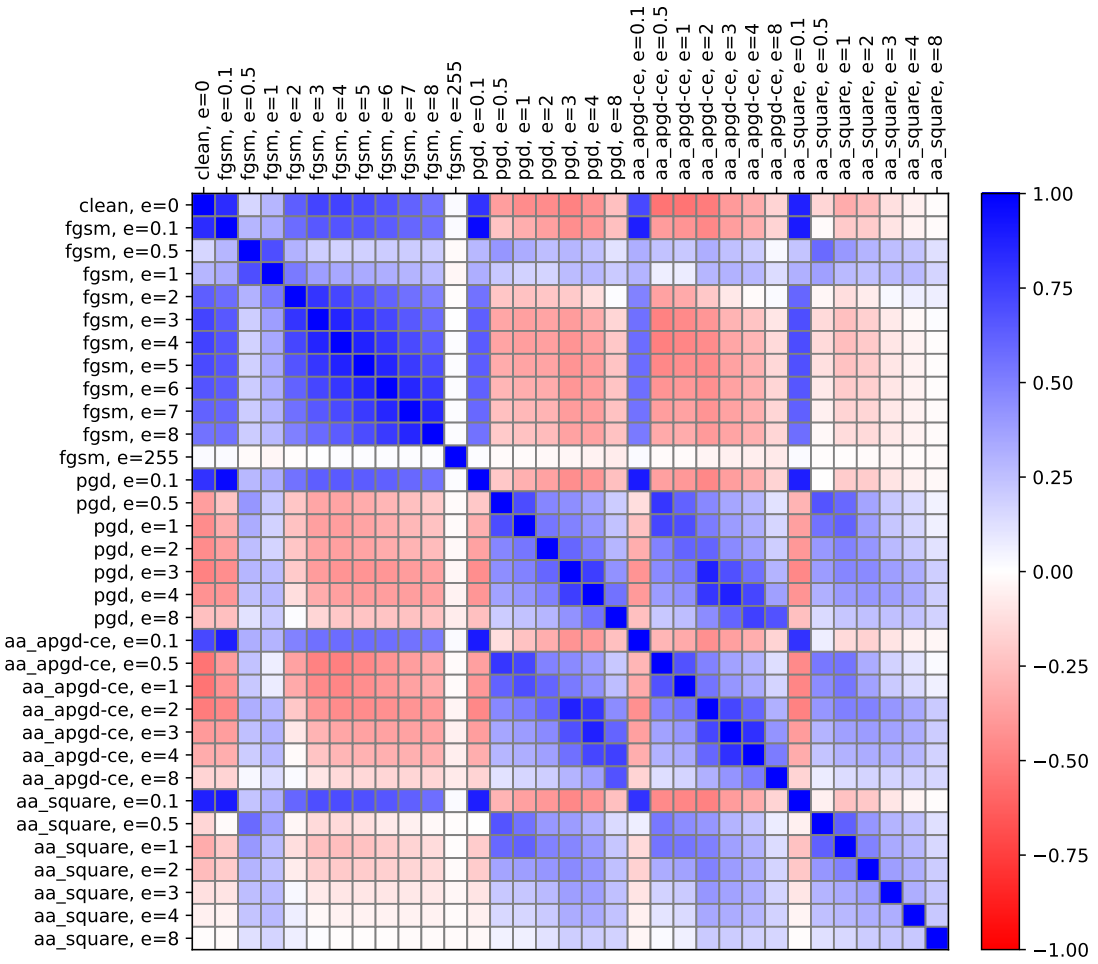


Figure G.17: Kendall rank correlation coefficient between clean accuracies and robust accuracies on different attacks and magnitude values ϵ on ImageNet16-120 for all unique architectures in NAS-Bench-201.

CIFAR-100-C Common Corruption Correlations (Figure 9.5).

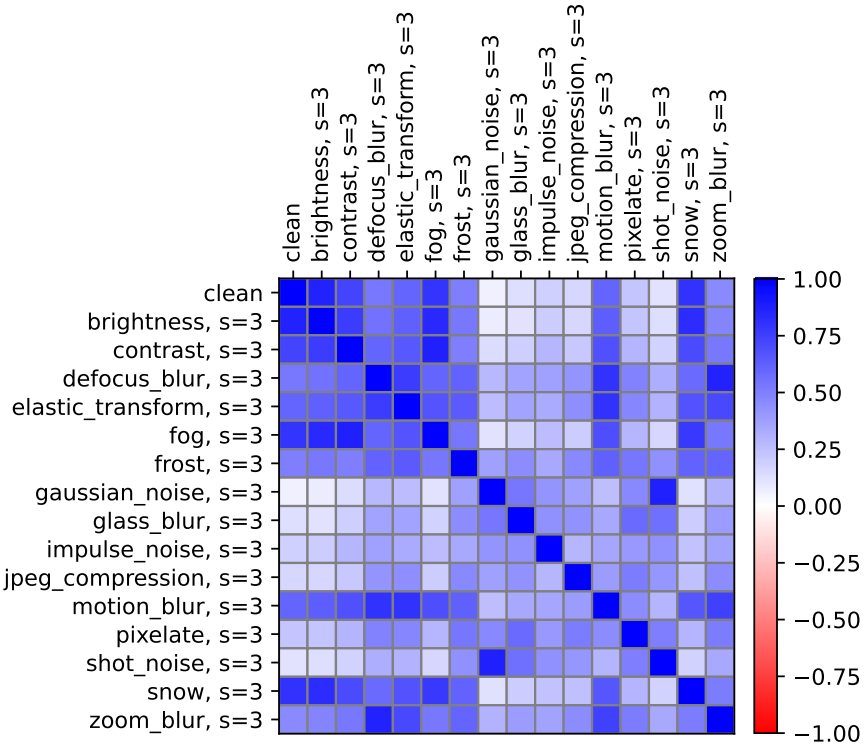


Figure G.18: Kendall rank correlation coefficient between clean accuracies and accuracies on different corruptions at severity level 4 on CIFAR-100-C for all unique architectures in NAS-Bench-201.

G.5 Analysis of Architectural Choices

In this section, we first depict the best architectures in NAS-Bench-201 [DY20] in Subsection G.5.1, then show the effect of parameter count on robustness and the magnitude of potential gains in robustness in a limited parameter count setting in Subsection G.5.2, and lastly show the effect of single changes to the best performing architecture according to clean accuracy in Subsection G.5.3.

G.5.1 Best Architectures

Figure G.19 visualizes the best architectures in the NAS-Bench-201 [DY20] search space in terms of clean accuracy, mean adversarial accuracy, and mean common corruption accuracy on CIFAR-10 and their respective edit distances. The edit distance is defined by the number of changes, either node or edge, to change the graph to the target graph. In the case of NAS-Bench-201 architectures, an edit distance of 1 means that exactly one operation differs between two architectures. So in order to modify the best performing architecture in terms of clean accuracy (#13714) into the best performing architecture according to mean corruption accuracy (#3456), we need to exchange two (out of six) operations: (i) exchange operation 2 from 3×3 convolution to zero and (ii) exchange operation 5 from 1×1 convolution to 3×3 convolution.

G.5.2 Cell Kernel Parameter Count

Figure G.20 displays the mean adversarial robustness accuracies (left) and the mean corruption robustness accuracies (right) against the clean accuracy, color-coded by the number of cell kernel parameters. We count 1 for each 1×1 convolution and 9 for each 3×3 convolution contained in the cell, hence, their number ranges in $[0, 54]$. Since these are multipliers for the parameter count of the whole network, we coin these *cell kernel parameters*. Overall, we can see that the cell kernel parameter count matters in terms of robustness, hence, that networks with large parameter counts are more robust in general. We can also see that the number of cell kernel parameters are more essential for robustness against common corruptions, where the correlation between clean and corruption accuracy is more linear. Also in terms of adversarial robustness, there seems to be a large magnitude of possible improvements that can be gained by optimizing architecture design.

Limited Cell Parameter Count. To further investigate the magnitude of possible improvements via architectural design optimization, we look into the scenario of limited cell parameter count.

In Figure G.21, we depict all unique architectures in NAS-Bench-201 by their mean adversarial robustness and cell kernel parameter count. Networks with parameter count 18 (408 instances in total) are highlighted in orange. As we can see, there is a large range of mean adversarial accuracies $[0.21, 0.4]$ for the parameter count 18 showing the potential of doubling the robustness of a network with *the same parameter count* by carefully crafting its topology. In Figure G.22 we show the top-20 performing architectures (color-coded, one operation for each edge) in the mentioned scenario of a parameter count of 18, according to (**top**) mean adversarial and (**bottom**) mean corruption accuracy. It is interesting to see that in both cases, there are (almost) no convolutions on edges 2 and 4, and additionally no dropping or skipping of edge 1. In the case of edge 4, it seems that a single convolution layer connecting input and output of the cell increases sensitivity of the network. Hence, most of the top-20 robust architectures stack convolutions (via edge 1, followed by either edge 3 or 5), from which we hypothesize that stacking convolutions operations might improve robustness when designing architectures. At

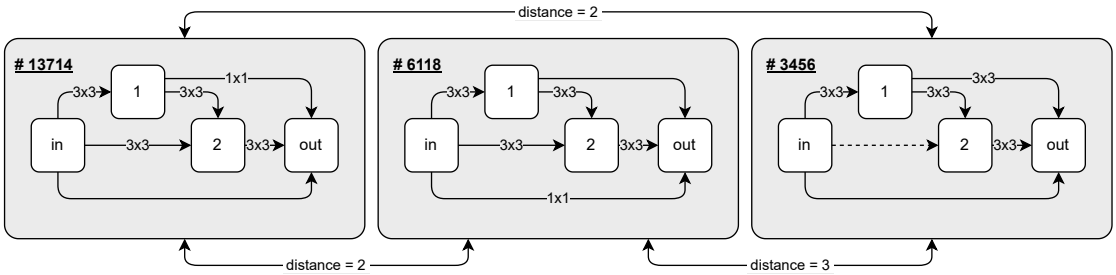


Figure G.19: Best architectures in NAS-Bench-201 according to (**left**) clean accuracy, (**middle**) mean adversarial accuracy (over all attacks and ϵ values as described in Subsection 9.3.2), and (**right**) mean common corruption accuracy (over all corruptions and severities) on CIFAR-10. See Figure 9.1 for cell connectivity and operations.

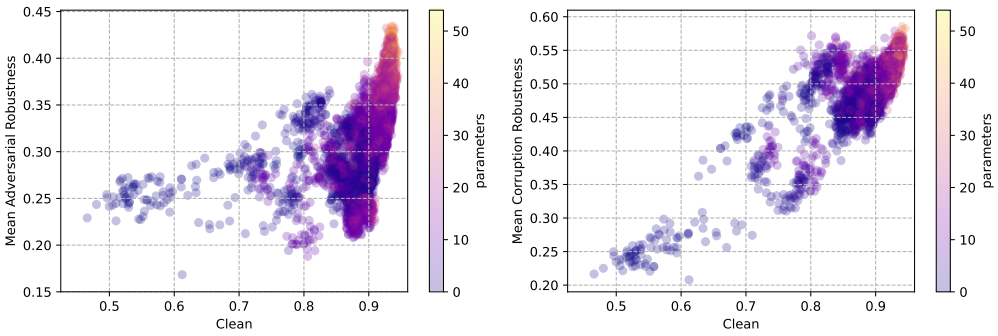


Figure G.20: (**left**) Mean adversarial robustness accuracies and (**right**) mean corruption robustness accuracies vs. clean accuracies on CIFAR-10 for all unique architectures in NAS-Bench-201. Scatter points are colored based on the number of kernel parameters of a single cell (1 for each 1×1 convolution, 9 for each 3×3 convolution).

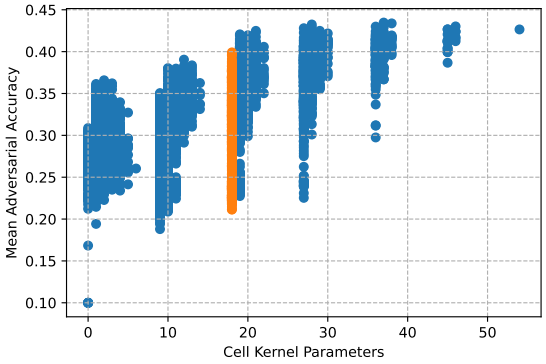


Figure G.21: Mean robust accuracy over all attacks as described in Subsection 9.3.2 on CIFAR-10 by kernel parameters $\in [0, 54]$ for all unique architectures in NAS-Bench-201. Orange scatter points depict all architectures with kernel parameter count 18, hence, architectures with exactly 2 times 3×3 convolutions. Although having exactly the same parameter count, the mean adversarial robustness of these networks ranges in $[0.21, 0.40]$.

the same time, skipping input to output via edge 4 seems not to affect robustness negatively, as long as the input feature map is combined with stacked convolutions. Important to note here is that this is a first observation, which can be made by using our provided dataset. This observation functions as a motivation for how this dataset can be used to analyze robustness in combination with architecture design.

G.5.3 Gains and Losses by Single Changes

The fact that our dataset contains evaluations for all unique architectures in NAS-Bench-201 enables us to analyze the effect of small architectural changes. In Figure G.23, we depict again all unique architectures by their clean and robust accuracies on CIFAR-10 [Kri09]. The red data point in both plots shows the best performing architecture in terms of clean accuracy (#13714, see Figure G.19), while the orange points are its neighboring architectures with edit distance 1. The operation changed for each point is shown in the legend. As we can see in the case of adversarial attacks, we can trade-off more robust accuracy for less clean accuracy by changing only one operation. While some changes seem obvious (adding more parameters as with 13 and 14), it is interesting to see that exchanging the 3×3 convolution on edge 3 with average pooling (and hence, reducing the amount of parameters) also improves adversarial robustness. In terms of robustness towards common corruptions, each architectural change leads to worse clean and robust accuracy in this case. Changing more than one operation is necessary to improve common corruption accuracy of this network (as we have seen in Figure G.19).

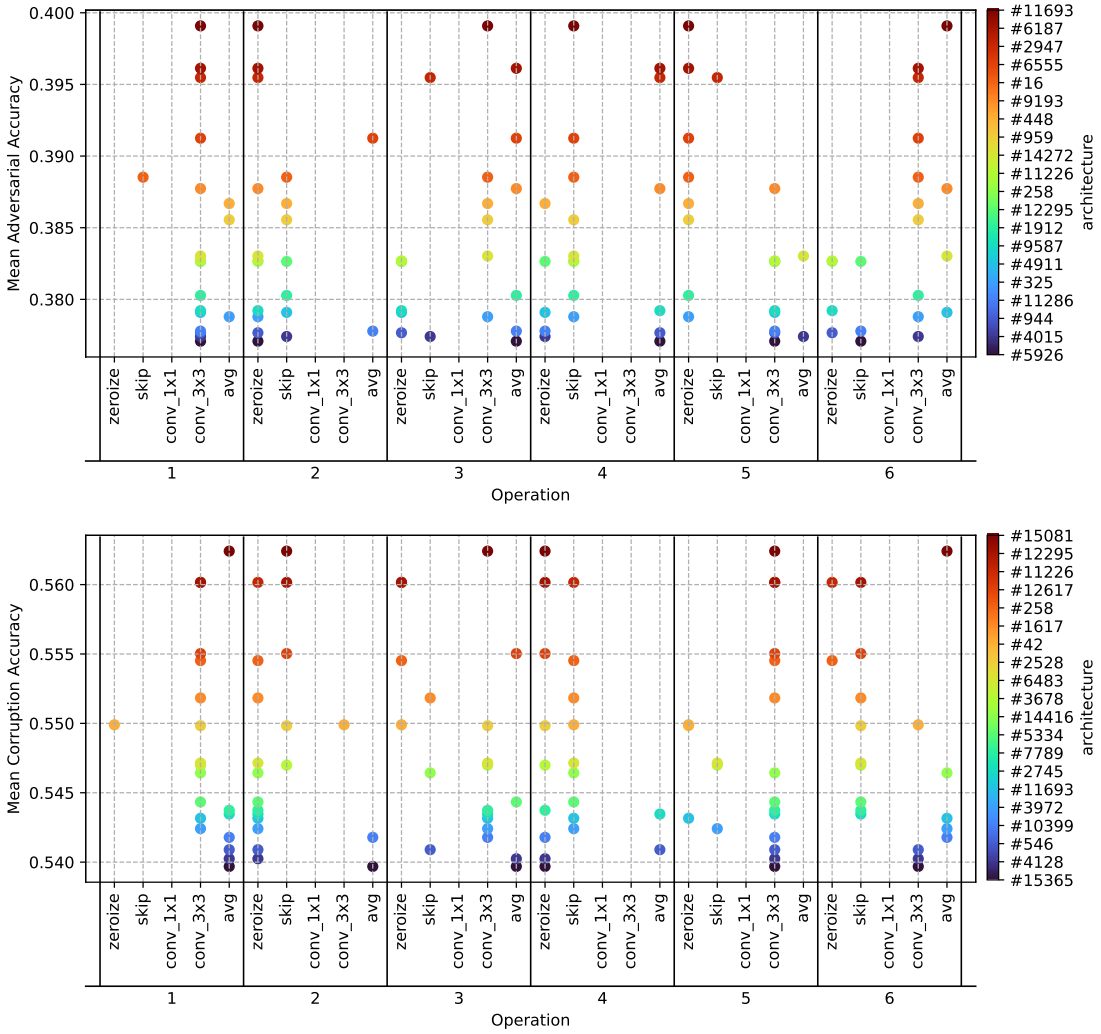


Figure G.22: Top-20 architectures with cell kernel parameter count 18 (hence, architectures with exactly 2 times 3×3 convolutions) according to (**top**) mean adversarial accuracy and (**bottom**) mean corruption accuracy on CIFAR-10. See Figure 9.1 for cell connectivity and operations (1-6).

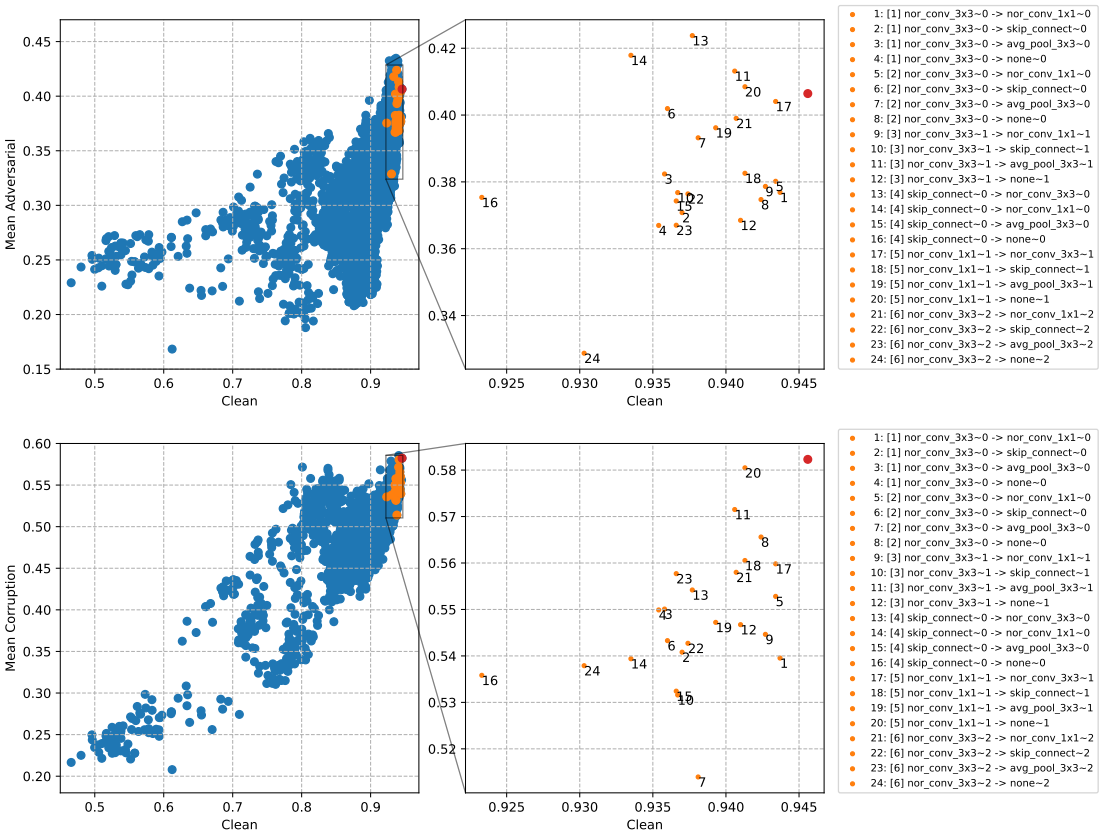


Figure G.23: (**top**) Scatter plot clean accuracy vs. mean adversarial accuracy (over all attacks and ϵ values as described in Subsection 9.3.2) on CIFAR-10. (**bottom**) Scatter plot clean accuracy vs. mean common corruption accuracy (over all corruptions and severities) on CIFAR-10. The red data point shows the best performing architecture according to clean accuracy on CIFAR-10. The orange data points are neighboring architectures, where exactly one operation differs. The change of operation is depicted in the legend. The number in brackets refers to the edge where the operation was changed. See Figure 9.1 for cell connectivity and operations (1-6).