Recognizing Hand-drawn Diagrams in Images

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

vorgelegt von

Bernhard Schäfer
aus Worms

Mannheim, 2023

Dekan: Dr. Bernd Lübcke, Universität Mannheim
Referent: Professor Dr. Heiner Stuckenschmidt, Universität Mannheim
Korreferent: Professor Dr. Henrik Leopold, Kühne Logistics University

Tag der mündlichen Prüfung: 6. Juni 2023

Abstract

Diagrams are an essential tool in any organization. They are used to create conceptual models of anything ranging from business processes to software architectures. Despite the abundance of diagram modeling tools available, the creation of conceptual models often starts by sketching on a whiteboard or paper. However, starting with a hand-drawn diagram introduces the need to eventually digitize it, so that it can be further edited in modeling tools. To reduce the effort associated with the manual digitization of diagrams, research in hand-drawn diagram recognition aims to automate this task. While there is a large body of methods for recognizing diagrams drawn on tablets, there is a notable gap for recognizing diagrams sketched on paper or whiteboard. To close this research gap, this doctoral thesis addresses the problem of recognizing hand-drawn diagrams in images. In particular, it provides the following five main contributions. First, we collect and publish a dataset of business process diagrams sketched on paper. Given that the dataset originates from conceptual modeling tasks solved by 107 participants, it has a high degree of diversity, as reflected in various drawing styles, paper types, pens, and image-capturing methods. Second, we provide an overview of the challenges in recognizing conceptual diagrams sketched on paper. We find that conceptual modeling leads to diagrams with chaotic layouts, making the recognition of edges and labels especially challenging. Third, we propose an end-to-end system for recognizing diagrams modeled with BPMN, the standard language for modeling business processes. Given an image of a hand-drawn BPMN diagram, our system produces a BPMN XML file that can be imported into process modeling tools. The system consists of an object detection neural network, which we extend with network components for recognizing edges and labels. The following two contributions are related to these components. Fourth, we present several deep learning methods for edge recognition, which recognize the drawn path and connected shapes of each arrow. Last, we describe a label recognition method that consists of three steps, one of which features a network that predicts whether a label belongs to a specific shape or edge. To demonstrate the performance of the proposed methods, we evaluate them on both our collected and the existing diagram datasets.

Zusammenfassung

Diagramme sind ein unverzichtbares Werkzeug in jedem Unternehmen. Sie werden zur konzeptionellen Modellierung von Geschäftsprozessen bis hin zu Software-Architekturen verwendet. Trotz der großen Auswahl an Software zur Modellierung von Diagrammen werden sie zunächst oft auf Whiteboard oder Papier skizziert. Zur Weiterverwendung handgezeichneter Diagramme in Modellierungssoftware müssen diese jedoch digitalisiert werden. Um den mit der manuellen Digitalisierung verbundenen Aufwand zu reduzieren, werden automatisierte Methoden zur Erkennung von handgezeichneten Diagramme entwickelt. Während es eine Vielzahl von Methoden zur Erkennung von auf Tablets gezeichneten Diagrammen gibt, existieren deutlich weniger Ansätze zur Erkennung von auf Papier oder Whiteboards skizzierten Diagrammen. Um diese Forschungslücke zu schließen, befasst sich diese Dissertation mit der Erkennung von handgezeichneten Diagrammen in Bildern, und liefert hierbei die folgenden fünf Forschungsbeiträge. Erstens veröffentlichen wir einen Datensatz von auf Papier skizzierten Geschäftsprozessdiagrammen. Der Datensatz stammt aus von 107 Personen bearbeiteten Modellierungsaufgaben. Er weist daher ein hohes Maß an Diversität auf, was sich in unterschiedlichen Modellierungsstilen, Papiertypen, Stiften und Digitalisierungsmethoden widerspiegelt. Zweitens geben wir einen Überblick über die Herausforderungen bei der Erkennung von auf Papier modellierten Diagrammen. Unter anderem stellen wir fest, dass die konzeptionelle Modellierung zu Diagrammen mit chaotischen Layouts führt, was insbesondere die Erkennung von Pfeilverbindungen und Beschriftungen erschwert. Drittens stellen wir ein System zur Erkennung von mit BPMN, der Standardsprache für die Modellierung von Geschäftsprozessen, modellierten Diagrammen vor. Basierend auf einem Bild eines handgezeichneten BPMN Diagramms erzeugt unser System eine BPMN XML Datei die in gängige Prozessmodellierungssoftware importiert werden kann. Das System besteht aus einem neuronalen Netz zur Objekterkennung, welches wir um Komponenten für die Erkennung von Pfeilverbindungen und Beschriftungen erweitern. Die nächsten beiden Forschungsbeiträge beziehen sich auf diese Komponenten. Viertens beschreiben wir mehrere Deep Learning Ansätze zur Erkennung

der Pfade und verbundenen Knoten von Pfeilverbindungen. Zuletzt stellen wir eine aus drei Schritten bestehende Methode zur Erkennung von Beschriftungen vor. Eines dieser Schritte beeinhaltet ein neuronales Netz, welches prognostiziert, ob eine Beschriftung zu einem ausgewählten Symbol oder einer Pfeilverbindung gehört. Um die Genauigkeit der vorgeschlagenen Methoden zu demonstrieren, evaluieren wir sie sowohl auf dem von uns veröffentlichten, als auch auf anderen Diagrammdatensätzen.

Contents

List of Figures

List of Tables

Acknowledgements

On these pages, I would like to express my sincere gratitude to all the people who supported me during my time as a PhD student and who made this thesis possible.

To begin with, I am truly grateful to my supervisor Prof. Dr. Heiner Stuckenschmidt, who guided me throughout the whole process. In particular, I want to thank him for helping me frame a research topic and for connecting me with other researchers. Second, I am immensely thankful to my second supervisor Prof. Dr. Henrik Leopold, and to Prof. Dr. Han van der Aa, both of which supported me early on and with whom I published my most comprehensive research results. Thank you both for the countless hours you invested in our joint research project, for constantly helping me to improve my rather mediocre writing skills, and for a great time at two BPM conferences. Third, I would like to thank Prof. Dr. Margret Keuper for her invaluable feedback on conceptual questions related to computer vision, and for her contributions to two research papers. Fourth, I would like to thank my research colleagues in the Data and Web Science Group for providing a great working atmosphere in the office, especially before the coronavirus pandemic. This includes all members of the group, with special mentions to Diana Sola, David Friede, Christian Meilicke, and Patrick Betz.

I also received a great deal of support at SAP SE, where I was employed as a PhD student. Dr. Andreas Gerber played a decisive role in making this dissertation possible, as he hired me and agreed to supervise my dissertation from SAP's side. I am immensely thankful to him for allowing me to focus on research and for always being available to discuss questions and new ideas. Shortly after I started Andreas also hired Diana Sola as a PhD student, who became a close friend over the years. Diana was always available to talk about research and other matters of life, and she always managed to cheer me up with her great sense of humor. Together with Dhaval Aarya, the three of us formed an office trio over an extended period, which has been a great mental support, especially in times of limited social interactions due to the coronavirus pandemic. I also received support from the innovation teams at SAP Signavio. I am particularly grateful to have had the chance to turn my research findings into a prototype that was showcased to several customers.

This prototype was realized with the enormous help of several colleagues, including Philipp Schumacher, Muhammad Adeel Nawaz, Timotheus Kampik, Peyman Badakhshan, Sebastian Kaim, and Diana Veit.

Last but most importantly, I want to thank my wife Katharina for her incredible support and love throughout these years. The selflessness she showed in taking care of our three children almost single-handedly during the most stressful periods of my PhD is a testament to her strength and love as a mother and wife. I could not have completed this journey without her. Thank you for being my rock and my partner in all things. I am also grateful to my parents and siblings, who encouraged me to pursue a PhD and who never stop to believe in me.

Chapter 1

Introduction

This chapter serves as an introduction to this doctoral thesis. Section 1.1 motivates the research topic that this doctoral thesis addresses. Section 1.2 describes the specifics of the diagram recognition problem that this thesis is concerned with. Section 1.3 presents the research contributions of this thesis, and Section 1.4 the publications that have been created in this context. Finally, Section 1.5 provides an overview of the remaining chapters of this thesis.

1.1 Motivation

To motivate the research topic, we first provide some background on why diagrams are ubiquitous in a large number of disciplines. Next, we discuss why diagrams are often drawn by hand, and why this introduces the need for a hand-drawn diagram recognition system. Finally, we briefly summarize existing research in diagram recognition and highlight the main research gaps that need to be addressed to turn the idea of an automated system for recognizing hand-drawn diagrams into reality.

Diagrams. Larkin and Simon's seminal cognitive science paper [49] with the title "*Why a Diagram is (Sometimes) Worth Ten Thousand Words*" provided critical insights into the benefits of diagrams. The authors found that diagrams are easier to comprehend than informationally equivalent textual representations because the information is not provided sequentially. Instead, information in diagrams is organized (in a two-dimensional layout) by location, where items of information that are likely to be processed together are often spatially co-located. A follow-up work by Cheng [16] found that diagrams are even superior to tabular representations, even though both organize information in a two-dimensional layout.

Considering these advantages, diagrams are nowadays an important tool in all business, design, and engineering disciplines [17, 24]. In the field of software en-

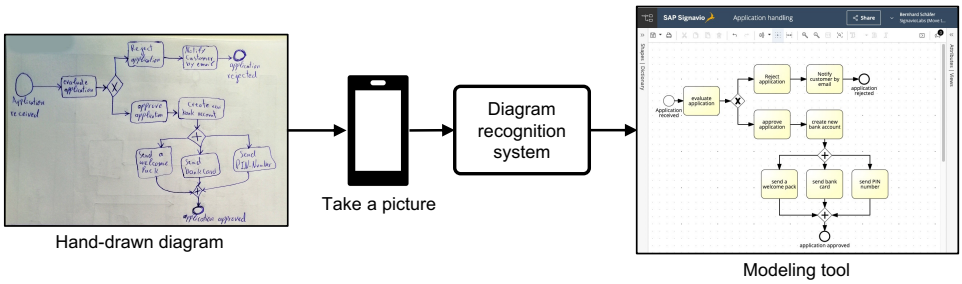


Figure 1.1: Exemplary use case for a diagram recognition system that is able to recognize hand-drawn diagrams in camera-captured images.

gineering, for instance, the design of a system is commonly modeled using the Unified Modeling Language (UML), which offers many diagram types to model the structure and behavior of software components. For modeling business processes there is also an established diagram type, the Business Process Model and Notation (BPMN), which provides a comprehensive graphical notation to model everything from simple linear processes to complex process semantics. In both domains, a comprehensive body of diagram modeling tools has been developed over the last decades, providing support for all stages of the modeling lifecycle [4, 24].

Hand-drawn diagrams. Even though there is a plethora of available modeling tools, the creation of conceptual models often starts by sketching on a whiteboard or paper [17, 24]. One reason is that digital modeling tools have been found to not effectively support activities that involve communication and collaboration with others [4]. In this regard, drawing on a whiteboard or paper has been found to have great advantages. Whiteboards are not only ubiquitous and easy to use [17], but also immediate [100]. This aspect of immediacy is of great importance since it allows people involved in the model creation to continue their thought process or conversation without interruption [100]. In addition, it has been shown that drawing a diagram by hand only takes about 10% of the time that it takes to create the same diagram with a tool like Microsoft Visio [76]. However, starting with a hand-drawn model introduces the need to eventually digitize it so that it can be further used in modeling tools. This transformation is very time-consuming if done manually, creating undesirable friction in the modeling process. To reduce the effort associated with the manual digitization of diagrams, hand-drawn diagram recognition aims to automate this transformation task.

Recognition of hand-drawn diagrams. Figure 1.1 depicts how a diagram recognition system could be used to enable a seamless transition from whiteboard to digital modeling. After sketching the diagram on a paper or whiteboard, the user takes a picture with a camera (e.g., a smartphone). The resulting image is provided

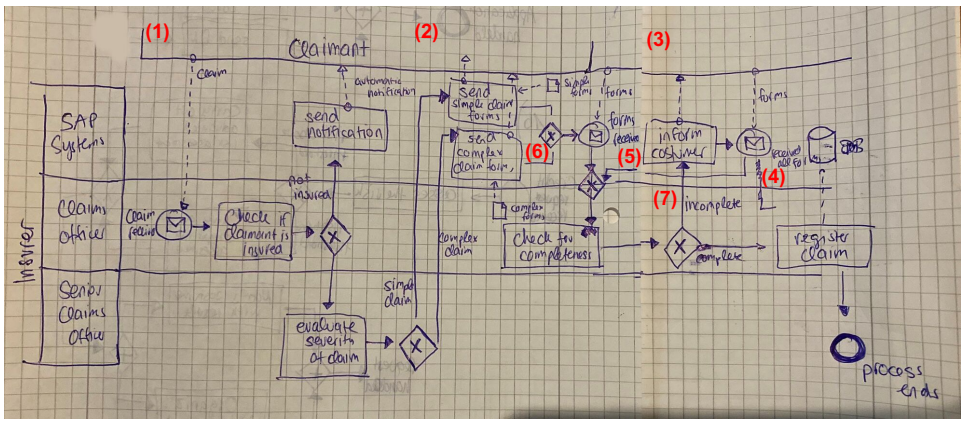
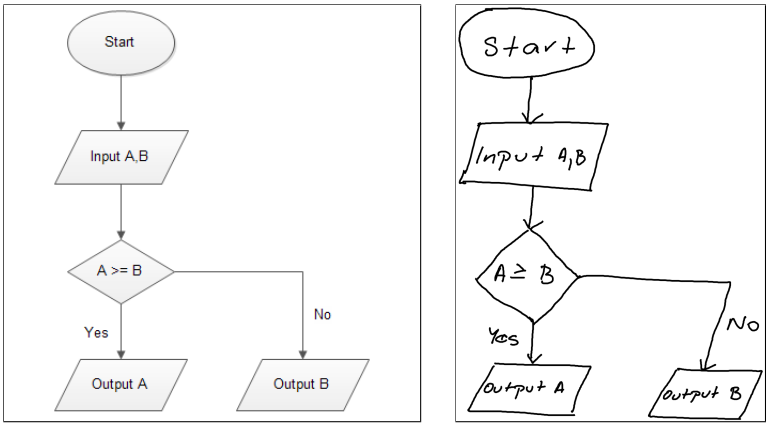


Figure 1.2: Exemplary pen-on-paper diagram. Recognition challenges include partially drawn shapes (1), bleed-through ink (2), multiple sheets and paper warping (3), crossed-out elements (4), interrupted lines (5), missing arrowheads (6), and crossing lines (7).

as input to a diagram recognition system, after which the recognized diagram is uploaded into a modeling tool for further editing. The usefulness of such an application depends on the accuracy of the recognition method. If the user only needs to manually correct some minor recognition errors, the semi-automated process depicted in Figure 1.1 is much faster than manually recreating the entire model from scratch. However, if the number of errors is very large, the overall time of the semi-automated process might exceed the time of the manual process. Therefore, an accurate hand-drawn diagram recognition method is needed to overcome the recognition challenges associated with camera-captured drawings on paper or whiteboard. Figure 1.2 illustrates some of these challenges for a business process diagram sketched on paper. This leads to the question of to what extent current methods are suited to recognize diagrams in such a challenging scenario, which we address next.

Research in hand-drawn diagram recognition. Most methods for recognizing hand-drawn diagrams have been developed specifically for diagrams that resemble the ones in existing datasets. However, the existing datasets, of which four have been published to this date [3,6,11,27], differ in the following aspects from camera-captured diagrams modeled on paper or whiteboard:

1. *Online datasets*: All existing diagram datasets are online datasets, in which each diagram has been drawn using a digital device (e.g., a tablet). These datasets promote research in online diagram recognition, where the drawing is provided as a temporal sequence of strokes. However, online methods are not applicable in our so-called offline scenario, where the input is an image. The lack of offline



(a) Flowchart Template **(b)** Hand-drawn diagram

Figure 1.3: Example template and diagram from the `FC_B` [11] online flowchart dataset.

datasets resulted in comparatively little attention towards offline methods.

2. *Template-based diagrams*: In existing diagram datasets, each participant was asked to copy a set of computer-generated flowchart [3, 11, 27] or finite automata [6] templates, resulting in well-organized sketches. This is illustrated in Figure 1.3, where the drawing closely follows the tidy layout of the template. Yet, in practice, diagrams are often sketched to solve a modeling task, such as graphically modeling software architectures or business processes [17, 24]. The iterative nature of conceptual modeling results in diagrams with chaotic layouts and arrows [70]. This is also illustrated in Figure 1.2, where the process model was sketched from a given textual process description.
3. *Modeling language*: Existing datasets use rather simple modeling languages with a small number of element types, including flowcharts, with five shape and one arrow type, and finite automata diagrams, with two shape and two arrow types. In comparison, a modeling language such as BPMN has a much larger vocabulary, differentiating between more than 70 shapes and four arrow types.
4. *Diagram labels*: In existing datasets, it is straightforward to identify the shape or edge that each label is associated with. This is shown in Figure 1.3, where shape labels are located within their associated shape, and edge labels are only used as binary indicators to define the control flow for a decision shape. In comparison, label recognition is much more challenging in modeling languages such as BPMN, where shape labels can be located outside of shapes, as observed for BPMN events and data elements in Figure 1.2. Moreover, there can be multiple shape or edge candidates close to a label, which makes it difficult to identify the

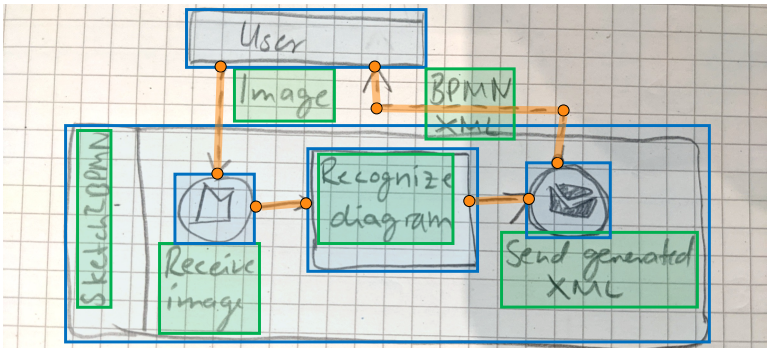


Figure 1.4: An *arrow-connected diagram* consists of shapes (blue), edges (orange), and labels (green). Each shape is defined by its type and bounding box, and each edge by its type, its source and target shape, and its drawn arrow path. Finally, each label is defined by its textual content, the shape or edge it is associated with, and its optional bounding box.

correct candidate. Presumably due to the less complex appearance of labels in existing datasets, existing works mostly focused on textblock detection, which is concerned with identifying the bounding box of each label. In fact, there is no existing method that fully addresses label recognition for camera-based images, even though the semantics of diagrams cannot be inferred without labels.

To summarize, there is a need for more research from both the dataset and method perspective. Regarding datasets, more realistic and challenging publicly available datasets of hand-drawn diagrams are needed, which originate from conceptual modeling tasks, and which have been captured by commodity cameras. In addition, methods are needed that can overcome the recognition challenges associated with such diagrams, and which can recognize diagrams in their entirety, including diagram labels.

1.2 Problem Description

This doctoral thesis is primarily concerned with the development of recognition methods for *arrow-connected*, *hand-drawn*, and *camera-captured* diagrams. Figure 1.4 shows an exemplary arrow-connected diagram that corresponds to this definition. In the following, we detail each of these three aspects.

Arrow-connected. The term diagram has a broad meaning and can refer to quantitative (e.g., a bar chart) and conceptual (e.g., a Venn diagram) diagrams. We follow related work [11] and use the term *arrow-connected diagram* to refer to the diagram type that this thesis focuses on. The BPMN diagram in Figure 1.4 illustrates that

an arrow-connected diagram consists of shapes (blue), edges (orange), and labels (green). Each shape is defined by a bounding box to denote its location and size, and a type that specifies its graphical notation. For example, the circular shape with the blank envelope in Figure 1.4 has the type *start message event*. Regarding edges, each edge is defined by the two shapes it connects, an arrow type, and the drawn path of the arrow. Figure 1.4 contains two arrow types, a solid and a dashed arrow, which both have different semantics in BPMN. In addition, the arrow path is specified through a sequence of waypoints located on the path from the source to the target shape. Finally, each diagram label is defined by its textual content, the shape or edge it is associated with, and a bounding box. In most modeling tools, labels located within their enclosed shape do not have their own bounding box but instead are automatically positioned in the center of the box. We consider the detection of the bounding box for such labels as optional, as it is not required for converting a hand-drawn diagram into its digital counterpart.

Overall, an arrow-connected diagram consists of shapes, edges, and labels, each of which have multiple components. For a diagram recognition system to be useful in practice, it needs to be able to recognize all these components.

Hand-drawn. Diagram recognition methods are usually developed specifically for either computer-generated or hand-drawn diagrams. A computer-generated diagram is a diagram that has been produced with a digital modeling tool, such as a flowchart modeled in Microsoft Visio. Such diagrams have been targeted by existing works with the goal of recognizing flowcharts in patents [63, 72, 78, 91] or BPMN diagrams created with a modeling tool [1, 43]. While the recognition of computer-generated diagrams is important, this doctoral thesis focuses on hand-drawn diagrams, i.e., diagrams drawn on a surface with a writing instrument, such as a pen on paper or a marker on whiteboard. However, the methods we develop for hand-drawn diagrams can also be used to recognize computer-generated diagrams. We demonstrate this in Chapter 8, where our approach outperforms the state-of-the-art method in computer-generated BPMN recognition.

Camera-captured. Handwritten documents are commonly converted into images using scanners or digital cameras, among which the analysis of camera-captured documents has proven to be much more difficult [5, 21]. Unlike scanned document images, which have a high resolution and fairly simple structure (black text on white background), camera-captured images come with a large number of additional recognition challenges, including varying resolution, uneven lighting, perspective distortion, and complex backgrounds [21]. This thesis focuses on the more challenging scenario of camera-captured images that depict hand-drawn diagrams. Compared to digitization with scanners, this enables a broader application scenario, since cameras can also be used to capture diagrams on other surfaces, such

as brown paper or whiteboards.

In summary, this thesis addresses the problem of recognizing diagrams that are arrow-connected, hand-drawn, and camera-captured. This combined scenario is more challenging than the scenarios addressed in most existing works and has so far received little attention. As we demonstrate in experiments, the methods proposed in this thesis not only perform well in our target domain but can also be applied to other scenarios, including computer-generated diagrams and images obtained from online datasets, where they outperform state-of-the-art methods. In the next section, we list the five main contributions of this thesis.

1.3 Contributions

This doctoral thesis is concerned with the automated recognition of hand-drawn diagrams contained in a given image. To this end, we collect a dataset of hand-drawn BPMN models, and we develop several methods that we not only evaluate on this dataset, but also on datasets of flowcharts and finite automata diagrams. In particular, the five main contributions of this thesis are the following:

1. *Collection and publication of a dataset that consists of diagrams sketched on paper*: The publication of the first online flowchart dataset by Awal et al. [3] achieved widespread attention and led to the development of numerous online recognition methods. In comparison, the lack of offline datasets has resulted in little attention being paid to offline methods. This is problematic, as in practice diagrams are often modeled on whiteboard or paper, which can only be recognized using offline methods. In Chapter 4, we address this issue by presenting the `hdBPMN` dataset, which consists of 702 business process models drawn on paper that we collected from 107 participants. `hdBPMN` contains over 70,000 manually annotated elements and covers more than 25 different types of shapes, edges, and labels. In addition, it has a high degree of diversity, as reflected in various drawing styles, paper types, pens, and digitization methods.
2. *Overview of challenges in recognizing camera-captured diagrams sketched on paper*: To enable users to seamlessly digitize their paper-based diagrams, it is important to develop diagram recognition methods that can work directly with the diagram images that users capture after their modeling session. For the design of such methods, it is important to know the key characteristics of camera-captured diagrams obtained from pen-and-paper sketching. Due to the lack of offline datasets, the recognition challenges associated with such diagrams are largely unexplored. We address this shortcoming in Chapter 4, where we analyze the `hdBPMN` dataset to identify the key challenges in recognizing

each diagram component. For example, we observe that the iterative nature of conceptual modeling results in diagrams with chaotic layouts, which leads to long-range arrows that cross other elements, and artifacts such as crossed-out diagram fragments. In addition, we identify general challenges related to paper types, pens, and camera-captured images.

3. *Development of an end-to-end system for recognizing business process diagrams*: Business process modeling is an integral activity in many organizations, with BPMN as the standard modeling language. As mentioned in Section 1.1, process models are often sketched on paper or whiteboard, which introduces the need for an automated diagram recognition system. In Chapter 8, we alleviate this problem by introducing `Sketch2Process`, the first approach for end-to-end recognition of sketched BPMN diagrams. Given an image of a hand-drawn BPMN diagram, our approach produces a BPMN XML file that can be imported into process modeling tools. Sketch2Process extends an object detector with neural network components for edge and label recognition. The next two points describe the major contributions related to these two components.
4. *Recognition of diagram edges using keypoint and visual relationship detection methods*: In the context of flowcharts and BPMN diagrams, the edges define the control flow, i.e., they indicate the execution order of the steps. Despite their importance, the recognition of hand-drawn diagram edges in images has received little attention. In fact, most existing methods are not applicable to camera-captured diagrams with complex backgrounds, or they are not able to identify the shapes that each arrow connects. This thesis addresses this research gap by proposing several edge recognition methods that are applicable in the mentioned scenario. Chapter 5 introduces a neural network for detecting arrow keypoints, which it combines with a heuristic to identify the connected shapes of an arrow. Chapter 6 adapts this approach to the specifics of BPMN and proposes rules to identify invalid edges. Chapter 7 addresses the inflexibility of the rule-based method by introducing an edge prediction network, which directly predicts whether two given shapes are connected through an edge. Finally, Chapter 8 improves the edge prediction network such that it can also recognize arrows that connect two shapes with a large detour.
5. *Recognition of diagram labels using a three-step approach*: Recognizing diagram labels can be decomposed into three subtasks: identifying the bounding box of each label (textblock detection), recognizing the handwritten text within each textblock (textblock handwriting recognition), and identifying the shape or edge that each textblock labels (textblock relation detection). As we detail in Chapter 3, the recognition of labels is hardly considered in existing works. In fact, no existing work that targets hand-drawn diagrams goes beyond textblock

detection. This is problematic, as the semantics of diagrams cannot be inferred without labels. In Chapter 8, we address this issue by proposing the first method that fully addresses the recognition of diagram labels. To this end, we formulate textblock relation detection as a visual relationship detection task and design a network architecture that learns to identify the shape or edge that each textblock labels. In addition, we propose a textual content decoding method that alleviates the shortcomings of off-the-shelf OCR services, which are mostly designed for handwritten documents.

1.4 Publications

The research conducted as part of this doctoral thesis has led to the publication of two journal, two conference, and one workshop paper, which are all concerned with the recognition of hand-drawn diagrams. The following list gives a chronological overview of the publications:

- B. Schäfer and H. Stuckenschmidt, "Arrow R-CNN for Flowchart Recognition," *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, 2019, pp. 7-13.
- B. Schäfer, M. Keuper, and H. Stuckenschmidt, "Arrow R-CNN for handwritten diagram recognition," *International Journal on Document Analysis and Recognition (IJDAR)*, 2021.
- B. Schäfer, H. van der Aa, H. Leopold, and H. Stuckenschmidt, "Sketch2BPMN: Automatic Recognition of Hand-Drawn BPMN Models," *Advanced Information Systems Engineering (CAiSE)*, 2021, pp. 344–360.
- B. Schäfer and H. Stuckenschmidt, "DiagramNet: Hand-Drawn Diagram Recognition Using Visual Arrow-Relation Detection," *International Conference on Document Analysis and Recognition (ICDAR)*, 2021, pp. 614–630.
- B. Schäfer, H. van der Aa, H. Leopold, and H. Stuckenschmidt, "Sketch2Process: End-to-end BPMN Sketch Recognition Based on Neural Networks," *IEEE Transactions on Software Engineering (TSE)*, 2022.

1.5 Outline

The remainder of this thesis is organized into eight chapters. As illustrated in Figure 1.5, the first chapters cover relevant existing datasets and methods, and subsequent chapters detail the dataset and methods that we propose in the context of this thesis. In particular, the eight chapters cover the following subjects:

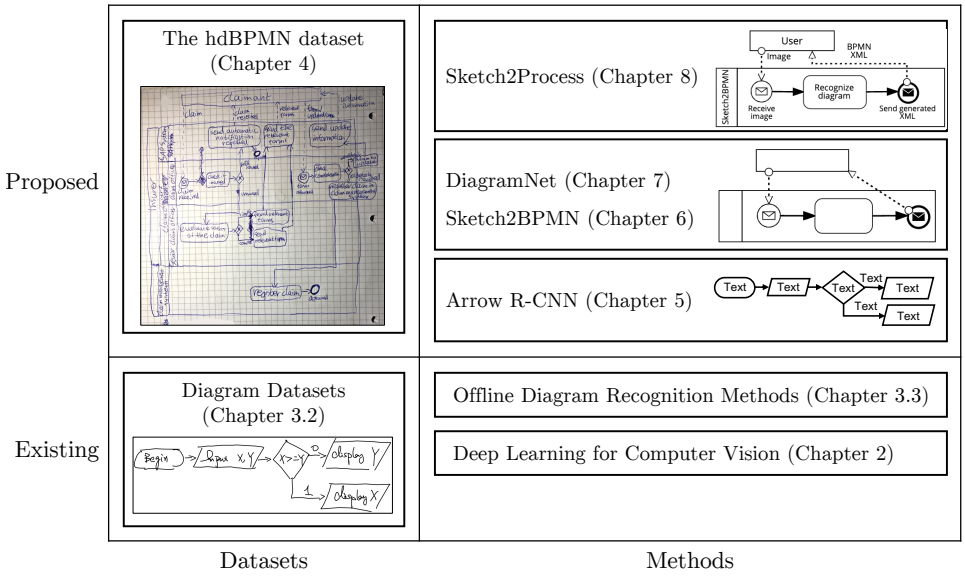


Figure 1.5: *Thesis outline*: we first cover relevant existing datasets and methods (first row), before we detail the dataset and methods proposed as part of this thesis (second row).

- *Chapter 2: Deep Learning for Computer Vision*. Deep learning methods have become the dominant approach for almost all computer vision tasks. They also form the basis of our proposed diagram recognition approaches and the ones of related works. Our research draws inspiration from three main computer vision tasks: object detection, keypoint detection, and visual relationship detection. Therefore, this chapter explains the deep learning methods that are commonly used to address each of these tasks.
- *Chapter 3: Diagram Recognition*. This chapter gives an overview of research in diagram recognition. After a general overview of the field itself, we detail the existing diagram datasets. Next, we survey related work that addresses offline diagram recognition. We conclude this chapter by summarizing the main research gaps that we identified.
- *Chapter 4: The `hdBPMN` Dataset*. This chapter presents the `hdBPMN` dataset that we created as part of our work. We first explain the collection procedure and the annotation process. Next, we provide various dataset statistics and elaborate on how we split the dataset into a training, validation, and test set. Finally, we detail the main recognition challenges that we identified for this dataset.
- *Chapter 5: Arrow R-CNN*. This chapter presents Arrow R-CNN, the first deep learning system for recognizing offline hand-drawn diagrams. Arrow R-CNN

extends a deep learning object detector with an arrow keypoint detector. For a given image, the object detector aims to locate the arrows, shapes, and textblocks. The detected arrow keypoints are then used to identify the shapes that each arrow connects. We evaluate Arrow R-CNN on the existing diagram datasets, for each of which it considerably outperforms the state of the art. Figure 1.5 shows a diagram from one of these datasets and the recognized diagram that Arrow R-CNN produces.

- *Chapter 6: Sketch2BPMN*. This chapter uses Arrow R-CNN to develop a first shape and edge recognition baseline for a preliminary version of the `hdBPMN` dataset with 502 images. The evaluation results indicate that `hdBPMN` is much more challenging than existing datasets, especially with respect to edge recognition. We also propose Sketch2BPMN, which improves Arrow R-CNN with a rule-based method that prevents invalid edges, and an extended data augmentation pipeline that simulates the varying properties of camera-based documents.
- *Chapter 7: DiagramNet*. In this chapter, we present DiagramNet, our first approach that recognizes edges using a learning-based method inspired by visual relationship detection research. Sketch2BPMN uses a rule-based method to identify the source and target shape of each edge. Even though this approach outperforms Arrow R-CNN, it still accounts for a large proportion of edge recognition errors. Against this background, DiagramNet introduces an edge prediction network, which directly predicts whether two given shapes are connected through an edge. We evaluate DiagramNet on the preliminary `hdBPMN` dataset, where it considerably outperforms Sketch2BPMN in edge recognition.
- *Chapter 8: Sketch2Process*. In this chapter, we present Sketch2Process, the first approach that provides end-to-end recognition of hand-drawn BPMN models from images. Unlike Sketch2BPMN and DiagramNet, Sketch2Process also targets label recognition, which is illustrated in Figure 1.5. Apart from the expanded scope, Sketch2Process improves the edge recognition component of DiagramNet in a way that it can also recognize arrows that connect two shapes with a large detour. We evaluate Sketch2Process on the final `hdBPMN` dataset with 704 images, where it outperforms related work and our previous works.
- *Chapter 9: Conclusion*. In this chapter, we summarize the main findings of this thesis, and we discuss the implications of the findings for research and practice. Finally, we give an outlook on directions for future research.

As mentioned in Section 1.4, some parts of this thesis have been published in research papers. This especially concerns the proposed `hdBPMN` dataset (Chapter 4) and diagram recognition methods (Chapters 5–8). For that reason, we refer to the respective publications in the introduction of each mentioned chapter.

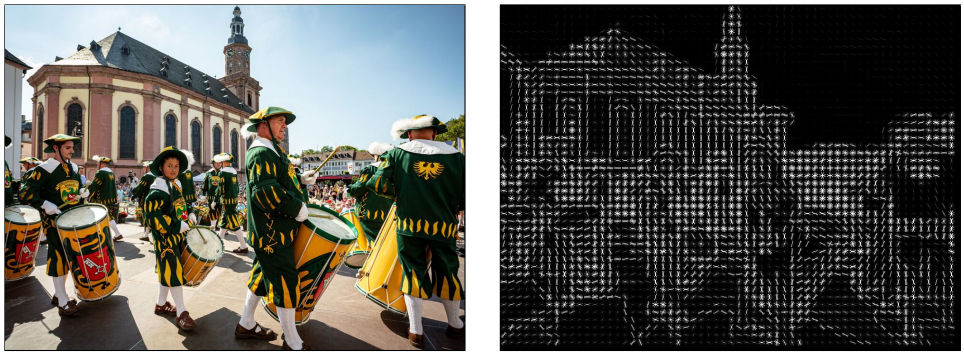
Chapter 2

Deep Learning for Computer Vision

This chapter covers the computer vision fundamentals that our diagram recognition approaches, and the ones of some related works, build on. Section 2.1 introduces the field of computer vision by discussing the advancements in image classification, a classical problem in computer vision. This section also explains the functioning of a convolutional neural network, which is an essential component of most computer vision solutions today. Given this groundwork, the next three sections focus on the main computer vision tasks that our research draws inspiration from. Section 2.2 gives an overview of object detection. In particular, it details Faster R-CNN, the detector that we use in our methods. Section 2.3 introduces the field of (human) keypoint detection and describes popular methods used in this domain. Our research is also inspired by this domain, as we use a keypoint detection method to predict the drawn path of an arrow. Last, Section 2.4 explains visual relationship detection (VRD), where the task is to predict the relationship between objects in natural images. In our research, we use approaches inspired by VRD as part of our edge and label recognition components.

2.1 Image Classification

A classical problem in computer vision is image classification, where the task is to predict a class for an entire image. This task assumes that each image can be assigned to exactly one class out of a set of predefined classes. A key challenge in image classification (and computer vision in general) is that images of a particular class (e.g., a person) are highly variable. One source of variation is the actual image-capturing process. Changes in aspects such as brightness or camera position



(a) Input image **(b)** Histogram of oriented gradients

Figure 2.1: Histogram of oriented gradients (HOG) example

all produce significant variations in image appearance, even in a static scene. A second source of variation is the variability in the intrinsic appearance of a class. For example, the pose of people can vary substantially, and on top, people can wear a large variety of clothes. The challenge is to develop computationally efficient algorithms that are *invariant* concerning these variations.

In the following, we first describe the traditional vision pipeline, which addresses this challenge by manually designing low-level features that are invariant to certain variations. Next, we describe the deep learning approach, which learns the whole pipeline, from raw pixels to outputs, using end-to-end training. In particular, we detail convolutional neural networks and transformers, which form the basis of most computer vision solutions.

Traditional vision pipeline. A popular example of a traditional vision approach uses the histogram of oriented gradients (HOG) [18]. In the HOG approach, the image is divided into small spatial regions. Next, a histogram of oriented gradient directions is computed per region. Figure 2.1 visualizes the HOG descriptors for a given image, which we have computed using the *scikit-image* [97] library. Finally, the HOG descriptors from all regions are combined into a feature descriptor. Given the feature descriptors, Dalal and Triggs [18] use a linear SVM classifier to predict if an image contains a person or not. A similar approach was proposed by Lin et al. [59] in 2010, which uses a linear SVM based on HOG and other feature descriptors. With a top-5 error rate of 28.2%, their approach won the 2010 ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [79], an annual challenge to track progress in image classification and other object recognition tasks.

As mentioned, an alternative approach to the traditional pipeline is the deep learning approach. This idea, which is typically implemented with convolutional

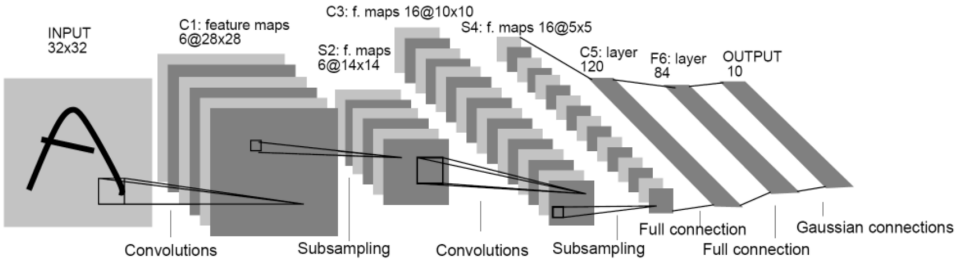


Figure 2.2: Architecture of LeNet-5 (Figure from LeCun et al. [51])

neural networks, dramatically reduced the ILSVRC error rates in 2012 at beyond, which we describe next.

Convolutional neural network. The convolutional neural network (CNN) was popularized in 1998, when LeCun et al. [51] introduced the LeNet-5 network for recognizing handwritten digits, which is shown in Figure 2.2. As illustrated, a CNN is composed of a hierarchy of layers. In the context of images, the input to the first layer is the raw pixels of the input image. Each subsequent layer learns a feature representation, typically using large annotated datasets. The first layers learn low-level features, which often resemble the features extracted by hand-crafted edge detectors, whereas the upper layer combines the local features to produce more semantically meaningful high-level features. In standard feedforward neural networks, all units in a layer are connected to all units in the preceding layer. In comparison, a CNN organizes each layer into feature maps, as indicated in Figure 2.2. Unlike a color channel in an RGB image, which represents the color intensity of one of the three primary colors red, green, and blue, each feature map represents to what extent a feature is present at a certain image location.

In a convolutional layer, a feature map is computed by sliding a filter with learned weights over the feature maps of the previous layer. Here, a filter contains different convolution kernels for each feature map of the previous layer. The output of a kernel at a given position is computed as the weighted sum of the pixel values within a small window (the kernel size), and the filter output is obtained by summing over the kernel results. Because the weights in a convolution kernel are the same for all of the pixels within a given layer and channel, there are much fewer weights to learn than in fully-connected layers. For example, as illustrated in Figure 2.2, the first convolutional layer of LeNet-5 consists of six filters, each of which produces a 28×28 feature map by sliding a 5×5 kernel over the (grayscale) input image. If the input image was an RGB image, as is standard in modern CNNs, each filter of the first convolutional layer would have three kernels.

As mentioned, the weights of the convolutional and fully-connected layers of a

CNN are trainable. The training is done through stochastic gradient descent (SGD), an iterative method, where, at each step, k images are sampled from the training set [51]. Here, k is referred to as the batch size, and the set of sampled images is called a mini-batch. During each SGD iteration, a loss function quantifies the difference between the outputs and the ground-truth labels in an annotated dataset. For a comprehensive introduction to CNNs and how they are trained, we refer the reader to Goodfellow et al. [29] and Szeliski [93].

Despite their successful application, CNNs were rather unpopular until 2012, when a CNN called AlexNet [47] achieved spectacular results in the ILSVRC, winning with almost half the error rate of the second-best method. This success came from the efficient use of GPUs and some innovations in the network architecture. In the following years, CNNs became the dominant approach for almost all recognition and detection tasks [52]. Further developments on CNNs resulted in additional performance improvements in subsequent ILSVRC. One notable improvement is the introduction of the Residual Network (ResNet) [36] in 2015, which won the 2015 ILSVRC with a top-5 error rate of just 3.6%. The main innovation of ResNets was the introduction of skip connections, which significantly increased the number of layers that could successfully be trained. For example, the ResNet that won the competition had 152 layers, which is much deeper than AlexNet, the winner from 2012, which has just eight layers.

Besides CNNs, other network architectures are used in image classification and related domains. The most notable architecture is the transformer, which we describe next.

Transformer. The Transformer [99] is a sequence-to-sequence network based on an attention mechanism, which became the dominant approach in natural language processing soon after its introduction in 2017. With the introduction of the Vision Transformer (ViT) [22] in 2021, approaches based on transformers also started to outperform CNNs in vision tasks such as image classification. As shown in a follow-up work by Zhai et al. [115], transformers achieve excellent results when pretrained on very large datasets with more than 10 million images. However, the authors also mention that they do not generalize well when trained on insufficient amounts of data, as transformers lack some of the inductive biases inherent to CNN, such as translation equivariance and locality.

Unlike images of natural scenes, which can be collected in abundance on the Internet, images of hand-drawn diagrams are comparatively rare. As our work targets this domain with limited data, our methods proposed in subsequent chapters all rely on CNNs to learn image features. In particular, we use a ResNet variant in all experiments, given that they are easy to train and provide a good performance. The next section describes object detection, where CNNs play an essential role.

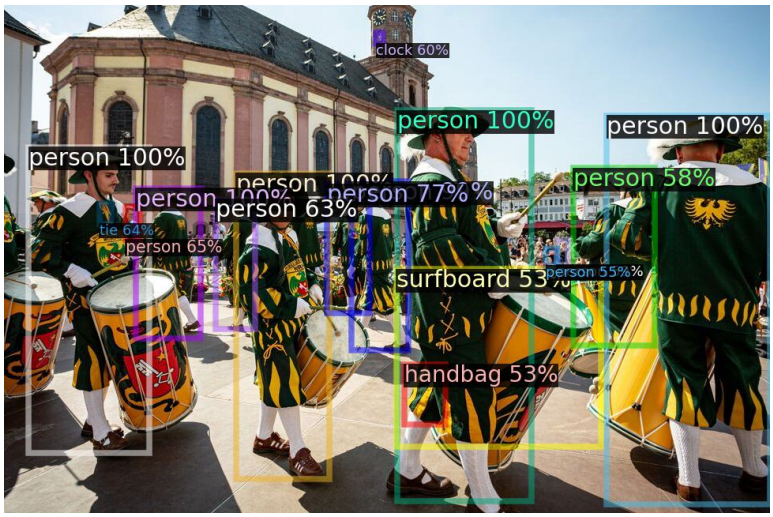


Figure 2.3: *Object detection example*: The visualized instances have been detected by a Faster R-CNN [77] system that has been trained on the COCO dataset [58].

2.2 Object Detection

In this section, we first provide an introduction to object detection, and then explain the details of Faster R-CNN, the detector that our approaches build on.

2.2.1 Overview of Object Detection

The goal of object detection is to detect all object instances from one or more known classes in an image. The most popular benchmark for object detection is the COCO dataset [58], where the goal is to detect objects from 80 common categories such as people, cars, and dogs, in photographs. The COCO dataset consists of 328k images, with a total of 2.5 million manually annotated instances. Figure 2.3 gives an example of the detected instances of an object detector trained on COCO.

In object detection, each detected instance can be defined by a tuple (b, c, s), with b as its bounding box, i.e., a rectangle encompassing the predicted area of the instance, c as the predicted class of the instance, and s as the classification score. The main challenge in object detection is that although an image usually contains a small number of objects, they can appear at a very large number of possible locations and in various scales. The most common approaches to object detection use the concept of a sliding window, which reduces the problem to one of image classification. In the sliding-window approach, the image is divided into a grid. If the center of an object is located within a grid cell, the cell is responsible

for detecting that object. Unlike networks for image classification, which have a relatively simple design due to their single classification output, networks for object detection have additional components to address the more complex nature of the task. There are two dominant approaches to object detection: one stage (e.g., [73]) and two stage (e.g., [77]).

In the one-stage approach, a classification model predicts K objects for each grid cell, where the prediction encompasses the class, bounding box, and score of each object. There are two reasons to predict multiple objects per grid cell. First, there might be more than one object whose center is located within this grid cell. Second, each object slot has a different bounding box prior, such that each slot learns to predict boxes relative to a specific scale and aspect ratio, which leads to a better performance [56, 75]. During postprocessing, one-stage detectors remove instances whose score is below a threshold, and they apply non-maximum suppression (NMS), in which object duplicates, i.e., multiple instances with a large bounding box overlap and the same predicted class, are resolved by only keeping the instance with the highest score.

In the two-stage approach, the first stage generates a set of so-called region of interests (RoIs), where each RoI is defined by a bounding box and an objectness score. The second stage classifies each RoI and predicts a refined bounding box. Due to their flexibility, two-stage systems are extended for various computer vision tasks beyond object detection. They also form the basis of most approaches that target the tasks also discussed in this chapter: human keypoint detection (Section 2.3) and visual relationship detection (Section 2.4). In particular, most extensions build on Faster R-CNN [77], a popular two-stage system. As our approaches also extend Faster R-CNN, the next section explains this system in detail.

2.2.2 Faster R-CNN

Faster R-CNN is the successor of R-CNN [28], which has popularized a two-stage approach in object detection, and consists of three sub-networks: A *CNN backbone network*, a *region proposal network (RPN)*, and a *box-head network*.

CNN backbone network. As in image classification, Faster R-CNN uses a CNN to learn a feature representation of the image. The CNN is commonly initialized by pre-training an image classification model on a large dataset (e.g., ImageNet), of which the weights of all layers up until the last convolutional layer are used. The fully-connected hidden layers and the output layer of the image classification model are not considered, as these are specific to the task at hand. During object detection training, the CNN is jointly trained with the other networks, as we elaborate at the end of the section. Given an image, the CNN extracts a feature representation with a smaller spatial resolution $w \times h$ than the original image, but

a much higher number of channels. The leading example image in Figure 2.3 has a size of $970 \times 728 \times 3$, where the third dimension represents the three RGB channels. Using a standard ResNet with 50 layers, this image is transformed into a $67 \times 50 \times 1024$ feature representation.[1]

One of the limitations of Faster R-CNN is that it has difficulties with datasets where objects have a large scale variance. Lin et al. [57] addressed this issue by incorporating *Feature Pyramid Networks (FPN)* into Faster R-CNN. In this extension, the backbone network generates a pyramid of features at different scales. The image feature pyramid is a multi-scale feature representation in which all levels are semantically strong, including the high-resolution levels. As the FPN extension is both faster and more accurate, we use such a backbone in all our approaches. During the description of the box-head network, we explain how and why this pyramid structure increases the accuracy of the box-head network. In the leading example, the ResNet-50-FPN backbone transforms the image into a pyramid with five levels, where each level has 256 channels, and the spatial dimensions range from 17×13 to 272×200.

Region proposal network. Given the image feature pyramid, the region proposal network (RPN) predicts a large set of RoIs, i.e., locations in the image that might contain an object. In image classification, the final layer produces a fixed-size output, whose size corresponds to the number of classes. This is different in object detection, where the number of objects per image, and thus also the number of meaningful RoIs, varies greatly. To predict a varying number of RoIs, the RPN uses the concept of reference bounding boxes, also referred to as anchors.

At each pixel position in an image feature map, Faster R-CNN creates K anchors, which vary in terms of size and aspect ratio. For example, the Faster R-CNN implementation with the FPN extension uses $K = 3$ anchors with aspect ratios of $\{1{:}2, 1{:}1, 2{:}1\}$, and it assigns a different anchor size to each pyramid level. Specifically, the anchor sizes are defined as $\{32^2, 64^2, 128^2, 256^2, 512^2\}$, where 32^2 is used for the highest-resolution, and 512^2 for the lowest-resolution level. For each anchor, the RPN predicts an objectness score and a refined bounding box. The RPN is trained to predict an objectness score of 1 for each anchor that has the highest bounding box overlap with a ground-truth object, and 0 for all other anchors. The bounding box overlap is quantified using the Intersection-over-Union (IoU), which is measured as the intersection area divided by the union area of both bounding boxes.

For the leading example in Figure 2.3, Faster R-CNN generates more than 200k anchors, of which 163,200 ($272 * 200 * 3$) stem from the highest-resolution pyramid level. During training, Faster R-CNN computes the RPN loss on a fixed number

[1] Specifically, we have used the `R50-C4` configuration from the Detectron2 Model Zoo.

Figure 2.4: Faster R-CNN top-100 region of interest (RoI) ranked by objectness score.

of sampled anchors. As the class imbalance between positive and negative anchors is very large, the anchors are sampled such that the positive and negative anchors have a ratio of 1:1. During inference, Faster R-CNN eliminates RoI duplicates using NMS, after which it keeps the top 1000 RoIs ranked by objectness score. For the leading example, Figure 2.4 illustrates the top-100 RoIs. Given the RoIs, the box-head network predicts the final object instances, which we describe next.

Box-head network. In the second stage of Faster R-CNN, the *box-head network* predicts the object class and final bounding box of each RoI, which is illustrated in Figure 2.5.

For each RoI, Faster R-CNN uses the *RoIAlign* [35] method to generate a feature representation that is used as input for the network. RoIAlign uses the size of the RoI bounding box to match it to a pyramid level. Next, it cuts out the feature map positions that are (partially) located within the bounding box. Finally, it applies a pooling mechanism to reduce the RoI feature maps to a fixed spatial size of 7×7. In the original Faster R-CNN implementation without feature pyramids, the single feature map has a much lower resolution than the highest-resolution pyramid level, which leads to very low-resolution representations for small bounding boxes. The key advantage of FPN is that the varying-resolution feature maps can be much better aligned with the varying-size bounding boxes.

Given the $7 \times 7 \times 256$ RoI features, the structure of the box-head network is relatively straightforward: two 1024-d fully-connected layers (each followed by ReLU) process the input, before a classification and a bounding box regression layer predict the outputs. For the classification task, the object classes are extended with an additional background class, so that the network can recognize false-positive RoIs. During inference, Faster R-CNN removes these background

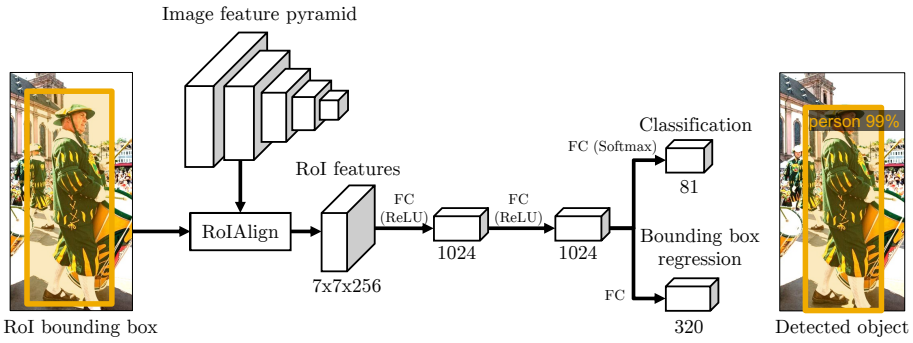


Figure 2.5: *Faster R-CNN box-head network*: Given the RoI bounding box, *RoIAlign* extracts the RoI features from the image feature pyramid. The box-head network processes the features using two 1024-d fully-connected layers (FC) with ReLu activation. A classification and a bounding-box regression layer predict the class and bounding-box refinements, which get transformed into a detected object during post-processing.

RoIs. Further, it applies a score threshold and runs NMS for each class to obtain the final detected objects. In addition, the encoded bounding boxes, which have been predicted relative to the RoI boxes, are converted into absolute bounding boxes. During training, the loss of the box-head network is computed on a sample of RoIs, again with a positive ratio of 50%.

For the overall training, the Faster R-CNN loss is defined as the sum of the RPN and the box-head network losses. During backpropagation, the weights of all networks are updated, including the CNN backbone. In this multi-task training setup, the CNN backbone learns features specific to the tasks in both stages.

2.3 Keypoint Detection

Keypoint detection methods are commonly used for pose estimation tasks in natural images. In *human pose estimation* (HPE), the task is to predict human body keypoint locations. Figure 2.6 shows the human keypoints predicted by a model that has been trained on the COCO keypoint dataset [58], a prominent benchmark for keypoint detection methods. The dataset contains over 200k images with more than 250k annotated persons, each with 17 annotated keypoints.

Most HPE systems approach multi-person keypoint estimation in a top-down process. In the first stage, person instances are detected with an object detector. In the second stage, the person instances are cropped from the image, resized to a fixed resolution, and fed into a dedicated single-person pose estimation network. The only notable exception to this strategy is Mask R-CNN [35], which

Figure 2.6: *Human pose estimation example*: The human keypoints have been predicted by a Mask R-CNN [35] model trained on the COCO keypoint dataset [58].

directly extends Faster R-CNN with a keypoint-detection network. This keypoint network predicts a 56×56 heatmap per person and keypoint, where each heatmap is transformed into a keypoint by determining the position with the highest heat value. Overall, the heatmap approach has become very popular in recent years, and it has been used by winning systems in the COCO keypoint detection challenge [35, 90, 105, 116]. Apart from the heatmap approach, a second popular approach is to directly regress the position of keypoints [92, 96]. In this approach, a regression layer predicts the keypoints relative to the person bounding box, which is conceptually similar to bounding box regression in Faster R-CNN.

As shown in Chapter 5, we formulate arrow path prediction as a keypoint detection problem. While the HPE domain can serve as inspiration for arrow keypoint detection, there are substantial differences in dataset size and task characteristics.

Dataset size. Unlike the COCO keypoint dataset, which contains more than 200k images with 250k persons, existing diagram datasets have less than 1,000 images, which contain between 2,800 and 4,500 arrows. Given the larger dataset, HPE methods can use much deeper and more complex network architectures.

Task characteristics. Human keypoint detection comes with several challenges that are not present in arrow keypoint detection. Person keypoints can be occluded, under-exposed, and blurry, and keypoints are often either not visible or there is some visual ambiguity. This forces the models to consider spatial and contextual relationships. Popular architectures for HPE, such as stacked hourglass networks [66], try to enable such contextual reasoning with complex CNN architectures. In contrast to person keypoints, arrow keypoints are typically clearly visible. Further, there is a task overlap between arrow keypoint detection and bounding box detection: arrow head and tail keypoints are often the outermost arrow pixel

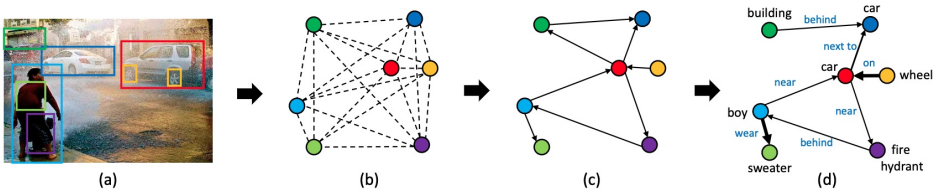


Figure 2.7: Scene graph generation: Figure from Yang et al. [110]

in one spatial direction, in which case they define a border of the bounding box. This suggests that it might be beneficial to learn both tasks together, and use shared feature extraction layers.

To conclude, although HPE is dominated by deep networks that rely on heatmap methods, this does not mean such approaches work best for recognizing arrow keypoints. As shown in Chapter 5, we instead opt for a lightweight keypoint regression method, which shares its features with the object detector network. We find this approach is effective in the diagram domain, where most datasets are small, and head and tail keypoints are typically located at the bounding box border.

2.4 Visual Relationship Detection

In visual relationship detection (VRD), the task is to predict the relationship between objects in natural images [60]. In our research, we frame two diagram recognition subtasks as VRD problems. In particular, we define two visual relations, one to identify the source and target shape of each arrow (edge relation), and one to identify the shape or edge that each label is associated with (label relation). A popular application of VRD is scene graph generation, which we describe next. Given the popularity of scene graph methods for natural images, similar approaches have also been developed in the document domain, which we also discuss in a dedicated paragraph.

Scene graph generation. In scene graph generation, a graph models the objects (nodes) and their relationships (edges) in a scene image, as illustrated in Figure 2.7. There are several ways to formulate hand-drawn diagram recognition as an image-conditioned graph generation task. For example, the diagram shapes and labels could be defined as the nodes of the graph, and the edge-relation and label-relation form the edges of the graph. Existing methods decompose scene graph generation into a sequence of multiple steps [106, 110, 114, 117]. First, they use Faster R-CNN to generate a set of object proposals. Given that reasoning over the quadratic number of potential relationships in the scene graph is intractable, existing methods use

heuristics or learning-based approaches to come up with a set of candidate object pairs. For a given object pair, the common approach is to extract the visual features from the union of both object bounding boxes using RoIAlign [35]. For our baseline approach in Chapter 7, we also extract the relation features for edge recognition using the union bounding box of a candidate shape pair. However, we observed that this simple strategy is only partially suited for edge recognition, as an arrow can be partially located outside the union bounding box when it connects two shapes with a detour. After extracting the node and edge features of the graph, these features are enriched with global context using recurrent or graph neural networks, before predicting the final set of relations. Scene graph methods are commonly evaluated on datasets with more than 100k images [45]. Given that we target a domain with small datasets, we opt not to update local shape and edge features based on the global context, in order to prevent our method from simply memorizing the graphs in the training set. However, from a conceptual view, our approaches presented in Chapter 7 and Chapter 8 follow a similar multi-stage approach.

VRD for Documents. In the document domain, VRD approaches have been developed to relate different kinds of textblocks [19, 20]. Davis et al. [19] use a VRD approach to detect and associate pre-printed and input texts in historical form images. To detect the pre-printed and input text lines in the image, they use a one-stage object detector similar to YOLOv2 [74]. Next, the relationship candidates are generated using a line-of-sight approach, which considers spatially close objects whose direct connection path does not cross the bounding box of other objects. This spatial heuristic for candidate generation works well for forms, where associated pre-printed texts are typically located next to or below the input text. But it is not suitable for arrows, which also connect shapes far away from each other. To classify object pairs, Davis et al. propose a relationship classifier network, which receives visual and spatial features as input. Further, a neighbor prediction network predicts the number of associated objects for each text object. To determine the final set of relations, a global optimization step combines the relationship probabilities and the predicted number of neighbors. This is similar to our edge optimization procedure in Chapter 7, except that our edge relations are directed, and that we predict the in- and out-degree of each object, instead of the (undirected) number of neighbors.

The next chapter covers related work in diagram recognition, of which many build on the computer vision fundamentals introduced in this chapter.

Chapter 3

Diagram Recognition

This chapter gives an overview of research in diagram recognition. Section 3.1 provides a short historical outline and explains the difference between online and offline recognition methods. Section 3.2 details the datasets that are commonly used to evaluate offline methods. This lays the foundation for Section 3.3, which surveys related work that addresses offline diagram recognition. Finally, Section 3.4 summarizes this chapter, and it also discusses the research gaps that we identified in the area of diagram recognition.

3.1 Overview of Diagram Recognition

Research in diagram recognition dates back to the early days of artificial intelligence and computer vision. In the 1960s and 1970s, researchers developed screens that can record the drawing of a user as a sequence of strokes, and they proposed algorithms that can recognize simple geometric shapes from the recorded strokes [40, 65]. In the decades that followed, researchers continued to build on these early ideas. For example, in the 1990s, researchers began using devices that consist of a digitizing tablet and a pen for input, and from a recognition perspective, they started developing template-based methods to recognize diagram elements such as arrows, rectangles, and circles [31, 32, 48]. Overall, research before 2000 was focused on developing recognition methods for digital sketching tools.

These recognition methods, where the input consists of a temporal sequence of strokes, are commonly referred to as *online recognition* methods. In the context of a digital pen, each stroke is captured as a sequence of points between pen-down and pen-up events. Each stroke point is then defined by its x and y coordinates on the drawing canvas, and by an associated time stamp. In addition, a pressure value may be provided by the input device. Online recognition methods typically

leverage the richness of this data. For example, early approaches detected stroke corners by identifying a group of spatially close points where the pen slowed down and afterward accelerated again [31].

Complementary to online recognition, there is also *offline recognition*, where the input is represented by a picture. This picture is commonly obtained by scanning a piece of paper or taking a photo of a whiteboard or paper. In this context, all the stroke information is missing and recognition is thus more difficult [71]. Research in offline diagram recognition can be divided into two subtypes based on how the diagram was created. Besides hand-drawn diagrams, there are also computer-generated diagrams, i.e., diagrams created through digital tools such as Microsoft Visio. The recognition of computer-generated diagrams is much easier, as, e.g., it is straightforward to distinguish the white background from colored (foreground) diagram pixels. Early research in offline diagram recognition focused on computer-generated diagrams, presumably as these are easier from a recognition perspective. In the 1990s, researchers worked on methods for recognizing flowcharts scanned from books [111]. In the next decades, methods were also developed to recognize other types of computer-generated diagrams, such as flowcharts in patents [63] and process flowcharts in the telecom domain [98]. In Section 3.3, we review recent works in offline diagram recognition in more detail.

While this doctoral thesis focuses on the offline scenario, there is an overlap with online recognition concerning the utilized *methods* and *datasets*. From a *method perspective*, online methods that only operate on spatial coordinates, i.e., that require neither temporal nor pressure information, can be adjusted for offline recognition. To this end, a stroke reconstruction preprocessing step can extract a set of strokes from an image. From a *dataset perspective*, the recorded diagram strokes in an online dataset can be converted into an image by rendering them on a white canvas. Due to the lack of public datasets that are offline by nature, this strategy is commonly used to evaluate offline systems [9, 26, 41, 103]. In the next section, we give an overview of the diagram datasets that are commonly used to evaluate offline methods, which also includes online datasets.

3.2 Diagram Datasets

In 2011, Awal et al. published a paper with an accompanying online hand-drawn flowchart dataset, which is commonly referred to as `FC_A` [3]. `FC_A` consists of 419 flowchart drawings, one of which is illustrated in Figure 3.1. The flowcharts are composed of arrows, textblocks, and five different shape types (data, terminator, process, decision, and connection). As mentioned in Section 1.1, the flowcharts are template-based, which means that each participant was asked to copy a set of

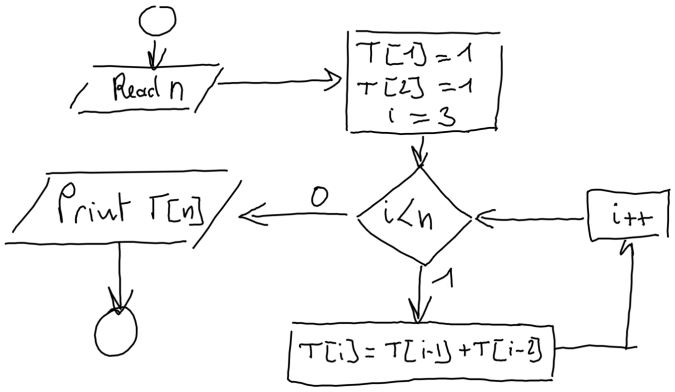


Figure 3.1: Exemplary flowchart from the FC_A dataset

predefined computer-generated flowcharts, which results in well-organized drawings. The FC_A dataset sparked a lot of interest in online flowchart recognition in the years after 2011, which led to the development of a large number of online methods [6–8, 10, 11, 14, 42, 53, 101, 102, 112].

A significant number of papers on online flowchart recognition have been published by Bresler et al. [6–8, 10, 11] Accompanying some of these papers, the authors introduced two datasets, one of which consists of flowcharts (FC_B dataset), and the other which contains finite automata diagrams (FA dataset). The FC_B dataset contains 672 flowchart drawings and is highly inspired by the FC_A dataset. For example, it uses the same symbols and the flowchart templates of FC_A, one of which is depicted in Figure 1.3. The FA dataset contains 300 finite automata diagrams, where each diagram consists of an initial arrow (an arrow without a source shape), a plurality of state (a circle) and final state (two concentric circles) shapes, and regular arrows and textblocks. A large number of textblocks consist of just one character. In addition, unlike the arrows in the flowchart datasets, the arrows can have the same source and target shape, and there can be two edges that connect the same shapes in opposite directions, both of which are illustrated in Figure 3.2.

The most recent public dataset is the DIDI online flowchart dataset, which has been published as part of a paper by Gervais et al. in 2020 [27]. DIDI is the first large-scale diagram dataset and consists of two parts: 22,287 diagrams with textual labels ($\texttt{DIDI}_{\texttt{text}}$) and 36,368 diagrams without textual labels ($\texttt{DIDI}_{\texttt{no\_text}}$). Each hand-drawn diagram has been collected by showing an image of a computer-generated flowchart to the participant, who was then asked to copy the flowchart by drawing over it. The computer-generated flowchart images were randomly generated using GraphViz. In particular, the authors randomly generated files in the dot format, a textual graph description language. Overall, the dataset features 6,555

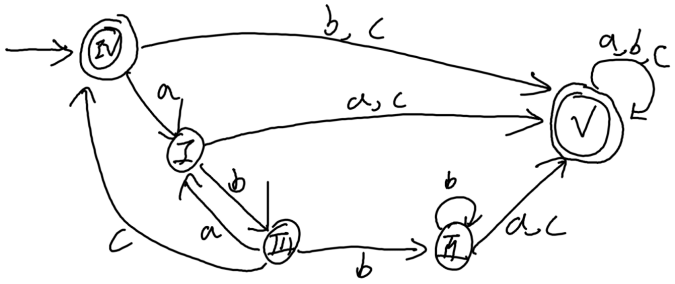
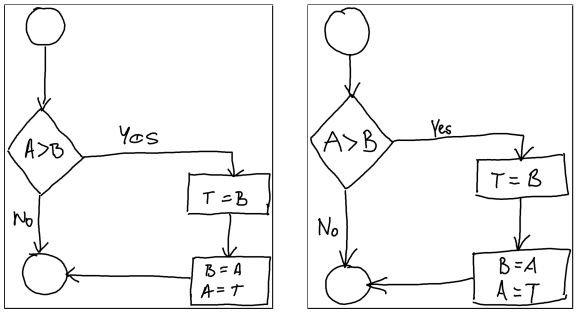


Figure 3.2: Exemplary finite automata diagram from the FA dataset



(a) Training diagram **(b)** Test diagram

Figure 3.3: Exemplary FC_B diagrams from the training and test set

flowchart templates generated using GraphViz. Unlike other online datasets, the drawings are not manually annotated, neither on stroke level nor using bounding boxes. While the position and size of all elements of a flowchart template can be obtained from the corresponding dot file, these positions are only meaningful if the participant accurately copied the hand-drawn diagram over the template. Unfortunately, though, the dataset contains a lot of drawing errors, such as drawings with missing elements, or drawings where the diagram was drawn next to the flowchart template. Chapter 5 provides an in-depth analysis of these errors.

Table 3.1 provides some statistics for all mentioned datasets, with a special emphasis on how the datasets were split into training, validation, and test set. As depicted, all datasets were split either by writers (FC_B, FA, DIDI) or by templates (FC_A). In the former case, the sets of writers in the training, validation, and test splits are disjoint. This means that experimental results show to what extent the model generalizes to unseen writers. However, this also means that the same template can appear in multiple splits. As indicated in Table 3.1, in the FC_B and FA datasets, each template is present in all three splits. Figure 3.3 shows that the diagrams from one template can look very similar. This means that very good

Table 3.1: *Overview of existing public diagram datasets*: Before this thesis, all existing public datasets were online datasets. The writer and template counts are provided per split and overall, which reveals how the dataset was split into its parts.

Dataset	Splits	Writers	Templates	Diagrams	Symbols
FC_A	Train	31/35	14/28	248	5,540
	Test	15/35	14/28	171	3,791
FC_B	Train	10/24	28/28	280	6,195
	Validation	7/24	28/28	196	4,342
	Test	7/24	28/28	196	4,343
FA	Train	11/25	12/12	132	3,631
	Validation	7/25	12/12	84	2,307
	Test	7/25	12/12	84	2,323
$\texttt{DIDI}_{\texttt{no\_text}}$	Train	?/364*	940/940	27,278	193,939
	Validation	?/364*	916/940	4,545	34,464
	Test	?/364*	919/940	4,545	34,139
$\texttt{DIDI}_{\texttt{text}}$	Train	?/364*	5,300/5,629	16,717	173,070
	Validation	?/364*	2,131/5,629	2,785	30,468
	Test	?/364*	2,090/5,629	2,785	34,052

* DIDI is split by writer, but the distribution is unknown as the writer identifiers are not public.

evaluation results might simply indicate that the model has memorized the training diagrams, and can recognize them when the writing style slightly varies. In other words, very good evaluation results could still mean that the model might perform poorly on similar diagrams with unseen layouts.

Unlike the three other datasets, the FC_A dataset was split by template in a way that 14 templates are used in the training set and another 14 templates in the test set. However, most participants that appear in the test set are also present in the training set. This means that experimental results show to what extent the model generalizes to unseen layouts, but not to unseen writing styles. As discussed in Chapter 5, we find that the template-based split is slightly more challenging for deep learning methods, though, as the diagrams of each template are very similar in existing datasets.

In summary, we observe that all four public datasets of hand-drawn diagrams are online datasets. Due to the lack of offline datasets, offline methods commonly evaluate their methods on these datasets, which we describe in the next section.

3.3 Offline Diagram Recognition Methods

In this section, we review existing offline diagram recognition methods and discuss their applicability to the domain this doctoral thesis addresses, namely camera-captured hand-drawn diagrams with non-uniform backgrounds. Existing approaches for offline diagram recognition can be further subdivided into stroke-based and object-based approaches, which we detail in the following.

3.3.1 Stroke-based Methods

The more traditional stroke-based methods try to first reconstruct the strokes in an image in a preprocessing step, after which they apply a recognition pipeline that eventually identifies the set of strokes that corresponds to each symbol.

To our knowledge, the paper by Notowidigdo and Miller [67] from 2004 proposes the first offline recognition method for hand-drawn diagrams. In the paper, the authors present a recognition system, which they evaluate on scanned grayscale images of pencil-based diagrams, and a graphics editor, where the user can iteratively correct recognition errors. To reconstruct the strokes, they first apply a Gaussian filter to smooth out the noise in the image. Next, an image thresholding method converts the image into a binary image. To this end, each grayscale pixel is converted to either black or white based on a constant pixel threshold. Given the binary image, they apply standard algorithms for line detection, contour following, line splitting, and segment merging [39]. The output of this procedure is a set of line segments, which we refer to as strokes. Given the extracted strokes, the authors use standard algorithms to detect geometric shapes, including rectangles, diamonds, and circles. For detecting textblocks, they develop an algorithm that iteratively merges small line segments in proximity. Finally, for recognizing arrows, they use a modified version of an algorithm originally proposed as part of an online method [34].

Another early approach was proposed by Refaat et al. [76] in 2008, which leverages machine learning to recognize shapes. For image preprocessing, the authors use an adaptive image thresholding method [81], which performs better than thresholding with a constant pixel value. To classify candidate shapes, Refaat et al. extract a large number of features, most of which are computed as the ratio of two areas, e.g., the area of the convex hull divided by its perimeter. Next, they train a support vector machine on these features, which classifies the type of shape. Finally, they evaluate their system on hand-drawn diagrams that consists of five standard geometrical shapes, including circle, triangle, rectangle, diamond, and ellipse, which are interconnected through lines.

As mentioned in Section 3.1, due to the lack of public datasets that are offline

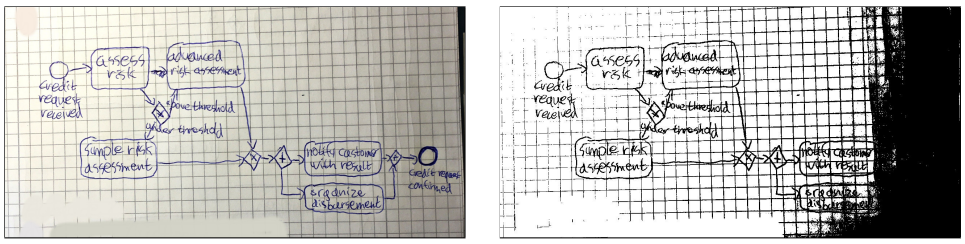
by nature, offline methods have also been evaluated on online datasets. The first paper in this line has been published by Wu et al. [103] in 2015. Wu et al. develop a recognition pipeline that receives the ground-truth strokes of the FC_A dataset as input, hereby excluding the temporal information of each stroke. From a method perspective, the main contribution of the work is a shapeness estimation algorithm, which figures out if a stroke grouping has a regular appearance. This algorithm leverages the fact that shapes of one class, e.g., a decision shape, look very similar throughout the dataset. To detect text phrases, they group all strokes inside a recognized shape as one detected text phrase. For text outside shapes, after arrow recognition, they form clusters of the remaining strokes based on a predefined distance threshold. On the FC_A dataset, their method achieves a symbol recognition accuracy of 83.2%. However, these evaluation results are not comparable with image-based recognition methods, as their evaluation does not account for stroke reconstruction errors.

In 2016, Bresler et al. published a paper with an offline diagram recognition method [9], for which they extended their previously proposed online system [11]. To this end, they use a standard image binarization step, after which they apply their online method, which they modified such that it does not require any temporal stroke information. To evaluate their method on more realistic offline images, they printed out and scanned the flowcharts from the FC_B dataset, which results in images with noise from the scanning procedure. We refer to this scan dataset as $\texttt{FC\_B}_{\texttt{scan}}$. The adapted recognizer was also tested on the unordered FC_A strokes, where it achieved a symbol recognition recall of 84.2%.

In addition to the mentioned two works that have been evaluated on online datasets, two papers have been evaluated on private datasets. In 2017, Herrera-Camara and Hammond [37] proposed a mobile phone app that allows users to take a photo of a flowchart drawn on paper, and combine this with a flowchart recognition component and a user interface where users can correct recognition errors. To reconstruct the strokes they propose a semi-automated process, where the user first crops the image to remove unrelated parts, after which a standard preprocessing pipeline identifies the strokes in the cropped image. To evaluate their approach, they performed a study with 20 subjects, where each participant solved three flowchart exercises using pen and paper.

Finally, in the business process modeling domain, Zapp et al. [113] presented a prototype to recognize shapes in EPC diagram images in 2017. Their shape recognition pipeline is based on the shapeness estimation method proposed by Wu et al. [103]. To overcome errors introduced during stroke reconstruction, the authors manually preprocessed each image to remove artifacts. The evaluation was conducted on a dataset of 108 sketched EPC diagrams.

Overall, stroke-based methods require that the strokes can be reliably extracted



(a) Input image **(b)** Otsu's thresholding method [68]

Figure 3.4: *Image thresholding example*: Otsu's method identifies the majority of background pixels, but struggles with the squared paper and the varying lighting conditions.

from a given image. A critical step in this procedure is thresholding, which generates a binary image. Thresholding has been shown to work well on online datasets [9, 103] or scanned drawings with a simple structure (black strokes on white background) [9, 67]. However, reliable stroke reconstruction from camera-based images of pen-and-paper drawings with complex backgrounds is still an unsolved research topic [5, 21]. This is illustrated in Figure 3.4, which applies the popular Otsu's method [68] to an image from the diagram dataset introduced in Chapter 4. The next section reviews object-based methods, which do not require an explicit stroke reconstruction step.

3.3.2 Object-based Methods

Given the success of deep learning methods in computer vision (cf. Chapter 2), in recent years such methods have also been applied in the diagram domain. We refer to these methods as object-based methods, as they commonly use object detectors to detect the diagram elements in a given image.

The first object-based method has been published by Julca-Aguilar and Hirata in 2018 [41]. In their paper, the authors evaluate object detectors as a general method to detect symbols in hand-drawn graphics. As part of their work, they train Faster R-CNN [77] to detect diagram shapes, arrows, and textblocks in rendered images of the `FC_A` dataset. The evaluation results reported in the paper are very impressive and indicate the potential of using object detectors for recognizing diagrams. In particular, the detector achieved a mAP@0.5 of 97.7%, where mAP@0.5 corresponds to the mean average precision (mAP) at a ground-truth bounding box overlap of at least 50%. Due to differing evaluation metrics, the results in the paper are not comparable to the stroke-based approaches mentioned in the previous section, though. In general, while object detectors can localize arrows through bounding boxes, they are not able to identify the shapes that each arrow connects.

We address this research gap in Chapter 5 and the chapters beyond, where we propose several methods that can properly recognize edges.

In 2022, Fang et al. [26] proposed DrawnNet, a CNN-based keypoint detector, which improves our Arrow R-CNN method proposed in Chapter 5. DrawnNet is based on CornerNet [50], which detects an object bounding box as a pair of keypoints, the top-left corner, and the bottom-right corner, using a single CNN. In the paper, Fang et al. extend CornerNet with a network branch for arrow orientation prediction, which predicts the head and tail keypoints of each arrow. In addition, they propose two new keypoint pooling modules, which take into account the geometrics aspects within diagrams. During training, Fang et al. reuse the Arrow R-CNN image augmentation pipeline. In addition, they use the same post-processing as Arrow R-CNN, which employs an NMS configuration tailored to hand-drawn diagrams, and which identifies the shapes that each arrow connects as the shapes closest to the respective arrow keypoints. Fang et al. evaluate their method on the `FC_A`, `FC_B`, and `FA` datasets, for which they use the rendered diagrams and corresponding annotations that we have made publicly available.[1] On the mentioned datasets, they consistently improve over the previous state-of-the-art Arrow R-CNN method. However, as mentioned in Section 1.1 and Section 3.2, the existing online datasets are not particularly challenging for modern deep learning methods. On all three datasets, Arrow R-CNN achieves mean symbol recognition precision and recall scores in the range of 97.9% to 99.3%. The results show that there is little room for improvement on the existing datasets, as the marginal precision and recall improvements obtained by DrawnNet also indicate, which are in the range between 0.1% and 0.5%.

Finally, two more methods that target the recognition of computer-generated diagrams have been proposed in 2022. First, Sun et al. [91] published the flowchart recognition method FR-DETR. FR-DETR is based on DETR [13], a transformer-based object detection approach. In addition, it is inspired by LETR [108], which extends DETR for line segment detection to predict wireframes of indoor and outdoor environments. FR-DETR fuses both DETR and LETR into a network with a joint CNN backbone. In this architecture, the shapes of each flowchart are detected with the DETR object detector, and the line segments of each edge are detected with the LETR line segment predictor. Sun et al. evaluate their approach on the FR-DETR dataset, which combines patent flowcharts from the CLEF-IP dataset with flowcharts collected through image search engines. The FR-DETR approach assumes that each edge can be decomposed into a sequence of line segments. This is the case for computer-generated diagrams, where edges are typically composed of vertical and horizontal line segments that are parallel to an axis. However, the

[1] `https://github.com/bernhardschaefer/handwritten-diagram-datasets`

Table 3.2: *Related work in offline diagram recognition*: For each approach, we show the method type, the targeted diagram type, and its recognition scope with respect to diagram shapes and edges. For diagram labels, we indicate the subtasks that each approach targets.

Method Type / Authors	Diagram Type	Shapes	Edges	Labels*		
				L1	L2	L3
Stroke-based approaches						
Notowidigdo & Miller [67]	Generic geometric shapes	✓	✓			
Refaat et al. [76]	Generic geometric shapes	✓	✓			
Wu et al. [103]	Online flowcharts	✓	✓			
Bresler et al. [9]	Online flowcharts	✓	✓	✓		
Herrera-C. & Hammond [37]	Flowcharts	✓	✓	✓		
Zapp et al. [113]	EPC process models	✓				
Object-based approaches						
Julca-Aguilar & Hirata [41]	Online flowcharts	✓		✓		
Fang et al. [26]	Online flowch., finite autom.	✓	✓	✓		
Sun et al. [91]	Comp.-gen. flowcharts	✓	✓			
Antinori et al. [1]	Comp.-gen. BPMN models	✓	✓	✓	✓	✓

* L1 = Textblock detection (identifies which parts of the image contain text)
L2 = Textblock handwriting recognition (recognizes the handwritten text contained within each textblock)
L3 = Textblock relation detection (identifies the shape or edge that each textblock labels)

edges of hand-drawn diagrams often consist of curves, which can not be effectively modeled through line segments. Thus, it is unclear to what extent the FR-DETR approach can be applied to hand-drawn diagrams.

Second, Antinori et al. [1] proposed the BPMN-Redrawer method for recognizing computer-generated BPMN diagrams in 2022. From an architectural perspective, BPMN-Redrawer is very similar to Arrow R-CNN [82] but builds on two different neural networks, one to recognize shapes, and one to recognize arrows. The shapes are recognized using the Faster R-CNN object detector. For recognizing edges, they first detect arrow objects and their keypoints using the Mask R-CNN keypoint detector [35]. Next, they use the heuristic of Arrow R-CNN to identify the shapes that each arrow connects. For textblock detection and handwriting recognition, they build on the Tesseract OCR engine, which can only recognize machine-printed characters. Last, for textblock relation detection, they simply assign each label to the closest shape or edge. Antinori et al. evaluate their method using two off-the-shelf networks from the Detectron2 library [104], which they train on a dataset of computer-generated BPMN model images. Given that they use off-the-shelf networks, it is very straightforward to reimplement their approach. Therefore, in Chapter 8, we evaluate the BPMN-Redrawer approach on hand-drawn diagrams and compare the results to the performance of our `Sketch2Process` approach.

3.4 Summary

This chapter reviewed related work in diagram recognition. We first provided an overview of diagram recognition and explained the difference between online and offline methods. Next, we discussed the existing diagram datasets. Finally, we surveyed related work that addresses offline diagram recognition, which we summarize in Table 3.2. Overall, we identify the following research gaps:

Dataset limitations. From a dataset perspective, we observe that there are no offline datasets of hand-drawn diagrams. The existing works, therefore, out of necessity, evaluate their methods on images generated from online datasets. However, the generated images do not capture the challenges associated with recognizing camera-captured diagrams sketched on paper. In fact, the images have a very simple structure (black strokes on white background), and as the drawings are template-based, they do not have the chaotic structure of diagrams that stem from conceptual modeling tasks. As another issue, we find that the training and test set diagrams look very similar in most existing datasets, which we illustrated in Figure 3.3. This means that the evaluation results obtained on these datasets are biased, and do not indicate how the model performs on unseen data. In summary, we argue that more realistic datasets are needed to advance offline diagram recognition research.

Method limitations. From a method perspective, we observe that existing works are limited in several aspects. For stroke-based methods, we find that these are not applicable to camera-based images of drawings with complex backgrounds, due to the lack of accurate stroke reconstruction methods for this domain. For object-based methods, we observe that most approaches either target the recognition of computer-generated diagrams [1, 91] or that they evaluate their method on online datasets [26, 41]. Therefore, it remains unclear to what extent these methods can recognize camera-captured images of pen-and-paper drawings. In addition, we observe that no approach, neither stroke-based nor object-based, fully addresses the recognition of hand-drawn diagram labels. In fact, except the BPMN-Redrawer by Antinori et al. [1], no method goes beyond textblock detection. As for the BPMN-Redrawer, this approach uses the Tesseract OCR engine, which does not recognize handwriting, and it uses a very simple heuristic for textblock relation detection. As the corresponding paper does not evaluate the performance of the label recognition component, it is unclear how this part performs, even for computer-generated diagrams. In summary, this means that, from a practical point of view, the existing approaches are only of limited use.

Chapter 4

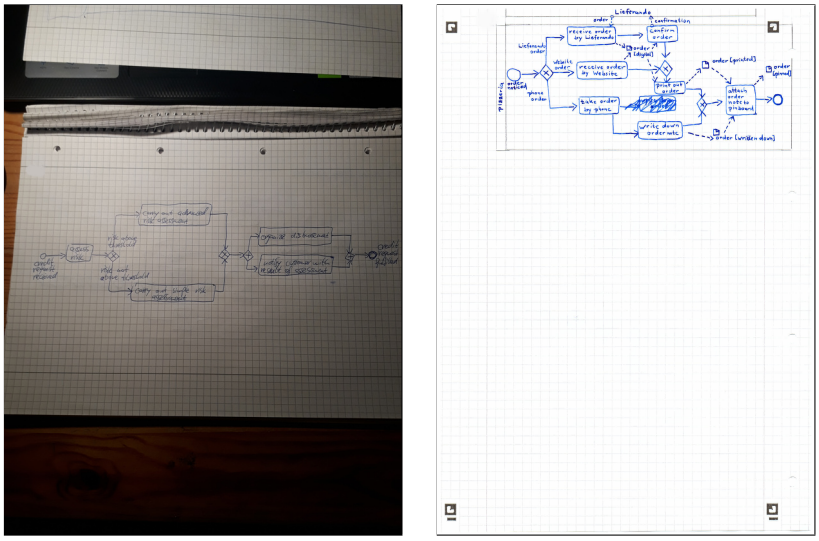
The hdBPMN Dataset

This chapter discusses hdBPMN, the dataset that we established as part of our work. Overall, the dataset contains 704 annotated images of hand-drawn BPMN models, corresponding to 11 modeling tasks, and containing over 70,000 annotated elements which cover 25 different types of shapes, edges, and labels. The images and BPMN annotations are publicly available at https://github.com/dwslab/hdBPMN. Furthermore, we provide a Python library to convert the images and BPMN annotations into a dataset that uses the COCO format [58]: https://github.com/dwslab/pybpmn-parser. COCO is a popular serialization format for object and keypoint detection datasets in computer vision research and is thus supported by common object detection libraries.

The chapter is structured as follows. Section 4.1 discusses the collection procedure of the dataset, i.e., how we obtained the 704 images of diagram sketches. Section 4.2 explains how we annotated each image in the dataset. Given the images and annotations, we dive into various dataset characteristics in Section 4.3, and detail how we split the dataset into a training, a validation, and a test set in Section 4.4. Finally, Section 4.5 illustrates the recognition challenges associated with camera-captured BPMN sketches.

4.1 Collection Procedure

We collected 704 images of hand-drawn BPMN models from 107 participants, all students at the University of Mannheim who participated in a lecture on business process modeling and agreed to the use of their drawings. Each image corresponds to a solution that was submitted by a student for a graded assignment in an exercise sheet or exam. The obtained models stem from 11 modeling tasks, 10 of which involved the establishment of a BPMN model based on a textual process descrip-



(a) ex01_writer0073 **(b)** ex06_writer0045

Figure 4.1: hdBPMN images where the diagram only covers a small portion of the image.

tion, while the other involved the conversion of a Petri net into a BPMN model. All modeling tasks and accompanying instructions are available in Appendix B.

The students were asked to solve the exercise sheets and the exams on paper, and then upload a solution PDF that contains a scan or photograph of each handwritten page, with the only constraint that the models should be readable. For each received submission, we used the `pdfimages` command-line utility to extract the images from the PDFs. Next, we split images manually into multiple ones when they spanned different modeling tasks. Note that, aside from splitting pages that covered multiple modeling tasks, we deliberately did not crop images, which means that the drawn diagram sometimes only covers a small fraction of the image. Two such cases are shown in Figure 4.1, where the percentage of the image area that is covered by the diagram is the lowest among the entire dataset.

Finally, we used an image editor to conceal personal details (e.g., names and student IDs) and task-related meta information (e.g., exercise number). The resulting images were assigned names that follow a `taskID_participantID` convention to recognize drawings by the same participant.

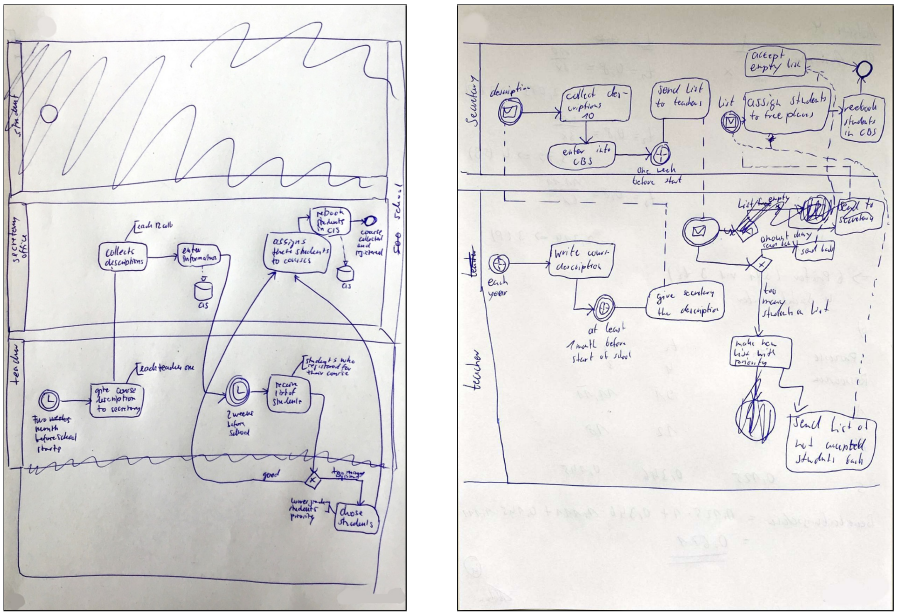
Table 4.1 shows the number of collected images per modeling task and some basic task properties. ex00 and ex01 are easy modeling tasks that stem from the first exercise sheet, where the students received a short textual process description and were instructed to only use some basic BPMN elements to model the corresponding process. The tasks ex02 to ex05 also originate from the first exercise sheet and

Table 4.1: Modeling task characteristics

Task ID	Images	Difficulty	Source	Collab. Shapes	Data Shapes
ex00	68	Easy	Sheet 1		
ex01	69	Easy	Sheet 1		
ex02	69	Intermediate	Sheet 1	✓	✓
ex03	65	Hard	Sheet 1	✓	✓
ex04	18	Intermediate	Sheet 1	✓	✓
ex05	22	Hard	Sheet 1	✓	✓
ex06	98	Intermediate	Sheet 2	✓	✓
ex07	98	Hard	Sheet 2	✓	
ex08	85	Hard	Exam 1	✓	
ex09	16	Hard	Exam 2	✓	
ex10	96	Hard	Sheet 2	✓	

include the modeling of collaboration shapes (pools and lanes) and data shapes (data objects and data stores). Among those, ex04 and ex05 were optional tasks, which explains the smaller number of collected images. For the first exercise sheet, about a third of the submitted solutions were drawn on a tablet, which we excluded from the dataset. Therefore, the collected number of images for each modeling task from ex00 to ex03 is substantially lower than the number of total participants. This is not the case for the second exercise sheet and the exam, where only diagrams drawn on paper were allowed.

In the lecture before the second exercise sheet, the students were introduced to more advanced BPMN elements such as timer events, terminate end events, and event-based gateways. This was reflected in the second exercise sheet, where the students were required to use a larger subset of the BPMN vocabulary. In ex06, the task was to translate a Petri net into a BPMN model, and ex07 and ex10 were among the most difficult modeling tasks with a relatively long textual process description. Finally, we also collected hand-drawn diagrams from the two exam questions ex08 and ex09. As the exam modeling tasks come with a severe time constraint, we noticed that the hand-drawn diagrams are much more chaotic than their exercise sheet counterparts. As illustrated in Figure 4.2, the exam images contain many elements that have been crossed-out and often contain long-range arrows that cross other elements. Thus, the exam modeling tasks can be considered the most challenging from a recognition perspective.



(a) ex08_writer0037 **(b)** ex08_writer0100

Figure 4.2: hdBPMN exam images with crossed-out elements and long-range arrows.

4.2 Annotation

In order to train and evaluate diagram recognition approaches, we need to annotate the hand-drawn elements in each collected image. In the following, we present the image annotation tool that we developed to annotate the BPMN elements in each image. We also outline our strategy to reduce the number of annotation mistakes. Finally, we outline how we annotated all handwritten words in each image to evaluate the handwriting recognition accuracy of our approaches.

BPMN image annotation tool. To annotate the BPMN elements in each image, we developed an image annotation tool based on the open-source `bpmn-js` BPMN viewer and editor[1], which we have made publicly available: `https://github.com/dwslab/bpmn-image-annotator`. Figure 4.3 depicts the annotation tool in action. The annotation workflow for each collected image works as follows. First, the image is uploaded into the annotation tool, where it is resized to a user-defined image width. Next, the annotator needs to manually model the process on top of the image, hereby ensuring that the shapes, edges, and labels match their hand-drawn counterpart. Note that to allow us to also annotate images that contain

[1] `https://github.com/bpmn-io/bpmn-js`

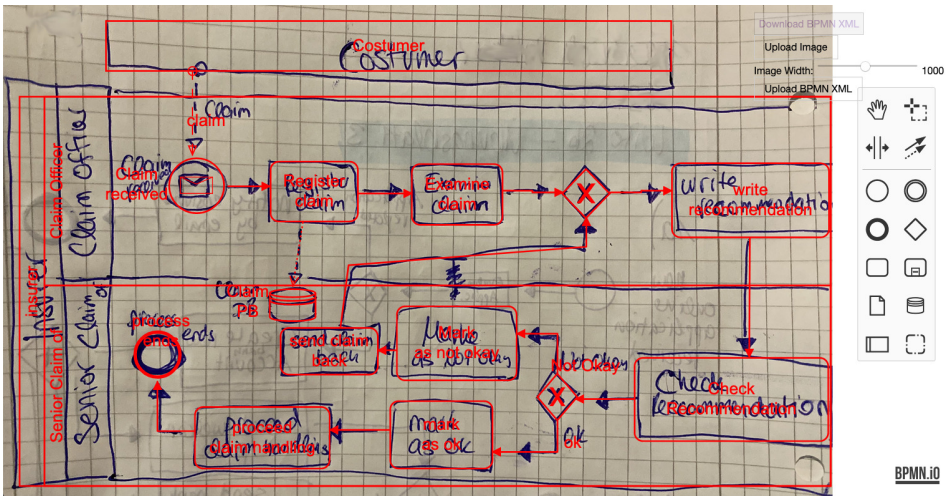


Figure 4.3: Example of an annotated sketch in the *BPMN image annotator*. Shapes are sized and positioned to match their hand-drawn counterparts, while edges are modeled using waypoints to resemble the drawn arrows.

modeling errors, we allow our annotation tool to violate certain correctness rules enforced by `bpmn-js`, e.g., an *end event* with an outgoing *sequence flow*. Upon completion, the annotations can be downloaded as a *BPMN 2.0 XML* file, which follows the standard serialization format for BPMN models. The BPMN XML format specifies the bounding box of each shape and label. For edges, the path is annotated using at least two waypoints, where the first waypoint intersects with the source shape, and the last waypoint intersects with the target shape. Since the image width during annotation is exported as a comment into this file, the location and size of each element in the BPMN model can be linked back to the location and size in the image.

Quality assurance. Due to budget and time constraints, each image was annotated by only one annotator. When analyzing the errors of the first trained models, we observed two main sources of annotation errors: missing edges and incorrect event types. To detect such errors, we compiled a large list of syntactical checks and inspected each image manually in case a syntax check was violated. Concerning missing edges, the syntactical rules that we inspected upon violation include:

- *Sequence flow*: Activities, gateways, and intermediate events should have at least one incoming and one outgoing sequence flow. Start events should have no incoming and one outgoing sequence flow, and vice versa for end events.
- *Message flow*: Catching message events should have an incoming message

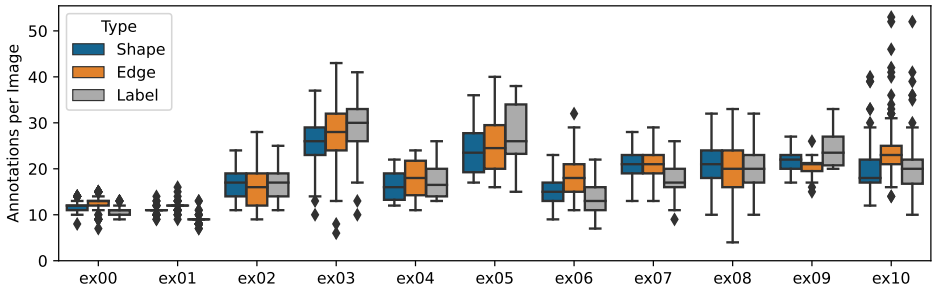


Figure 4.4: Modeling task annotation statistics

flow, and throwing message events should have an outgoing message flow.

- *Data association*: Data object and data stores should have at least one incoming or outgoing data association.

We found that these rules are not just effective for detecting missing or incorrect edges, but also to identify events where an incorrect subtype was used, e.g., an intermediate event annotated as a start event. For message events, we additionally checked for inconsistencies between the label and the existence of in- and outgoing message flow. For example, we inspected message events with an outgoing message flow whose label contained the word "receive".

Word annotation. To evaluate the handwriting recognition accuracy of our approaches, we also annotated all handwritten words in each image. To bootstrap the annotation process, we obtained a first set of words per image using the Microsoft Azure OCR service[2] version 2021-04-12. Next, we imported the initial word annotations into the *Label Studio* [95] annotation tool, where an annotator went over each image and corrected the mistakes of the OCR service. To reduce word annotation errors, we generated a text phrase from all words located within a label and compared this extracted text phrase to the annotated text of the label. We then manually inspected the word and label annotations in case of mismatches. Note that we did not resolve all mismatch cases, as sometimes the handwriting was so difficult to decipher that multiple word annotations were plausible. Finally, we exported the annotated words of each image using the Pascal VOC annotation format [25].

[2] https://docs.microsoft.com/azure/cognitive-services/computer-vision/overview-ocr

Table 4.2: BPMN elements in the 704 annotated images

Type	Group	Elements and their frequencies
Shape	Activity	task (4,094), subprocess (collapsed) (135), subprocess (expanded) (5), call activity (15)
	Event	start (424), intermediate throw (7), end (936), terminate (62) message start (508), message intermediate catch (507), message intermediate throw (291), message end (199), timer start (86), timer intermediate catch (289)
	Gateway	exclusive (1,347), parallel (661), inclusive (3), event-based (171)
	Resource	pool (1,103), lane (688)
	Data element	data object (887), data store (219)
Edge		sequence flow (9,893), message flow (1,821), data association (1,773), annotation association (170)
Label		textblock (12,498), word (31,997), text annotation (176)

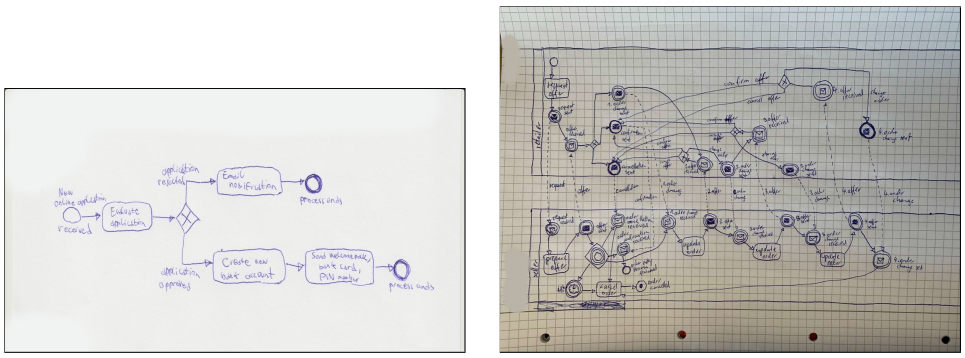
4.3 Dataset Characteristics

Overall, the 704 images in the hdBPMN dataset contain more than 70,000 annotated elements. On average, each image has more than 100 annotations, including 18.2 shape, 19.4 edge, 17.8 label, and 45.5 word annotations. As shown in Table 4.2, the models in the dataset are highly expressive, spanning 22 types of shapes and 4 types of edges. The different shape types can be further categorized by groups, which include 4 types of activity shapes, 10 types of events, and 4 types of gateways.

Largely owing to the different modeling tasks from which they stem, the individual BPMN models differ in terms of their size, complexity, and expressiveness (i.e., the number of types covered). The models resulting from the 11 different modeling tasks have up to 15 activities and 26 events, 0 to 10 gateways, 0 to 8 resources, and 0 to 15 data elements. As illustrated in Figure 4.4, some tasks result in simpler models (e.g., ex01: 11.0 shapes and 11.9 edges on average) and others in more complex ones (e.g., ex03: 25.8 shapes and 27.3 edges). To illustrate the model variety in the dataset, Figure 4.5 shows the images with the smallest and largest number of annotated BPMN elements, respectively.

Aside from the complexity of a particular process, the recognition difficulty of an image is affected by various other aspects, mainly corresponding to the *general challenges* highlighted in Section 4.5, such as the use of different paper types (blank, lined, or squared) and drawing implements (pen versus pencil), image-capturing issues (such as background objects, cut-off parts, and blurriness), the inclusion of crossed-out parts, and the presence of modeling errors.

As a result, the 704 images in the publicly-available hdBPMN dataset thus de-



(a) ex00_writer0041 **(b)** ex10_writer0024

Figure 4.5: hdBPMN images with the smallest and largest number of annotated BPMN elements, respectively.

pict BPMN models that span a broad range of BPMN elements and have a high degree of diversity. Figure 4.6 visualizes this by showcasing some of the different manners in which various kinds of shapes were drawn.

4.4 Dataset Splits

Following existing datasets of hand-drawn diagrams [11,27], we split up the dataset into training, validation, and test parts, which we release together with the dataset. As illustrated in Figure 4.7, each participant in the dataset contributed between one and ten diagrams. While the variability of factors such as writing style, writing medium and image-capturing method is high between participants, there are substantial similarities between the diagrams of one participant. Therefore, we split the dataset by participants, such that the participants in the training, validation, and test set are disjoint. Specifically, we created a random 60%/20%/20% split over the participants and assigned each diagram to the respective part. The resulting training/validation/test set contain 432/144/128 diagrams from 65/21/21 participants, respectively.

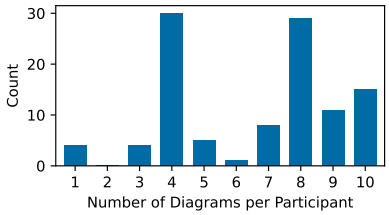


Figure 4.7: hdBPMN diagrams per participant

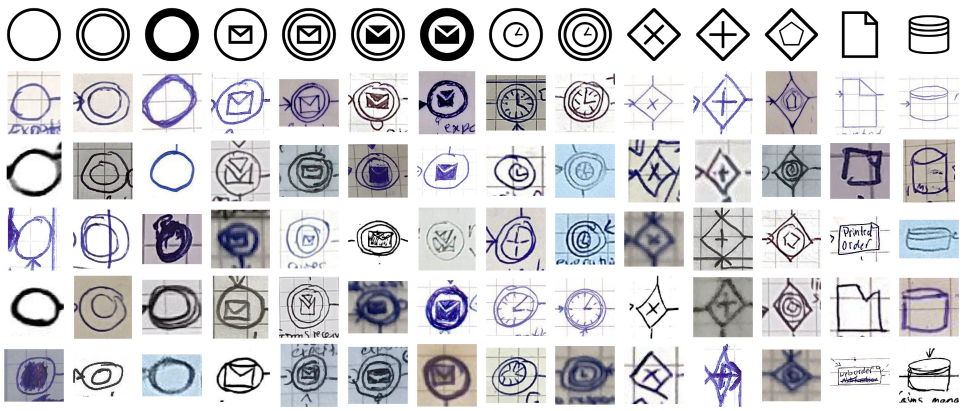


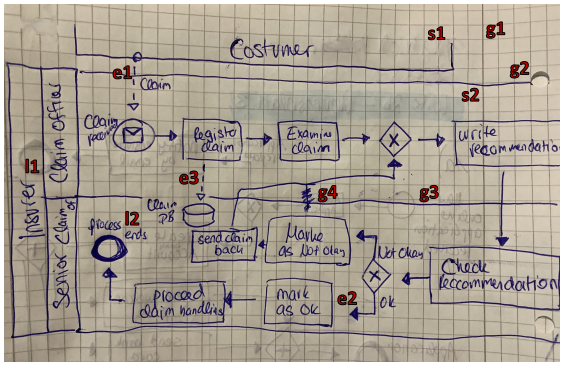
Figure 4.6: Examples of hand-drawn *events* (start 4.6a, intermediate 4.6b, end 4.6c, message start 4.6d, message intermediate catch 4.6e, message intermediate throw 4.6f, message end 4.6g, timer start 4.6h, timer intermediate 4.6i), *gateways* (exclusive 4.6j, parallel 4.6k, event-based 4.6l) and *data elements* (data object 4.6m, data store 4.6n).

4.5 Recognition Challenges

This section illustrates the challenges associated with the recognition of hand-drawn BPMN models in detail, which we have already briefly touched upon in Section 1.1. We discuss challenges specifically related to shapes, edges, and labels, as well as general challenges that occur when dealing with images of physical drawings. We will use the exemplary drawing in Figure 4.8, stemming from our `hdBPMN` dataset, to illustrate the challenges where applicable.

Shape recognition challenges. Shape recognition targets the identification of the nodes in a BPMN model, such as activities, events, gateways, resource pools, and data objects. From a recognition perspective, shapes are defined through a *bounding box*, capturing the location of the shape in the drawing, and a *shape type*, capturing its type. Compared to the recognition of other types of conceptual models (see Chapter 3), BPMN models have a considerably higher number of different node types, which increases the complexity of the recognition task.

The recognition of shapes and their types in hand-drawn diagrams can be highly complex due to a variety of challenges. From a conceptual point of view, shape recognition in BPMN is complex because of the high similarity between certain types. Clear examples of this are events, which are depicted as circles, where the circle's line determines whether it is a start (single line), intermediate (double), or end (bold) event of a certain type. Similarly, activities, sub-processes, pools, and



Depicted shape recognition issues:
s1) Shape drawn incompletely;
s2) Shape drawn using curved and interrupted lines.
Depicted edge recognition issues
e1) Edge interrupted by another model element;
e2) Edge not connected to corresponding nodes;
e3) Data association drawn using dashes rather than dots.
Depicted label recognition issues
l1) Rotated pool and lane labels;
l2) Textblock words that appear disconnected.
Depicted general issues:
g1) Additional lines due to paper type;
g2) Punch-holes in paper obscuring the model;
g3) Ink on other side of paper bleeding through;
g4) Crossed out part of the drawing.

Figure 4.8: Exemplary hand-drawn BPMN model with various recognition challenges.

lanes are all depicted as rectangles, where the specific node type follows from the node's context, e.g., a lane encompassing several activities, or subtle differences, e.g., a small *plus* symbol in a square indicates that a node represents a sub-process.

While challenging in itself, differentiating among such similar node types becomes more complex due to the characteristics of hand-drawn models, such as drawn lines being incomplete, curved (when they should be straight), or interrupted, such as e.g., seen for issues s1 and s2 in Figure 4.8. Furthermore, shapes in general are drawn in a broad range of styles, especially more complex ones such as events, databases, and certain gateways. This is, for instance, evidenced by the examples of intermediate (throwing) message events depicted in Figure 4.6f.

Edge recognition challenges. The edges in BPMN models indicate the connections between nodes. There are three types of edges, which have different drawing styles and are used to connect different node types. Solid edges indicate the *control flow* of a process by connecting nodes such as activities, events, and gateways, as e.g., seen in Figure 4.8 to indicate that the *Examine claim* activity occurs after *Register claim*. Dashed edges capture the *message flow* across organizational boundaries, as seen by the *Customer* sending a claim to the *Insurer* in the example. Finally, dotted edges capture *data associations*, showing the creation or retrieval of data, such as *Register claim* storing information in the *Claim DB*.

Each edge is defined through a sequence of waypoints (indicating the path of the edge), the edge type, and the source and target shape that the edge connects. The proper recognition of edges and their characteristics in an automated manner is complex, though. This complexity, for instance, results from edges commonly crossing each other or intersecting with other model elements, as e.g., seen for issue e1 in Figure 4.8, where just a single dash appears before the edge intersects with the *Insurer* pool. Such edge interruptions make it hard for a recognition approach

to identify which drawn lines belong to the same edge and which lines are actually separate ones. Furthermore, drawn edges are often not properly connected to model nodes, as seen for issue e2 in Figure 4.8, which makes it harder to recognize the source and target of an edge.

This recognition is particularly complex when there are multiple candidate shapes, such as seen in Figure 4.9, where two pool boundaries and a task are very close to the starting point of a message flow. Finally, the differentiation of message flows and data associations can be difficult since they are often drawn in a similar or even equal manner. For example, in Figure 4.8 we observe that also data associations are drawn using dashes, rather than dots (see issue e3).

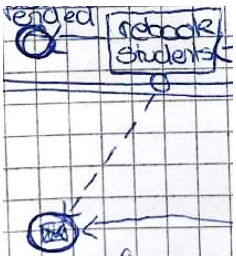


Figure 4.9: Multiple candidate shapes

Label recognition challenges. Labels are associated with shapes and edges in BPMN models. Some types, e.g., activities, require a label, whereas for others, e.g., control flow edges, labels are optional. Each label, in contrast, should be associated with a specific shape or edge. As described in Chapter 3, label recognition can be decomposed into a sequence of three steps.

Textblock detection strives to locate the labels in BPMN models through bounding boxes. These boxes are referred to as *textblocks* in diagram recognition [9, 11]. The primary challenge here is to appropriately recognize which pieces of text in a diagram belong together, i.e., which form a single textblock. This can be highly complex, since it may be hard to discern which words actually belong together, for instance, because they are apart from each other in the drawing (see e.g., issue l2) or even separated by (parts of) model shapes. The exception to this are activity labels, where we know that the label consists of the words located within the activity bounding box. Therefore, activity labels do not need to be detected through dedicated textblocks.

Textblock handwriting recognition aims to recognize the exact text that is contained within a textblock, i.e., to interpret the handwritten text. While handwriting recognition (HWR) methods for handwritten documents have been developed for decades, HWR for hand-drawn diagrams is largely unexplored and much more challenging [5], as textblocks can be rotated (see issue l1), in front of complex backgrounds, and with overlapping model element strokes.

Finally, *textblock relation detection* is concerned with finding the shape or edge that the textblock labels. This can be challenging when multiple shape or edge candidates are in close proximity to a textblock, which makes it hard to recognize the correct relation between the

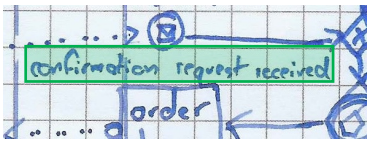


Figure 4.10: Label candidates

textblock and the model element. To give an example, the textblock in Figure 4.10 has three shapes and four edges nearby.

General challenges. The complexity of detecting shapes, edges, and labels due to the aforementioned issues is amplified by various general challenges related to sketch recognition for hand-drawn models, such as:

- The paper on which a hand-drawn model was drawn may be lined, squared, or dotted. For instance, the graph paper in Figure 4.8 adds numerous additional lines to the drawing, which may appear similar to lines used to denote model elements, such as resource pools (consider the thicker line denoted by issue g1).
- The image might contain additional contents that are visually similar to model elements. For example, Figure 4.8 has punched holes (g2), which look similar to events, and visible model elements from the back side of the paper (g3).
- Drawing implements, such as pencils, may affect the clarity, consistency, and thickness of drawn lines, which can negatively affect the interpretability of a sketch. While Figure 4.8 was drawn using a clear pen, a pencil would yield lines that look very similar to the lines that are already part of the paper (e.g., as indicated by g1).
- When hand-drawn models are captured using a camera, rather than a dedicated scanner, additional quality issues may be introduced [21]. This includes images that are rotated or blurry, as well as those that include content beyond the paper or where part of it has been cut off (e.g., the right-hand side of Figure 4.8).
- Finally, it is important to recognize that establishing BPMN models is notoriously difficult [15], which means that it is not uncommon for modelers to make mistakes [54]. On the one hand, this can result in parts of a drawing being crossed out (issue g4), on the other hand, there is no guarantee that the final drawing is free of errors, which means that a recognition approach cannot depend on the syntactic correctness of the drawing.

In the next chapters, we propose several methods that aim to address the aforementioned challenges, which we develop to accurately detect hand-drawn BPMN models in images.

Chapter 5

Arrow R-CNN

In the diagram domain, Julca-Aguilar and Hirata were the first to use an object detector to detect the symbols (shapes, arrows, textblocks) of hand-drawn diagrams [41]. In their work, the authors demonstrate that Faster R-CNN can be effectively trained to detect symbols in a small flowchart dataset. Even though the utilized dataset has only 200 training images, the evaluation shows near-perfect shape recognition results, and the model also performs well in recognizing arrow and textblock objects, which are more challenging due to their varying form and size. However, object detectors are severely limited concerning edge recognition: while a detector can localize arrow objects in a diagram through bounding boxes, it can not identify the source and target shape of an arrow. Recognizing the benefits and limitations of using object detectors for hand-drawn diagram recognition, in this chapter we propose the `Arrow R-CNN` method for recognizing arrow-connected diagrams. `Arrow R-CNN` extends the Faster R-CNN object detector with an arrow keypoint detector, and it uses a structure recognition method to form the final diagram from the detected objects and keypoints. As part of structure recognition, the detected objects and keypoints are used to identify the shapes that each arrow connects. In addition, we propose an image augmentation pipeline tailored to hand-drawn diagrams, and we demonstrate that this greatly improves the object detection results.

The research presented in this chapter is based on a paper titled "*Arrow R-CNN for handwritten diagram recognition*" by Bernhard Schäfer, Margret Keuper, and Heiner Stuckenschmidt [82].

The remainder of this chapter is organized as follows. Section 5.1 describes the `Arrow R-CNN` method and its training procedure. Section 5.2 details the datasets that we use to evaluate our approach. Section 5.3 describes the evaluation setup and the experimental results. Finally, Section 5.4 presents conclusions and limitations.

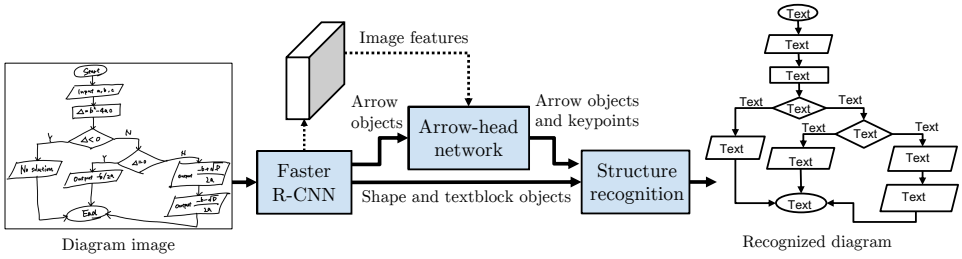


Figure 5.1: `Arrow R-CNN` overview: Given an image, *Faster R-CNN* detects arrow, shape, and textblock objects. The *arrow-head network* uses the arrow bounding boxes and image features to predict the arrow head and tail keypoints. Last, the *structure recognition* pipeline constructs the final diagram given the predicted objects and arrow keypoints.

5.1 The `Arrow R-CNN` Method

This section introduces `Arrow R-CNN`, our method for recognizing the shapes, edges, and textblocks in a given diagram image. As visualized in Figure 5.1, `Arrow R-CNN` consists of three components. In the first step, we use *Faster R-CNN* [77] to detect the arrow, shape, and textblock objects in an image. Next, we extend Faster R-CNN with an *arrow-head network*, which detects the head and tail keypoints of each arrow. Last, our *structure recognition method* identifies the shapes that each arrow connects and creates the final recognized diagram. In the following, we provide details on the individual steps of our approach (Section 5.1.1 to 5.1.3). Finally, Section 5.1.4 details how we jointly train all network components.

5.1.1 Faster R-CNN

We use Faster R-CNN to detect all diagram objects (shapes, arrows, and textblocks) in a given image. Specifically, we use Faster R-CNN with the *Feature Pyramid Network (FPN)* [57] extension, as our initial experiments showed that this leads to consistently better results. A detailed description of Faster R-CNN, including its different components and the mentioned FPN extension, can be found in Chapter 2.

Figure 5.2 shows the output of the trained Faster R-CNN object detector. As indicated, the object detector has detected the shape, arrow, and textblock instances in the image. The detected bounding boxes and predicted classes are sufficient to define the diagram shapes. However, the bounding box of an arrow does not always indicate which shapes the arrow connects, as there can be more than two shapes in proximity to an arrow. In addition, the bounding box does not indicate the direction of the arrow. For example, the bounding box of the detected arrow in the

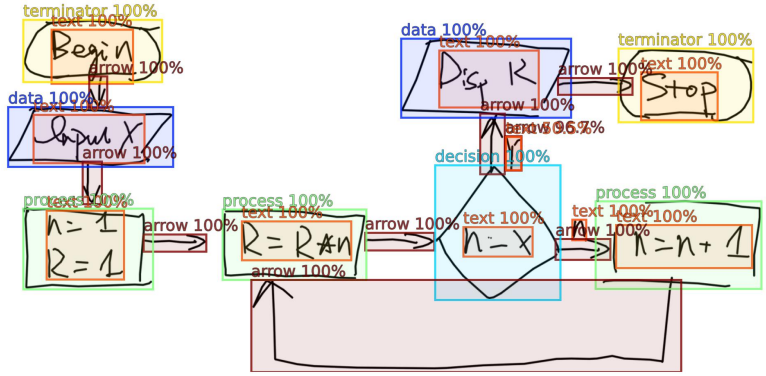


Figure 5.2: Object detection output

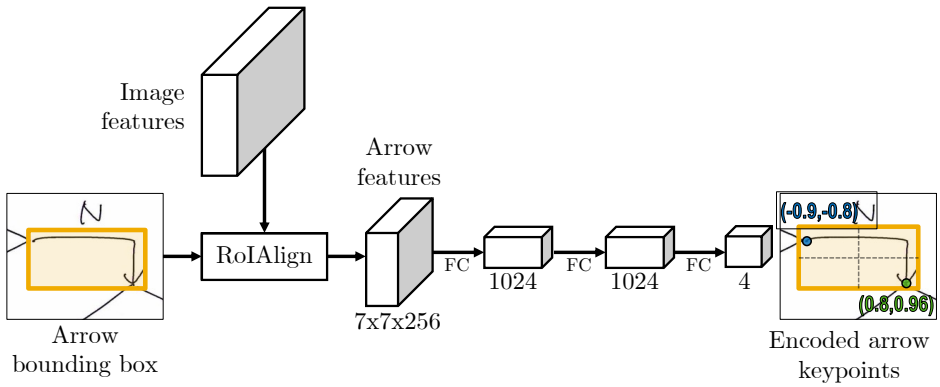


Figure 5.3: Arrow-head network architecture

top-right corner of Figure 5.2 does not indicate if the arrow connects the data to the terminator shape or vice versa. To identify the source and target shapes of each arrow, we extend Faster R-CNN with an arrow-head network, which we describe in the next section.

5.1.2 Arrow-head Network

Figure 5.3 shows the arrow-head network, which predicts the head and tail keypoints of each detected arrow. Given the Faster R-CNN image features, the head network uses *RoIAlign* to extract the arrow features that correspond to the arrow bounding box. Next, two fully-connected layers (FC) with ReLu activation function project the features to a vector. Last, a final regression layer predicts the head and tail keypoints relative to the bounding box. The network architecture of the

arrow-head network is identical to the box-head network of Faster R-CNN, except for the last layer. In fact, we implement `Arrow R-CNN` such that both head networks share the weights of the intermediate fully-connected layers, which introduces an additional regularization effect.

Given the predicted bounding box of each arrow, the arrow network regresses both arrow keypoints as a vector $(\mathbf{k}^{\text{head}^\top}, \mathbf{k}^{\text{tail}^\top})^\top$ of size 4 from the intermediate arrow feature representation of size 1024. In theory, we could directly use the absolute arrow keypoint pixel coordinates as regression targets. However, this would require the arrow features to capture the global image context, since the network would have to predict not only where the keypoints are located relative to the proposal bounding box, but also where they are located within the overall image. The Faster R-CNN bounding box regression thus encodes the bounding box regression targets relative to the proposal box. For arrow keypoint regression, we follow a similar strategy and encode the arrow keypoint targets relative to the arrow bounding box. Suppose we have an arrow bounding box $\mathbf{b} = (c_x, c_y, w, h)^\top$ with center point $\mathbf{c} = (c_x, c_y)^\top$, width w and height h, where $4wh$ measures the area of $\mathbf{b}$. For a ground truth arrow keypoint $\mathbf{k} = (k_x, k_y)^\top$, we define bounding box normalized keypoints as

$$\bar{\mathbf{k}} = \left(\bar{k}_x, \bar{k}_y\right)^\top = \left(\frac{k_x - c_x}{w}, \frac{k_y - c_y}{h}\right)^\top. \tag{5.1}$$

Thus, $\bar{k}_x$ and $\bar{k}_y$ are within the range $[-1.0, 1.0]$, for keypoints $\mathbf{k}$ contained in bounding box $\mathbf{b}$. During training, our arrow regression target is then the 4-d vector $\mathbf{t} = \left(\bar{\mathbf{k}}^{\text{head}^\top}, \bar{\mathbf{k}}^{\text{tail}^\top}\right)^\top$, representing the relative coordinates of the two keypoints per arrow. Section 5.1.4 details how we compute an arrow loss from the predicted and ground-truth encoded keypoints. During inference, we compute the absolute keypoint location of each arrow by applying the inverse encoding operation.

The next section explains how we use the detected objects and arrow keypoints to form the final recognized diagram.

5.1.3 Structure recognition

The standard Faster R-CNN post-processing method has a major downside for recognizing symbols in diagrams: it does not consider any domain knowledge about the structure of diagrams. In this work we design a post-processing structure recognition method that takes into account the following spatial and structural observations about hand-drawn arrow-connected diagrams:

1. Shapes in diagrams are typically drawn in a way that the bounding boxes of any two shapes have little overlap.

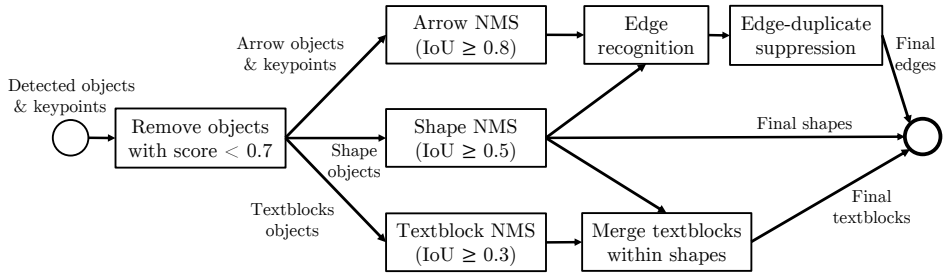


Figure 5.4: Structure recognition pipeline

2. There is at most one textblock within a shape that labels this shape.
3. Most graphical languages, including flowcharts and finite automata, allow at most one edge per direction between two shapes.
4. The bounding boxes of arrows can have a large overlap, especially for opposite arrows that join the same shapes.

Based on these observations, Figure 5.4 shows our structure recognition method. Given the detected objects and arrow keypoints, we employ a pipeline to obtain the final set of shapes, edges, and textblocks. We opt for a rule-based sequential method that starts by filtering object candidates based on a classification score threshold. Next, we use three dedicated post-processing procedures that take into account the specifics of shapes, arrows, and textblocks.

Shape Post-processing. For shapes, we propose a *Shape NMS* procedure, which is based on the observation that shape bounding boxes typically have little overlap. More concretely, we want to ensure that our model does not generate two detections for different shape classes with almost identical bounding boxes. While allowing multiple detections for the same shape would increase recall, it is unrealistic to assume such a scenario in practice, and it merely shows that the model is not sure about which class to assign. To prevent those duplicate detections, we perform NMS over all shape classes jointly.

Arrow Post-processing. Regarding arrows, we observe that arrows can have a large bounding box overlap. Figure 5.5 shows an exemplary diagram with two arrows that have close to 70% IoU. Even with a perfect model, the NMS post-processing in standard Faster R-CNN with a 50% IoU threshold would eliminate one of both arrows (the one with the lower score). Therefore, we use a dedicated *Arrow NMS* procedure, where we increase the IoU threshold for arrows to 80%. The *Edge recognition* step computes the distance of each arrow keypoint to its closest shape and creates a candidate edge between the two respective closest

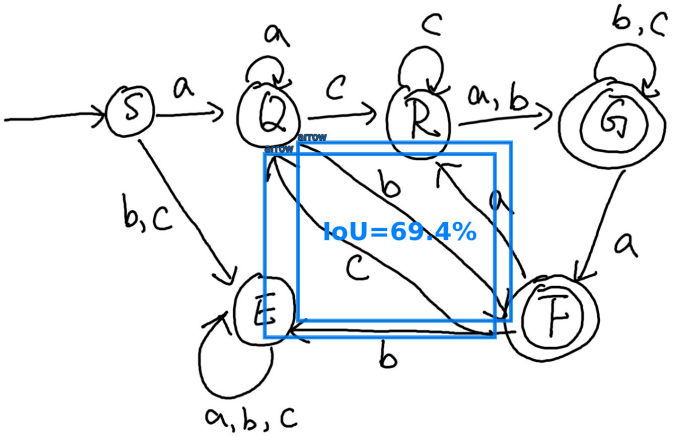
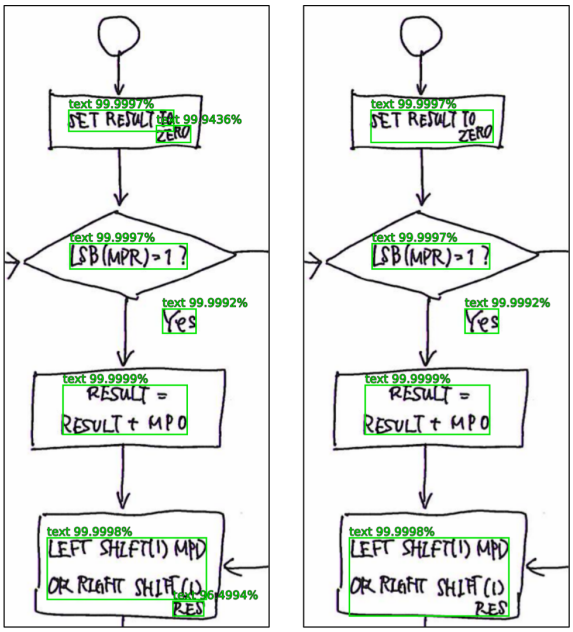


Figure 5.5: *Arrow bounding box overlap*: The two highlighted arrows have a large IoU. Faster R-CNN with standard NMS will always suppress the one with the lower score.

shapes. For the FA dataset, where initial arrows have no predecessor shape, we use a heuristic and only connect an arrow to a predecessor shape if the spatial distance between the arrow tail keypoint and the shape bounding box is lower than 50px. Increasing the arrow NMS IoU threshold generates a lot more arrow candidate detections, and often multiple candidates per ground truth arrow. To filter those duplicates, we employ an *Edge-duplicate suppression* step, which eliminates duplicate candidate edges that connect the same two shapes in the same direction. Duplicates are resolved by choosing the edge with the highest classification score.

Textblock post-processing. For textblocks, we propose a *Textblocks NMS* step with a reduced IoU threshold of 30%. This is based on the finding that the axis-aligned bounding boxes typically enclose the textblocks very well, and it is uncommon to have textblocks that largely overlap. Last, we *Merge textblocks within shapes*. Specifically, in case the model detects multiple textblocks within a shape bounding box, we merge those into a unified textblock during post-processing. The unified textblock is created with a union bounding box and maximum classification score over all textblocks in question. Fig. 5.6 illustrates this merging procedure with a concrete flowchart example.

The structure recognition method concludes our Arrow R-CNN inference pipeline. As illustrated in Figure 5.1, we use the final edges, shapes, and textblocks to create the recognized diagram as the output of our method. The next section details how we train the network components of Arrow R-CNN.



(a) Before merging texts (b) After merging texts

Figure 5.6: *Merge textblocks within shapes* example

5.1.4 Training

We follow the multi-task learning approach of Faster R-CNN and train all `Arrow R-CNN` networks jointly. Figure 5.7 gives an overview of how we compute a loss for a given image. First, we apply an *image augmentation* pipeline to obtain a randomly augmented version of a given diagram image. Given the augmented image, we use Faster R-CNN to generate a large set of object proposals. In *arrow proposal generation*, we identify the object proposals that sufficiently overlap with a ground-truth arrow bounding box. Finally, we use the arrow proposals to compute the *arrow loss* $\mathcal{L}_a$, and obtain an image loss $\mathcal{L}$ by combining this loss term with the Faster R-CNN loss $\mathcal{L}_f$. In the following, we introduce each step in more detail.

Image augmentation. We use the following image augmentation pipeline to improve the generalization capabilities of our model:

1. *LongestMaxSize*: Resize image to a longer side of 1333px while preserving aspect ratio ($p = 1.0$)
2. *IAMWordAugmentation*: Augment diagram with up to three random word images of size (w, h) from the words in the IAM-database [61], where $5 \leq w \leq 300$ and $12 \leq h \leq 150$ ($p = 1.0$)

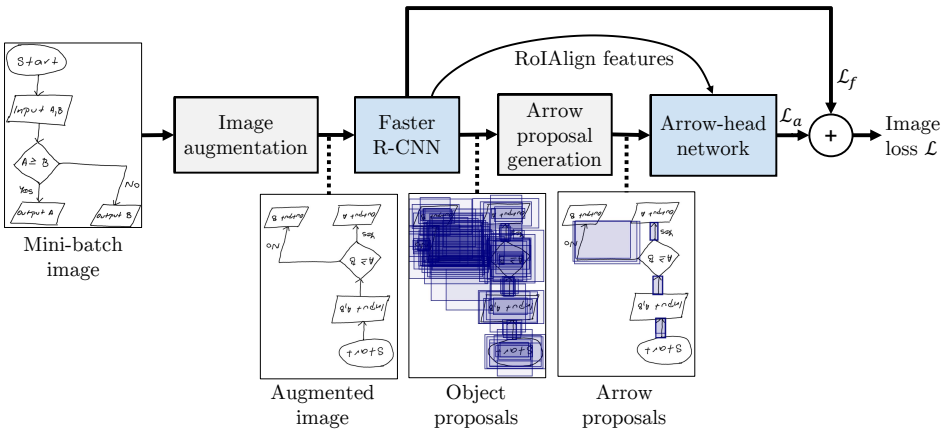


Figure 5.7: `Arrow R-CNN` training overview

3. *ShiftScaleRotate*: Use uniformly sampled ranges for shifting image by a factor $[-0.01, 0.01]$, scaling image by factor $[-0.2, 0.0]$, and rotating image $[-5°, 5°]$ ($p = 0.3$)
4. *RandomRotate90*: Rotate image by 90 degrees zero or more times ($p = 0.3$)
5. *HorizontalFlip*: Flip image horizontally ($p = 0.3$)
6. *VerticalFlip*: Flip image vertically ($p = 0.3$)

This pipeline is applied as a sequence, and each step is applied with probability p. We use the Albumentations library [12] for all augmentations except *IAMWordAugmentation*. For *IAMWordAugmentation*, we augment the diagram with words from an external dataset, as outlined in the following.

Due to the limited size of most datasets and the varying size and forms of arrow and text objects, we noticed that the model frequently confuses arrows with textblocks and vice versa. As an example, we noticed several cases where the detector falsely predicted an arrow within a textblock, e.g. a handwritten "l" within the term "false". To increase the robustness of arrow and text detection, we augment the training diagrams with handwritten words from the IAM-database [61]. The corpus consists of 1,066 forms written in English and produced by about 400 different writers, resulting in more than 80k word instances out of a vocabulary of roughly 11k words. Out of these word instances, we randomly sample words with a minimum word image height to exclude words that consist solely of punctuation marks and restrict the word image width to exclude overly long words. Unlike the online hand-drawn diagrams, the forms have been scanned and contain document noise. To assure our detector does not learn to classify those textblocks solely

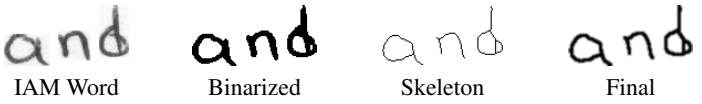


Figure 5.8: IAM Word Preprocessing Example

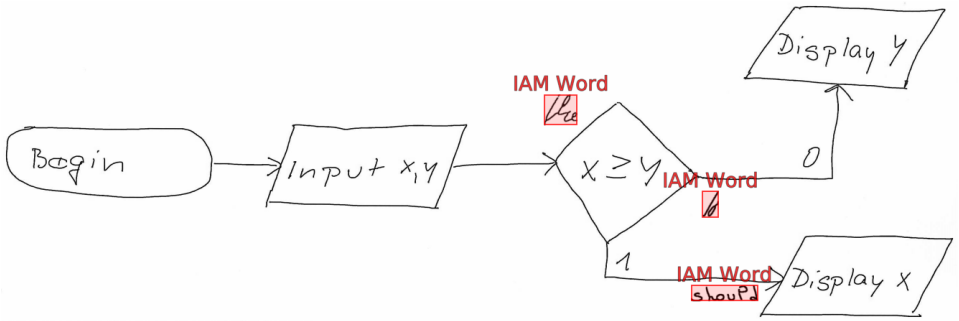


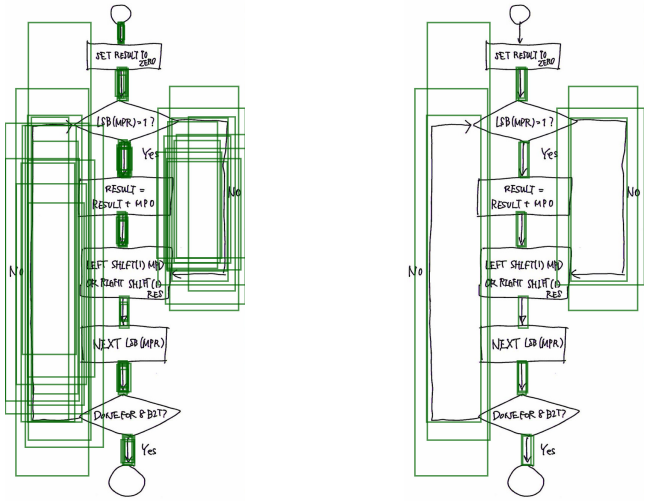
Figure 5.9: Exemplary $\texttt{FC\_B}_{\texttt{scan}}$ flowchart augmented with three IAM words

due to their document noise, we preprocess the IAM words to increase the visual similarity to the diagram textblocks. To derive a stroke-based representation, we binarize the image using Otsu's method and skeletonize it to a one pixel wide representation. Afterward, we use a procedure similar to the diagram rendering process described in Section 5.2 to create words with uniform 3 pixel wide smoothed strokes. Figure 5.8 shows an exemplary input IAM word and its preprocessed word image, along with intermediate word image representations.

During training, we augment each diagram by inserting up to three random IAM words into background regions. In the diagram datasets, textblocks are located quite close to the symbol or arrow that they annotate. To imitate this closeness, we place each IAM word close to an existing symbol while ensuring that the pixels of both objects do not overlap. Concretely, we ensure that the distance to the closest flowchart pixel is in the range $[5, 50]$. Figure 5.9 shows an exemplary flowchart augmented with IAM words.

The augmented image produced in this step is used as input for the next step in the training procedure (cf. Figure 5.7), in which we use Faster R-CNN to generate a large number of object proposals. In the following, we describe how we identify suitable object proposals to train our arrow keypoint detector.

Arrow proposal generation. In Faster R-CNN, the first RPN stage generates a set of proposals and then performs non-maximum suppression (NMS). For any two proposals that have an IoU of at least 70%, NMS iteratively removes the proposal with the lower objectness score. In order to train the RoI box network, Faster R-



(a) (a) + (b)

Figure 5.10: *Arrow proposal example*: the left image shows the 72 proposals with at least 70% IoU to a ground truth arrow, the right image shows the 19 proposals that additionally have both arrow keypoints located within their proposal box.

CNN considers the top 2000 proposals ranked by their objectness score (cf. Chapter 2). For training our arrow network, we use a subset of these 2000 proposals. Concretely, we define an arrow proposal as a proposal that fulfills two criteria:

(a) The proposal bounding box has an IoU of at least 70% with the bounding box of a ground-truth arrow.

(b) Both arrow keypoints of the matched ground-truth arrow are located within the proposal bounding box.

Figure 5.10 shows exemplary arrow proposals that fulfill either criterion (a), or both (a) and (b). We use the identified arrow proposals to compute the arrow loss term, as we outline next.

Arrow Loss. Given the bounding box of each arrow proposal $\mathbf{b}$, the arrow-head network (cf. Figure 5.3) predicts two keypoints $\bar{\mathbf{k}}^{\text{head}}$ and $\bar{\mathbf{k}}^{\text{tail}}$, which are encoded against the proposal bounding box. In the following, we discuss how we combine the predicted arrow keypoints into an overall arrow loss $\mathcal{L}_a$. Given our set of N arrow proposal pairs with the regression targets $\mathbf{t}_i = \left(\bar{\mathbf{k}}_i^{\text{head}\top}, \bar{\mathbf{k}}_i^{\text{tail}\top}\right)^\top \in T$ with the $\bar{\mathbf{k}}_i$ as defined in equation (5.1) for arrows $i = 1 \dots N$ and corresponding predictions $\hat{\mathbf{t}}_i \in \hat{T}$. The arrow loss is computed as the mean squared error over all

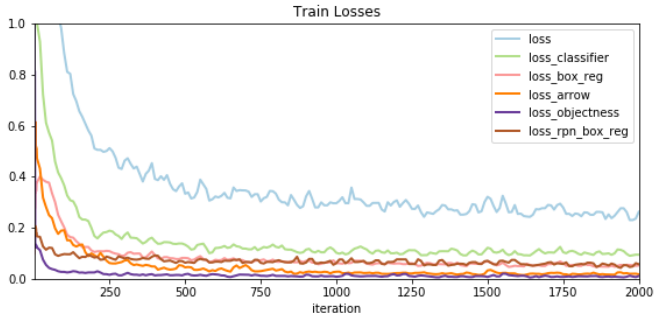


Figure 5.11: FC_B$_{\texttt{scan}}$ training losses during the first 2000 iterations ($\lambda = 1$)

predictions and dimensions:

$$\mathcal{L}_a(T, \hat{T}) = \frac{1}{4N} \sum_{i=1}^{N} \sum_{d=1}^{4} \left(\mathbf{t}_i[d] - \hat{\mathbf{t}}_i[d]\right)^2 \tag{5.2}$$

Finally, we obtain the image loss $\mathcal{L} = \mathcal{L}_f + \lambda \mathcal{L}_a$, where $\mathcal{L}_f$ is the combined Faster R-CNN loss. The hyperparameter λ balances the arrow and the other task losses. We found that $\lambda = 1$ sufficiently balances the loss terms and thus did not treat λ as another hyperparameter to tune. Figure 5.11 shows the individual loss terms throughout the first 2000 iterations on the FC_B$_{\texttt{scan}}$ database.

5.2 Datasets

We evaluate our method on four hand-drawn diagram datasets, three depicting flowcharts (FC_A [3], FC_B [11], and DIDI [27]), and one finite automata dataset (FA [6]). In the remainder of this section, we discuss how each dataset has been split into training, validation, and test set, how we render the online strokes as images, and how we obtain the ground-truth bounding boxes and arrow keypoints of each diagram image.

5.2.1 Dataset splits

As mentioned in Chapter 3, the DIDI dataset consists of two parts, one that contains diagrams with textual labels (DIDI$_{\texttt{text}}$), and one that contains diagrams without labels (DIDI$_{\texttt{no_text}}$). Throughout the experiments, we train on the entire DIDI dataset, but report the results for both parts separately. All datasets were split either by writers (FC_B, FA, DIDI) or by templates (FC_A), such that the sets

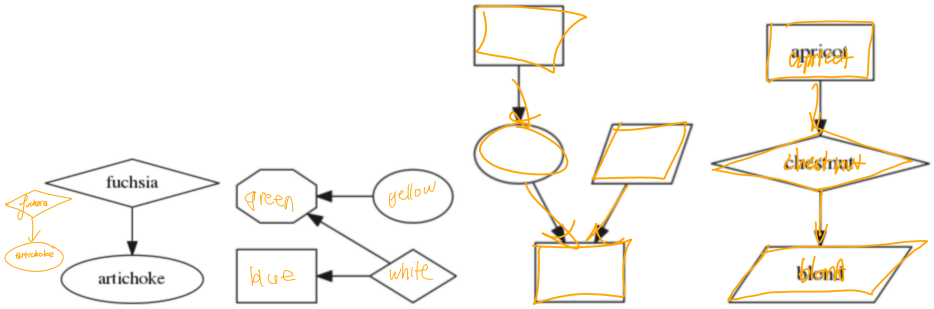


Figure 5.12: Exemplary DIDI diagrams with overlaid drawings

of writers (or respectively templates) in the training, validation, and test splits are disjoint. This means that the experimental results either show to what extent the model generalizes to unseen writers or unseen layouts, but not both at the same time. As another difference, FC_A has no dedicated validation set. To avoid overfitting to the test set, we conduct all hyperparameter tuning on the FC_B training and validation set and train a model on FC_A using the same configuration.

5.2.2 Stroke rendering

All four diagram datasets are online datasets, where each diagram has been captured as a sequence of strokes. In order to generate an image from the stroke sequence, we render the strokes as anti-aliased polylines with a stroke width of 3. The only exception is the FC_B dataset, where we use the offline $\texttt{FC\_B}_{\texttt{scan}}$ dataset introduced in [9], which contains scans of printed FC_B diagrams. For the FC_A and FA datasets, after rendering the strokes, we pad each image by 10 pixels on each side to ensure that border-touching strokes are fully visible. This padding strategy is not required for the DIDI dataset, as it contains the dimensions of the drawing area that was shown to the user. Here, we create a white image with the dimension of the drawing area and then plot the strokes at their corresponding positions. During data collection of the DIDI dataset, the generated flowcharts were rescaled to fill the drawing area. The size of this drawing area varies, with a maximum of 3600×2232 pixels. To avoid overly large images, we rescale each image to the initial scale of the flowchart.

Table 5.1: Excluded DIDI diagrams with drawing errors

	$\texttt{DIDI}_{\texttt{no\_text}}$		$\texttt{DIDI}_{\texttt{text}}$	
Split	**Count**	**Percentage**	**Count**	**Percentage**
Train	404	1.48	1,303	7.79
Validation	12	0.26	50	1.80
Test	12	0.26	867	31.13
Total	428	1.18	2,220	9.96

5.2.3 Bounding boxes

In the following, we outline how we generate the ground-truth bounding boxes for the datasets. For the offline $\texttt{FC\_B}_{\texttt{scan}}$ dataset, we use the bounding box annotations provided with the dataset. For FC_A and FA, we define the bounding box of each symbol as the union bounding box of all its strokes. As mentioned in Chapter 3, the DIDI dataset is not annotated on stroke level. Therefore, we instead use the symbol bounding boxes of the corresponding GraphViz diagram. Since the participants did not draw precisely over the diagrams, the extracted bounding boxes do not perfectly fit the hand-drawn symbols. To quantify this difference, we manually annotate 100 hand-drawn diagrams (50 without and 50 with textual labels), and compare the extracted GraphViz and the annotated hand-drawn bounding boxes.

Figure 5.12 shows some drawings from this sample and illustrates two major drawing issues that we identified: diagrams where a user did not draw over the flowchart as instructed (5.12a), and diagrams where a user forgot to draw some or all of the shapes (5.12b). Since the evaluation metrics described in Section 5.3.1 are based on bounding box IoU, we try to exclude these erroneous diagrams in order to get a meaningful assessment of our method. As a heuristic, we exclude a drawing if at least one bounding box contains no stroke pixels. This heuristic correctly identifies 8 diagrams with drawing errors out of the sample of 100 diagrams, but it misses one diagram where the user forgot to draw an arrow. Table 5.1 shows the proportion of excluded diagrams using mentioned heuristic, and it reveals that drawing mistakes occur very frequently in the $\texttt{DIDI}_{\texttt{text}}$ train and test set.

Besides excluding drawings, we also account for inaccurate drawings such as Figure 5.12c and 5.12d in the evaluation procedure. Specifically, we use an IoU threshold of 50% instead of 80% for the evaluation metrics described in Section 5.3.1. Within the sample of 92 diagrams without drawing mistakes, 0 out of 325 annotated shapes, 19 out of 271 arrows (i.e., 7%), and 40 out of 188 (i.e., 21%) textblocks have less than 50% IoU between handwritten and GraphViz bounding box. Two of those textblocks and two arrows can be found in Figure 5.12c

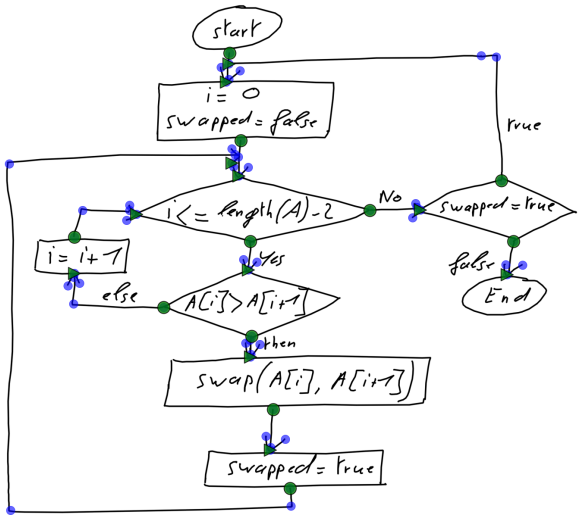


Figure 5.13: *FC_A ground-truth arrow keypoint heuristic*: The detected corner peaks (blue) are used to compute the head (green triangle) and tail (green circle) arrow keypoints.

and 5.12d. Overall, this means that a bounding box that perfectly encloses a hand-drawn symbol can still be evaluated as incorrect, even with the relaxed IoU threshold. Instead, for a positive evaluation result the model is required to predict the location and size of the corresponding GraphViz symbol.

5.2.4 Arrow keypoints

For training our arrow keypoint detector, we need to specify the ground-truth arrow head and tail points. The head and tail points are explicitly annotated in the FA and $\text{FC\_B}_{\text{scan}}$ dataset. For the DIDI dataset, we extract the head and tail keypoints from the arrow control points in the GraphViz dot file of the generated flowchart. For the FC_A dataset, we use a heuristic to extract the keypoints from the stroke data. For each arrow, we compute the Harris corner measure response image, and then identify corner peaks with a minimum distance of 5. We use the *scikit-image* [97] library implementations of the respective algorithms. We set the arrow head and tail keypoints as the corner points closest to the source and target shape, respectively. Figure 5.13 illustrates this heuristic using a flowchart from FC_A, including the detected corner peaks and extracted keypoints.

We also quantitatively evaluate the accuracy of our heuristic on the FC_B dataset, where the head and tail points have been annotated. For the flowcharts in the training split, we compute the mean absolute error (mae) based on the euclidean

distance between each approximated and annotated arrow keypoint. We find that the approximated arrow tail (mae=1.38) and arrow head (mae=5.82) keypoints are sufficiently close to the human annotations.

5.3 Evaluation

In this section, we describe the evaluation setup (Section 5.3.1) before we present the experimental results (Section 5.3.2). We complete the experiments with an error analysis (Section 5.3.3), where we also outline how future work could address common sources of error.

5.3.1 Evaluation Setup

Below we elaborate on the implementation details and the metrics that we use to evaluate our approach.

Implementation details. Our `Arrow R-CNN` implementation is based on the *maskrcnn-benchmark* [62] R-CNN framework and uses PyTorch [69]. As CNN backbone we use ResNet-101 with FPN. For training, we adopt the recommended framework parameters for our CNN backbone. We use SGD with a weight decay of 0.0001 and momentum of 0.9. Each model is trained on a Tesla V100 GPU with 16GB memory for 90k mini-batches, while reducing the learning rate after 60k and 80k iterations by a factor of 10. On the Tesla V100 GPU the training takes between 25 and 30 hours. We use a batch size of 4, and decrease the learning rate from the recommended value of 0.02 for a batch size of 16 to 0.005 according to the linear scaling rule [30]. To decrease memory usage during training, we use the default framework configuration and group images with similar aspect ratios in one batch.

To demonstrate the general applicability of our approach, we use identical configurations to train and evaluate models for all datasets, except for two exceptions: as discussed in Section 5.1.3, we use an arrow distance threshold to account for arrows without a predecessor shape in the `FA` dataset. For the `DIDI` dataset, we do not use augmentation methods since the dataset is very large.

Metrics. We evaluate our method using recognition metrics on symbol and diagram level. Regarding *symbol recognition*, Bresler, Průša, and Hlaváč [9] compute the symbol recognition recall at an IoU threshold of 80%. Additionally, arrows are required to be connected to the correct shapes. When using an object detector framework, the recall negatively correlates with the utilized detection score threshold. Without using NMS and a detection score threshold, a Faster R-CNN system generates one detection per object proposal, which would result in more than 1000

Table 5.2: *Diagram recognition rate*: Comparison of our method with other online and offline methods. For FC_B, the offline results are based on $\texttt{FC\_B}_{\texttt{scan}}$.

	FC_A	FC_B	FA	$\texttt{DIDI}_{\texttt{no\_text}}$	$\texttt{DIDI}_{\texttt{text}}$
Online methods					
Wang et al. [102]	5.8	—	—	—	—
Julca-Aguilar et al. [42]	34.0	—	—	—	—
Bresler et al. [11]	59.1	67.9	79.8	—	—
Offline methods					
Bresler et al. [9]	—	37.7	—	—	—
Arrow R-CNN	**68.4**	**78.6**	**83.3**	**83.9**	**85.1**

detections per image. Therefore, recall on its own does not possess much informative value for evaluating object detectors, since any detector can be configured to achieve very high recall. This is an important distinction between related work that uses algorithms based on reconstructed strokes. Here, this trade-off is less severe, since each reconstructed stroke is assigned to at most one symbol. However, false-positive reconstructed strokes, such as noise patterns that stem from the scanning process, might still lead to false-positive symbols, which affect the precision of the system. In [9] symbol recognition precision is not reported. To make the symbol recognition recall comparison somewhat fair, we use a score threshold of 0.7 throughout all our experiments, which corresponds to the default threshold of our object detector framework. Moreover, with our *Shape NMS* post-processing, we also ensure that we do not have multiple predictions for one symbol.

On a more aggregate level, the *diagram recognition* metric intuitively assesses the performance of a diagram recognition system as the ratio of correctly recognized diagrams in a test set. In this setting, a diagram has been recognized if the number of detected symbols equals the number of ground truth symbols, each symbol has been correctly classified and localized with at least 80% IoU, and each arrow predecessor and successor shape has been correctly identified.

5.3.2 Results

Diagram recognition. Table 5.2 shows that Arrow R-CNN achieves state-of-the-art performance in offline recognition. Even though our method uses no stroke information, the diagram recognition rates are also higher than online systems.

Further, we conduct two ablation studies to quantify the effect of each augmentation and post-processing method. Table 5.3 shows that the augmentation methods substantially improve the diagram recognition rate for small datasets, es-

Table 5.3: *Augmentation ablation study*: augmentation increases the diagram recognition rate for the small datasets substantially, but lowers the rate on the large DIDI dataset. All results use standard Faster R-CNN post-processing and are based on the test set.

	FC_A	$\text{FC\_B}_{\text{scan}}$	FA	$\text{DIDI}_{\text{no\_text}}$	$\text{DIDI}_{\text{text}}$
No Augmentation	23.4	70.4	52.4	82.5	**83.7**
ShiftScaleRotate	33.9	73.0	60.7	**82.6**	83.5
+ Rotate90 & Flip	57.3	**77.0**	79.8	80.9	**83.7**
+ IAMWordAug.	**66.7**	76.0	**81.0**	80.8	83.5

Table 5.4: *Structure recognition ablation study*: the post-processing methods increase the diagram recognition rate on all test sets.

	FC_A	$\text{FC\_B}_{\text{scan}}$	FA	$\text{DIDI}_{\text{no\_text}}$	$\text{DIDI}_{\text{text}}$
Standard NMS (IoU ≤ 0.5)	66.7	76.0	81.0	82.5	83.7
+ Shape post-processing	66.7	**78.6**	82.1	83.5	83.8
+ Arrow post-processing	67.8	**78.6**	**83.3**	**83.9**	**85.1**
+ Textblock post-processing	**68.4**	**78.6**	**83.3**	**83.9**	**85.1**

pecially for FC_A, where the model has to generalize to unseen layouts. For the large DIDI dataset, the augmentation methods slightly lower the recognition rate. To demonstrate the general applicability of our method, we used the same number of training iterations for all datasets. However, the combination of a large dataset size and multiple augmentation methods might require more training iterations. We leave the investigation of the interplay between augmentation methods and dataset size to future work.

Table 5.4 shows the results of the post-processing ablation study and reveals that Shape NMS improves the diagram recognition rate on four out of five datasets. Increasing the Arrow NMS IoU threshold and introducing edge suppression leads to further improvements on all datasets except $\text{FC\_B}_{\text{scan}}$, where the rate stays the same. Also, merging textblocks within a shape is a straightforward method to improve the results on FC_A.

Symbol recognition. Table 5.5 – 5.8 show the symbol recognition results for the evaluated datasets. Overall, Arrow R-CNN achieves perfect recognition results for several shapes, which can be explained by the fact that the appearance and the scale of shapes have a much lower variance than for arrows and textblocks.

On the FC_A dataset (Table 5.5), where related works report 83.2% [103] and 84.2% [9] symbol recognition recall, Arrow R-CNN has a much higher recall

Table 5.5: `FC_A` symbol recognition at IoU 80% on test set

Class	**Arrow R-CNN**		**Wu et al.** [103]
	Precision	**Recall**	**Recall$_{\text{IoU50}}$**
Arrow	94.7/97.3*	96.0/98.5*	80.3
Connection	99.2	100	73.4
Data	100	99.7	78.5
Decision	100	99.5	78.9
Process	99.8	100	88.3
Terminator	100	100	90.6
Text	99.3	99.1	86.0
Micro avg.	97.9/98.8*	98.3/99.1*	83.2

* does not consider if arrows have been correctly matched to shapes

Table 5.6: `FC_B`$_{\texttt{scan}}$ symbol recognition at 80% IoU on test set

Class	**Arrow R-CNN**		**Bresler et al.** [9]
	Precision	**Recall**	**Recall**
Arrow	98.0/98.0*	98.0/98.0*	84.3
Connection	100	100	86.6
Data	100	94.9	94.4
Decision	100	100	96.9
Process	95.5	100	98.8
Terminator	100	100	93.6
Text	99.2	99.3	93.7
Micro avg.	98.7/98.7*	98.7/98.7*	91.3

* does not consider if arrows have been correctly matched to shapes

Table 5.7: `FA` symbol recognition at 80% IoU on test set

Class	**Precision**	**Recall**
Arrow	98.4/99.0*	98.4/99.0*
Final state	100	100
State	100	100
Text	99.6	99.7
Micro avg.	99.3/99.5*	99.3/99.5*

* does not consider if arrows have been correctly matched to shapes

Table 5.8: DIDI symbol recognition at 50% IoU on test set

Class	$\texttt{DIDI}_{\texttt{no\_text}}$		$\texttt{DIDI}_{\texttt{text}}$	
	Precision	**Recall**	**Precision**	**Recall**
Arrow	95.6/97.5*	94.7/96.5*	96.6/99.2*	95.2/98.0*
Box	97.1	96.5	99.9	99.8
Diamond	99.2	97.5	99.9	99.9
Octagon	96.3	92.2	100	99.7
Oval	92.6	97.2	99.7	99.4
Parallelogram	97.9	97.0	99.9	99.8
Text	—	—	98.5	97.7
Micro avg.	96.1/97.0*	95.4/96.3*	98.4/99.1*	97.6/98.4*

* does not consider if arrows have been correctly matched to shapes

(98.3%). The largest source of error of our method is in the arrow class, where arrows have either not been detected with at least 80% bounding box overlap, or the arrow has not been joined to the correct shapes. On the FC_A training set, we noticed that our model fails to recognize diagrams of templates 5 and 7. In these layouts, shapes are sometimes connected through a sequence of two arrows. Yet, our current post-processing logic assumes that an arrow always points to a shape, and thus connects the arrow to the shape closest to its head keypoint. We leave the development of appropriate methods for the arrow-after-arrow scenario to future work.

Table 5.6 shows that Arrow R-CNN can accurately recognize symbols in scanned flowcharts. In the $\texttt{FC\_B}_{\texttt{scan}}$ test set, all arrows that have been detected correctly are also connected to their ground truth shapes. This demonstrates the effectiveness of the Arrow R-CNN arrow keypoint mechanism and post-processing. Arrow R-CNN is also applicable to diagrams other than flowcharts. As Table 5.7 illustrates, the model perfectly recognizes the state and final state shapes in the FA finite automata test set, and also achieves very good results for arrows and text. For the DIDI dataset, the symbol recognition results in Table 5.8 are all above 90%, but slightly lower than the results on the other dataset, even though a lower IoU threshold of 50% is used. As discussed in Section 5.2, this has to do with the discrepancy between the ground truth bounding boxes extracted from the diagram and the actual drawings. In Section 5.3.3 we further discuss the implications of this discrepancy and show some error cases.

Finally, Table 5.9 shows that our system recognizes a diagram in less than 100ms on average, and is two orders of magnitude faster than related work.

Table 5.9: FC_B$_{scan}$ runtime in milliseconds per image: Arrow R-CNN timings are taken using a Tesla V100 GPU, with the image already resized and loaded to memory.

Method	Min	Mean	Std	Max
Arrow R-CNN	59	91	15	119
Bresler et al. [9]	2,623	10,970	—	37,972

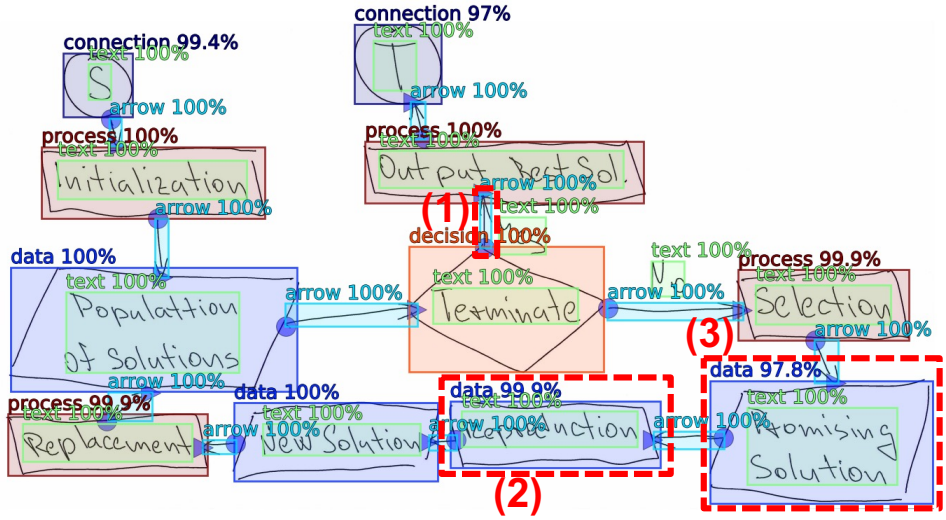


Figure 5.14: FC_B$_{scan}$ flowchart with most missing symbols in validation set: highlighted in red are (1) an arrow with only 74% IoU, (2) a process symbol confused as data, and (3) a data symbol confused as a process.

5.3.3 Error Analysis and Future Work

Figure 5.14 shows the predicted symbols and arrow keypoints of a flowchart from the FC_B$_{scan}$ dataset. For textblocks and straight arrows, where one bounding box side is often very short, a prediction off by a few pixels can result in less than 80% IoU. This raises the question of whether an 80% IoU threshold for all symbol types is too strict. From an end-user perspective, it might only matter that the arrow has been correctly identified as an edge between two shapes. To this end, future research could investigate graph similarity measures to evaluate diagram recognition systems. The two confusions between process and data are likely due to the fact that the writer in question draws very uneven lines. These uneven lines are typically caused by uncontrolled oscillations of the hand muscles, dampened by inertia [88]. In handwriting recognition, elastic distortion augmentation is a way

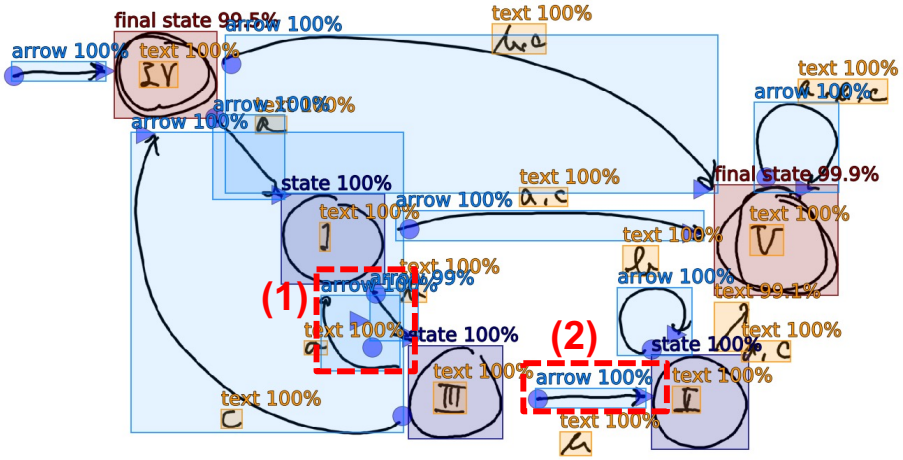
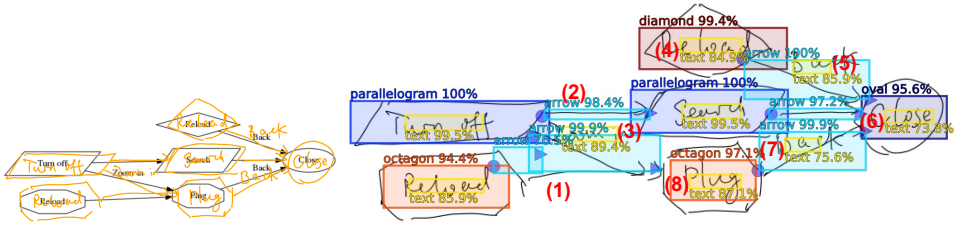


Figure 5.15: *FA diagram with arrow-match errors*: The highlighted arrows have not been correctly matched to their source shape since the distance between predicted tail keypoint and shape bounding box exceeds the threshold.

to simulate these oscillations [46, 88]. We found that although elastic distortion augmentation improves classification results, it has a negative effect on localization accuracy. This is due to the fact that the distortions cause the annotated bounding box and keypoints to be inaccurate, e.g. by distorting a line close to a bounding box such that it surpasses the bounding box. Future work could investigate elastic distortion methods that also adapt ground truth annotations accordingly.

Figure 5.15 shows a finite automata diagram. The diagram has two arrows with overlapping bounding boxes, and the model does not accurately predict the keypoints for the larger arrow. This suggests that the model is not sure which arrow head it should attend to. The example shows that localizing arrows through axis-aligned bounding boxes has its limitations when their bounding boxes overlap. A ground-truth arrow bounding box can contain multiple arrow heads, which forces the model to not just recognize arrow heads in a local context. Instead, the model is required to consider a wider context to identify the relevant arrow. Future research could investigate more robust methods to detect arrows and their keypoints. In scene text detection, it is common to predict rotated instead of axis-aligned bounding boxes [119]. These rotated bounding boxes would capture arrows in a more compact way and lead to fewer overlapping bounding boxes. As another approach, edge recognition could be framed as visual relationship detection [45]. Instead of detecting arrow instances, a classifier could directly predict if two given shapes are connected through an arrow. Alternatively, a classifier could predict which arrow head and tail keypoint belong together.



(a) GraphViz diagram and drawing **(b)** Drawing and predictions

Figure 5.16: DIDI$_{\texttt{text}}$ diagram example: 5.16a shows how severely the drawn shapes and texts differ from their corresponding GraphViz symbols. The errors highlighted in red in 5.16b are (1) a missing arrow due to confusion between two crossing arrows, (2) an arrow with only 0.48 ground-truth IoU, and (3) – (8) texts with insufficient IoU.

For the DIDI dataset, Fig. 5.16 shows that the model is not only required to recognize the hand-drawn diagram but also to predict the GraphViz diagram it originates from. As illustrated, the model correctly predicts shape bounding boxes which are smaller than the hand-drawn shapes, i.e., it recognizes that the shapes have been drawn excessively large. However, when the position and size of the hand-drawn and GraphViz symbols differ too much, this task becomes nearly impossible. This is indicated by the numerous text localization errors in the example, where e.g., the handwritten "Back" arrow labels have a very small intersection with the corresponding GraphViz labels. Future work could propose evaluation methods that better disentangle handwriting and GraphViz diagram recognition performance for DIDI.

5.4 Conclusion

We propose Arrow R-CNN, the first deep learning system for offline hand-drawn diagram recognition. Our system correctly recognizes more than 68% of all diagrams in four diagram datasets and also improves the symbol recognition state of the art in all symbol classes. We show that we can train highly accurate deep R-CNN models on small datasets when using data augmentation methods tailored to hand-drawn diagrams. Since standard Faster R-CNN post-processing is not well suited for diagram symbol recognition, we propose a post-processing method that takes into the specifics of shapes, edges, and textblocks. On average, Arrow R-CNN recognizes arrow-connected diagrams in less than 100 ms, which allows it to be used in environments that require quick response times.

Naturally, this work is subject to several limitations. First, Arrow R-CNN does not integrate with a handwriting recognition system for recognizing the tex-

tual content of the detected textblocks. We address this limitation in Chapter 8.

Second, `Arrow R-CNN` uses a heuristic for edge recognition, where the source and target shape are defined as the shapes that are closest to the tail and head keypoint, respectively. In Section 5.3.3, we observe that this heuristic sometimes connects an arrow to an incorrect shape, especially when the respective predicted arrow keypoint is inaccurate. One way to prevent such invalid edges is to leverage the syntactical rules of the modeling language. We investigate such an approach in the next chapter. In Section 5.3.3, we also hypothesize that edge recognition could be framed as a visual relationship detection task, where the model directly predicts the source and target shape that each arrow connects. We follow up on this in Chapter 7 and Chapter 8, where we propose two such approaches.

Third, we observe that `Arrow R-CNN` has a symbol recognition recall of at least 98% on all datasets except `DIDI`, which indicates the need for more challenging datasets. For the `DIDI` dataset, we find that the ground-truth bounding boxes in the dataset significantly deviate from the corresponding drawn symbols and that the drawings often contain misplaced or missing symbols. We conclude that `DIDI` is only partially suited for developing and evaluating diagram recognition methods, as neither approaches (e.g., object detectors) nor evaluation metrics (e.g., IoU-based metrics) based on bounding boxes can be effectively applied. Given the limitations of the existing datasets, we proposed `hdBPMN` in Chapter 4.

In the next chapter, we use `Arrow R-CNN` to develop a first baseline for `hdBPMN` and propose several shape and edge recognition improvements specific to BPMN models.

Chapter 6

Sketch2BPMN

In the previous chapter, we presented Arrow R-CNN, which we have trained and evaluated on several online diagram datasets. As pointed out in Chapter 3, the diagrams in existing online datasets do not resemble the characteristics of offline diagrams encountered in practice, as they are drawn on a digital device and copied from a diagram template. To address these limitations, we have collected and annotated the `hdBPMN` dataset, which we presented in Chapter 4. While Chapter 4 details the recognition challenges associated with `hdBPMN`, it does not answer how state-of-the-art methods perform on this dataset. This chapter addresses this gap by developing a first `hdBPMN` baseline. To this end, we train Arrow R-CNN on a preliminary version of `hdBPMN` and evaluate its shape and edge recognition performance.[1] Further, we propose the `Sketch2BPMN` method, which improves Arrow R-CNN with an *edge post-processing* method that prevents invalid BPMN edges, and an *extended data augmentation* pipeline that simulates the varying properties of camera-based documents. `Sketch2BPMN` takes an image of a hand-drawn BPMN model as input and produces a respective BPMN XML file, which can be imported into process modeling tools.

The research presented in this chapter is based on a paper titled "*Sketch2BPMN: Automatic Recognition of Hand-Drawn BPMN Models*" by Bernhard Schäfer, Han van der Aa, Henrik Leopold, and Heiner Stuckenschmidt [85].

The remainder of this chapter is organized as follows. Section 6.1 describes the `Sketch2BPMN` approach. Section 6.2 describes the evaluation procedure and discusses the experimental results. Finally, Section 6.3 concludes the chapter.

[1] The preliminary version has 502 instead of 704 images and does not have annotated labels. The full dataset was published as part of `Sketch2Process` [86], which we cover in Chapter 8.

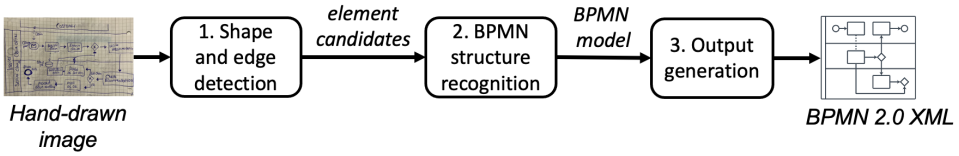


Figure 6.1: `Sketch2BPMN`: overview of the main steps

6.1 The `Sketch2BPMN` Method

This section introduces `Sketch2BPMN`, our method for recognizing a hand-drawn BPMN model from an image. As visualized in Figure 6.1, `Sketch2BPMN` consists of three main steps: 1) shape and edge detection, 2) BPMN structure recognition, and 3) output generation. Below we introduce each step in detail.

6.1.1 Shape and Edge Detection

This step of our approach generates sets of candidate BPMN shapes S_C and edges E_C for a provided hand-drawn image. Each candidate shape $s \in S_C$ is formalized as a tuple $s = (b, c, l)$, where b refers to the coordinates of a *bounding box*, i.e., a rectangle encompassing the predicted area of a drawn element, c the predicted BPMN element category, and l the likelihood that the shape s corresponds to category c. Furthermore, each candidate edge $e \in E_C$ is a tuple $e = (b, c, l, src, tgt)$, where b, c, and l are the respective shape counterparts, and src and tgt correspond to the *keypoint* coordinates of the edge, reflecting the points at which e is predicted to connect to shapes in S_C. Figure 6.2 visualizes such predicted candidates for the running example, showing, for example, that the upper bounding box is determined to have a 99.2% likelihood of being a *pool*.

To predict S_C and E_C, we build on *Arrow R-CNN* [82], an approach we established in earlier work for the recognition of hand-drawn diagrams, and adapt it to the specific characteristics of hand-drawn BPMN models. Particularly, we expand upon existing work through *improved keypoint detection* and *extended data augmentation*.

Arrow R-CNN. Arrow R-CNN detects objects in an image through the aforementioned bounding boxes, depicted in Figure 6.2. For each bounding box, Arrow R-CNN assigns a predicted class (from a set of predefined classes) and a likelihood between 0 and 1. Given its general applicability for the recognition of arrow-connected diagrams, we train Arrow R-CNN on the 21 different BPMN shape and three edge classes we consider in this work. However, we do this while also incorporating the following two key adaptations, which support the improved recognition of BPMN models in our scenario.

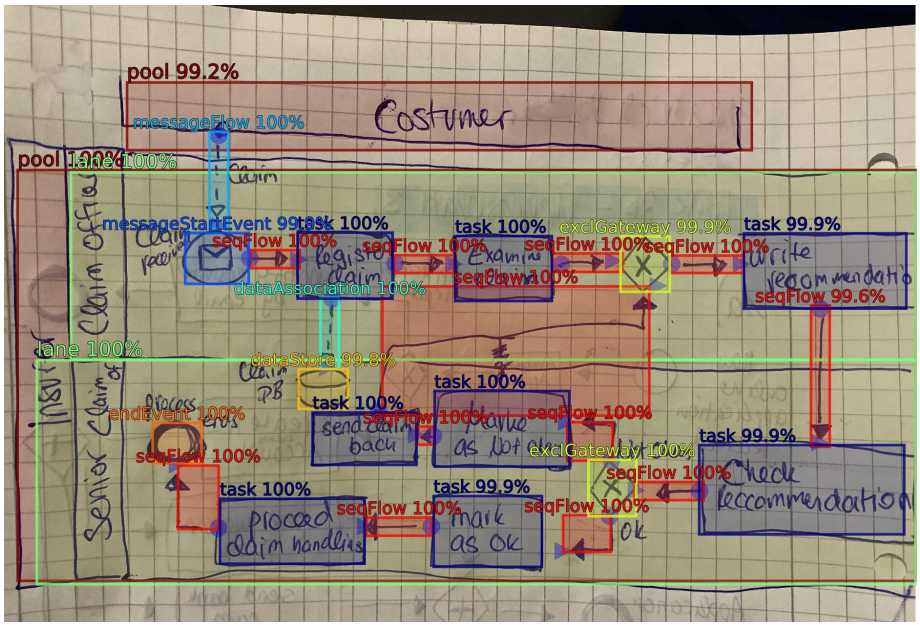


Figure 6.2: *Step 1*: Candidate shapes and edges have been classified and localized through bounding boxes, in addition, edges have predicted arrow head (▷) and tail (○) keypoints.

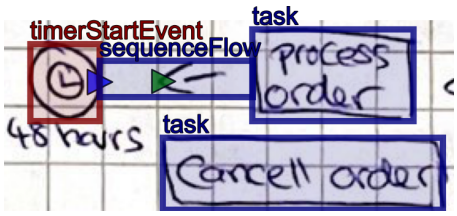


Figure 6.3: Improved keypoint detection due to annotating edge intersection (blue ▷) instead of drawn arrow head (green ▷).

Improved keypoint detection. A key challenge when dealing with hand-drawn BPMN models is the correct recognition of edges that are not properly connected to their respective source and target shapes. To account for this issue, we adapt the manner in which edge keypoints are encoded and predicted. For this, we first change the way edges are annotated in the training data. In particular, instead of using the drawn tail and head of the edge, we annotate the points where the edge intersects with its source and target, as illustrated in Figure 6.3. Then, the Arrow R-CNN model trained on these annotations will strive to predict where an edge *should* have ended (or started) if it had been drawn properly, rather than predicting the point where the edge ends (or starts) in the drawing. While this adapted method requires the keypoint predictor to perform reasoning beyond the recognition of a drawn arrowhead, this additional burden on the prediction model improves the accuracy of our approach. In par-

ticular, since this improved method considers the direction of a drawn edge when making predictions, it enables the approach to even properly recognize a shape to which an edge should connect, even when it is not the shape that is closest to the end of a drawn edge. For the example in Figure 6.3, our approach then correctly recognizes that the edge should connect to the *48 hours* event, rather than the *cancel order* activity, despite the latter shape being closer to the drawn arrowhead.

Extended data augmentation. Aside from BPMN specifics, we also have to account for a second particularity of the `hdBPMN` dataset, namely its diversity in terms of the means used to create and digitize the hand-drawn models, such as the type of paper and drawing implement (see also Section 4.3). Since Arrow R-CNN was designed to deal with much more uniform input (e.g., black drawing on white background), we need to adapt the training approach to the more difficult characteristics of our setting. To do this, we develop an image augmentation pipeline tailored to camera-based hand-drawn diagrams with varying backgrounds. Such augmentations have become a common regularization technique to combat overfitting in deep learning models for various image recognition scenarios [12]. They have been shown to be particularly valuable when training an approach on a dataset with only a few hundred images [82], such as in our case. Therefore, we add augmentation methods to simulate the varying properties of camera-based documents. Specifically, we randomly add gaussian noise, change the brightness and contrast of the image, and shift the hue, saturation, and value (HSV) color scale.

6.1.2 BPMN Structure Recognition

In the structure recognition step, `Sketch2BPMN` turns the sets of candidates S_C and E_C into filtered sets $S \subseteq S_C$ and $E \subseteq E_C$. In addition, each edge $e \in E$ is extended to a tuple $e' = (b, c, l, src, tgt, s_{src}, s_{tgt})$, where s_{src} specifies the source shape that e connects, and s_{tgt} the target shape. The resulting BPMN model obtained over S and E is connected and resembles the drawn model as closely as possible. We achieve this through shape disambiguation and edge post-processing.

Shape disambiguation. To turn a set of candidate shapes S_C into a set of predicted shapes S, we primarily need to disambiguate cases in which multiple candidates in S_C relate to the same drawn shape, i.e., duplicate detection and resolution. Although Faster R-CNN inherently resolves these issues for bounding boxes with the same predicted category, this is not the case when it comes to boxes with different categories. As a result, it may generate two candidate shapes, with different classes, for the same object in an image, as shown in Figure 6.4 for two categories of timer events.

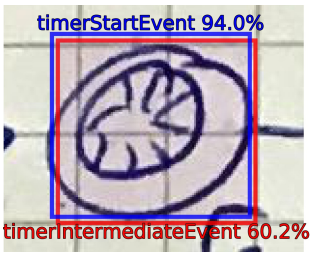


Figure 6.4: Duplicate shape candidates

However, determining that two candidates $s_1, s_2 \in S_C$ truly relate to the same drawn object is not trivial, especially in the context of BPMN models, whose hierarchical structure naturally leads to overlap between resource shapes (i.e., pools and lanes) and other shapes, such as activities and events. Therefore, we employ so-called *non-maximum suppression* (NMS) over all shape categories. NMS first determines if the bounding boxes of s_1 and s_2 have an overlap of at least 80%. We quantify this as the *Intersection over Union* (IoU), i.e., the ratio between the intersection area and the union area of $s_1.b$ and $s_2.b$. If this ratio exceeds 80%, NMS suppresses the shape with the lower classification score, i.e., it keeps the candidate that is predicted to be most suitable. In the case of Figure 6.4, the two candidates clearly have such significant overlap, which is why after employing NMS, we retain the blue shape corresponding to the more likely *timerStartEvent*, while omitting the other candidate from S.

Edge post-processing. To finalize the set of edges E to be included in a BPMN model we use a two-stage approach. First, we associate each edge candidate $e \in E_C$ with a source $s_{src} \in S$ and a target $s_{tgt} \in S$, which are the shapes that are closest to the edge's predicted keypoints, $e.src$ and $e.tgt$, and also correspond to a valid category with respect to the given edge. For instance, if $e.c = \textit{sequenceFlow}$, edge e will only be connected to shapes that are predicted to be *activities*, *events*, or *gateways*. To determine the closest shape, we compute the minimum Euclidean distance between a keypoint and all sides of a shape's bounding box. After identifying the closest, valid shapes, we turn a candidate edge $e \in E_C$ into a connected edge $e' = (b, t, l, src, tgt, s_{src}, s_{tgt})$.

Once each edge candidate is connected to source and target shapes, we omit all connected edges that correspond to highly unlikely or invalid constructs, such as *self-loops* ($e.s_{src} = e.s_{tgt}$) and invalid data associations (neither $e.s_{src}$ nor $e.s_{tgt}$ is a data store or pool). Finally, we also apply the same approach employed for shape disambiguation to detect and remove duplicate edge candidates.

Note that it is important to consider that this post-processing phase is intended to omit faulty predictions from our generated BPMN model, rather than to correct syntactic mistakes from the hand-drawn image. As such, the employed post-processing rules do not reflect cases that are observed in our training data but represent a design choice to improve the output of our approach. Figure 6.5 depicts the outcome of this step for the running example, highlighting that each drawn shape is associated with a single predicted shape in S and that each edge in E properly connects a valid source and target.

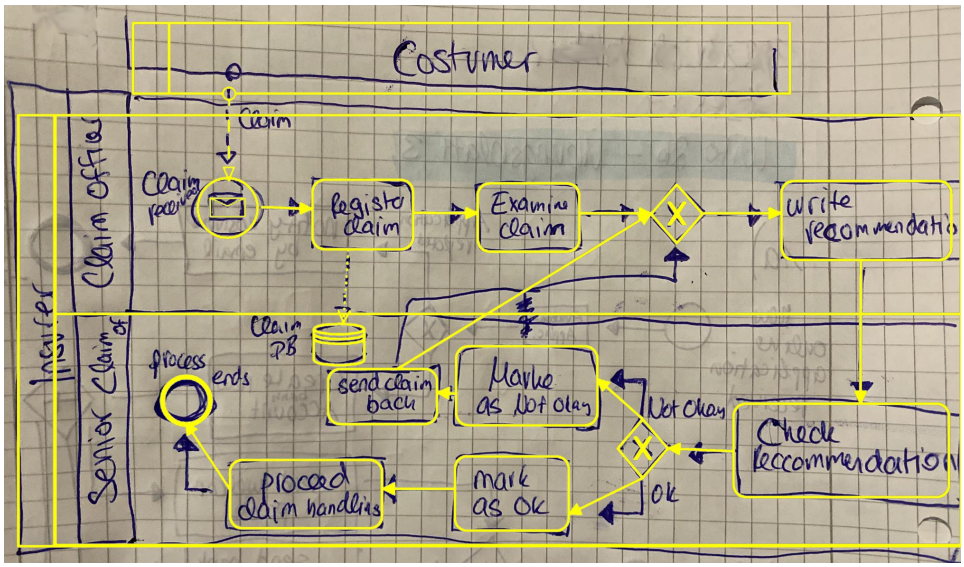


Figure 6.5: *Recognized BPMN Model*: the recognized BPMN model has been converted to an image and overlaid over the hand-drawn sketch. The entire process from the input image to the final BPMN model is automated.

6.1.3 Output Generation

The last step in our approach takes the final shapes and edges after structure recognition to create a BPMN process model in the BPMN 2.0 XML format. The XML consists of two main schemata: the actual process model and the BPMN DI schema, which defines the shape bounding boxes and the waypoints of edges.

Given the output from the previous step, the creation of the XML format is mostly trivial. For each predicted shape $s \in S$, we create a respective element in the XML file. When creating a BPMN DI edge element for each $e \in E$, we follow the typical convention and define the first and last waypoint as the points that intersect with the edge's source ($e.s_{src}$) and target ($e.s_{tgt}$) shapes, respectively. To that end, we shift each predicted keypoint (i.e., $e.src$ and $e.tgt$) to the nearest point on the bounding box boundary of the connecting shapes, except for gateways, where we shift the keypoint to the closest of the four diamond corner points.

6.2 Evaluation

To evaluate our approach, we trained and optimized it using the training and validation set of `hdBPMN`, and measured its performance on the test set.

6.2.1 Evaluation Setup

Below we elaborate on the details of our employed implementation, as well as the metrics, baselines, and configurations used to evaluate our approach.

Implementation. Our neural network implementation of the shape and edge recognition system Arrow R-CNN (Step 1 of our approach) is based on the *Detectron2* [104] object detection framework. To operationalize our extended augmentations, we use the *Albumentations* library [12]. For training, we use stochastic gradient descent with a batch size of 4 and a learning rate of 0.002. As the CNN backbone, we use ResNet-50 with FPN. We keep the remaining configurations originally used to train Arrow R-CNN [82].

Metrics. To evaluate our approach, we compare the sets of shapes and edges extracted by our approach to those in the manually annotated image (see Chapter 4), referred to as the *ground truth*. To quantify the performance, we follow related work in diagram recognition [9, 11, 103] and use different metrics to assess *shape* and *edge* recognition. A detected *shape* is considered a true positive if it is assigned the correct class and its bounding box overlaps sufficiently with its counterpart in the ground truth. Particularly, following [103], we consider this overlap sufficient if the bounding boxes have an overlap that exceeds an IoU threshold of 50%, which accounts for annotation inaccuracies in the bounding boxes of the ground truth. To quantify shape-recognition performance, we then use this notion of true positives to match the ground truth to the predicted shapes and compute the standard *precision*, *recall*, and F_1 scores. For *edge recognition*, a true positive requires that, as for shapes, the predicted class is correct and that its bounding box exceeds a 50% IoU threshold. However, we also require that a detected edge is associated with the correct source and target shapes, s_{src} and s_{tgt}. This means that edge recognition is indirectly affected by the shape-recognition quality: if a shape was not properly detected, all edges that connect to that shape result in false positives as well.[2]

Baselines and configurations. To demonstrate the efficacy of our approach, we compare its performance to two baselines: Baseline BL1 uses the Faster R-CNN object detector with the standard image augmentations coming with *Detectron2*. Since a standard object detector cannot recognize edges and their keypoints, BL2 corresponds to the original Arrow R-CNN system with its default image augmentation methods. To highlight the relevance of its individual components, we evaluate two configurations of our approach: Configuration C1 corresponds to the Arrow R-CNN system, enhanced with the extended augmentation (EA) of Section 6.1.1. Configuration C2 reflects our full-fledged approach, including the proposed BPMN-specific processing components. Note that we employ the same

[2]Note that an edge is still considered correct if its associated shapes are incorrectly classified.

Table 6.1: Overall approach results

Configuration	Shape		Edge	
	Micro F_1	**Macro F_1**	**Micro F_1**	**Macro F_1**
BL1: Faster R-CNN [77]	92.7	77.5	—	—
BL2: Arrow R-CNN [82]	93.2	80.8	85.5	76.0
C1: Arrow R-CNN + Ext. augm. (EA)	**95.7**	**86.2**	90.0	84.8
C2: Arrow R-CNN + EA + BPMN pp.	**95.7**	**86.2**	**91.8**	**87.4**

shape disambiguation procedure (proposed in Section 6.1.2) for all four systems, in order to ensure a fair comparison in terms of recall.

6.2.2 Results

This section presents the results of our evaluation for the `hdBPMN` test set, first in terms of overall results, before taking a detailed look at the results per BPMN element class.

Overall Results. The overall results presented in Table 6.1 reveal that the two configurations of `Sketch2BPMN` both outperform the baselines, achieving a micro F_1 score of 95.7 for shape recognition and 91.8 for edge recognition. Note that micro and macro measures differ because certain classes (e.g., *Tasks*) are much more common than others (e.g., specific events). However, the overall trends are consistent across the two.

Since each configuration in the table represents an extension of its predecessor, a closer look at the results reveals that the desired improvements associated with the gradual development from BL1 to C2 are achieved. In particular, we observe that Faster R-CNN (BL1), a general-purpose object detector, already recognizes more than 90% of all shapes, though it is unable to detect edges. The Arrow R-CNN approach (BL2), designed for the recognition of hand-drawn flowcharts, improves these results since it can also detect edges, achieving a macro F_1 of 76.0 for those, while also performing better in terms of shape recognition (80.8 versus 77.5). From BL2 to C1, we observe the improvements achieved by our extended augmentation step, which makes the recognition system more suitable to the diversity in the `hdBPMN` dataset, boosting the shape recognition from 80.8 to 86.2 and edge recognition from 76.0 to 84.8. Finally, we observe that the additional inclusion of BPMN-specific edge processing in C2 further improves the ability to recognize edges, achieving a macro F_1 of 87.4 and micro F_1 of 91.8.

Shape recognition. Table 6.2 provides detailed insights into the performance of our approach (with configuration C2), by depicting the results obtained per shape

Table 6.2: Shape and edge recognition results per class obtained for the test set

Group	Class	Precision	Recall	F_1	Count
Activity	Task	96.7	99.6	98.2	560
	Subprocess (collapsed)	100.0	72.2	83.9	18
	Subprocess (expanded)	n/a	0.0	n/a	2
	Call Activity	100.0	100.0	100.0	1
Event	Start Event	92.3	95.2	93.8	63
	End Event	94.5	96.3	95.4	107
	Message Start Event	94.2	94.2	94.2	52
	Message Interm. Catch Event	90.9	93.0	92.0	43
	Message Interm. Throw Event	78.3	94.7	85.7	19
	Message End Event	87.5	58.3	70.0	12
	Timer Start Event	83.3	83.3	83.3	12
	Timer Intermediate Event	90.0	75.0	81.8	12
Gateway	Exclusive Gateway	98.1	98.1	98.1	156
	Parallel Gateway	93.7	97.5	95.6	122
	Inclusive Gateway	n/a	0.0	n/a	1
	Event-based Gateway	90.0	81.8	85.7	11
Collaboration	Pool	95.3	96.8	96.0	125
	Lane	94.7	93.0	93.9	100
Data element	Data Object	96.3	96.9	96.6	161
	Data Store	95.5	84.0	89.4	25
Edges	Sequence Flow	95.7	94.0	94.9	1,216
	Message Flow	86.5	79.7	82.9	177
	Data Association	92.3	77.2	84.1	311
Overall	Macro avg.	92.7	80.9	86.4	3,307
	Micro avg.	94.8	92.7	93.7	3,307

and edge class. The table shows that our approach correctly recognizes the vast majority of shapes for most of the classes, achieving an F_1 score of at least 83.9 for the 13 classes that occur more than a dozen times. For other shape types, the number of data points is too low (in both the training and the test set), to sufficiently cover the spectrum of factors such as drawing styles and, therefore, to provide reliable evaluation results.

A posthoc analysis of the results reveals that the most difficult task for our approach is the correct classification of certain kinds of events. This comes as no surprise, though, the difference between some of the 8 kinds of events may only be due to marginal differences, such as a change in line thickness (start events), as well as different kinds of tiny envelopes (message events) and clocks (timer events). Especially in light of the diversity of shapes in our dataset, as highlighted in Figure 4.6, identifying such differences in hand-drawn models can already be highly complex for humans, let alone for an automated approach that lacks sufficient training examples for some of the rarer classes.

Edge recognition. The edge-levels results in Table 6.2 again demonstrate the overall strong performance of our approach, as well as that *sequence flows* (F_1 of 94.9) are easier to recognize than message flows (82.9) and data associations (84.1). To some extent, this can be attributed to the commonality of sequence flows and the fact that the latter two classes use dashed rather than continuous lines. However, it is also interesting to consider the different roles of these edges from a process modeling perspective. In particular, message flows connect (elements in) different pools, which are often placed relatively far from each other. This results in longer edges, which may also cross more nodes, and are, therefore, harder to analyze for an automated approach. For example, we observe that the distance between the head and tail keypoint is more than twice as high for message flows (499 pixels) as for sequence flows (216). For data associations, it is important to consider that elements related to the data perspective are often drawn last [24, p.177], whereas they also often are connected to numerous shapes, scattered throughout a model. These two factors thus commonly result in data associations that cross other edges or even shapes, which complicates their recognition.

6.3 Conclusion

In this chapter, we have developed a first shape and edge recognition baseline for `hdBPMN`. The evaluation results indicate that `hdBPMN` is much more challenging than existing datasets, which can be attributed to several aspects. As for shapes, `hdBPMN` features 21 different shape types, whereas existing datasets, in comparison, have between two and five different shape types. In addition, our analysis

reveals that the classification of the exact event type is very challenging, as the difference between two event types can be marginal, e.g., a change in line thickness to distinguish between a start and an end event. Concerning edges, we find that message flows and data associations are the most difficult to recognize, as they are often long-range, cross other arrows, and are drawn with dashed or dotted lines.

Besides providing a first baseline, we have also proposed the `Sketch2BPMN` method in this chapter. `Sketch2BPMN` improves `Arrow R-CNN` with an *edge post-processing* method that prevents invalid BPMN edges, and an *extended data augmentation* pipeline that simulates the varying properties of camera-based documents. We find that the extended data augmentations considerably improve the shape and edge recognition performance of `Arrow R-CNN`. We also observe that our rule-based approach to detect and identify invalid edges improves both the micro and macro edge F_1 score. However, even with these improvements, the F_1 score of both message flow and data association is still below 85%, which motivates the need for research on other edge recognition approaches. We take up this topic in the next chapter, where we present our first edge recognition approach inspired by visual relationship detection methods.

Chapter 7

DiagramNet

A limitation of the Arrow R-CNN method proposed in Chapter 5 is that it uses a heuristic for edge recognition, where the source and target shape are defined as the shapes that are closest to the tail and head keypoint, respectively. In the previous chapter, we presented Sketch2BPMN which improves this heuristic by identifying and correcting invalid BPMN edges. It, however, builds on a rule-based component for the identification of edges, which affects both the performance as well as the flexibility of the approach. Against this background, in this chapter, we propose `DiagramNet`, our first attempt to recognize edges using a learning-based approach inspired by research on visual relationship detection. DiagramNet includes a shape degree prediction network and an edge prediction network. The generated predictions of both networks are used to formulate edge recognition as a global optimization problem.

The research presented in this chapter is based on a paper with the title "*DiagramNet: Hand-drawn Diagram Recognition using Visual Arrow-relation Detection*" by Bernhard Schäfer and Heiner Stuckenschmidt [84]. We have developed this research in parallel to the `Sketch2BPMN` work presented in the previous chapter. Therefore, in their evaluation, both methods compare to the `Arrow R-CNN` method proposed in Chapter 5.

The remainder of this chapter is organized as the previous chapter. We first describe the `DiagramNet` method in Section 7.1. Next, we evaluate our approach in Section 7.2, and finally, we conclude this chapter in Section 7.3.

7.1 The `DiagramNet` Method

This section introduces `DiagramNet`, our proposed model for recognizing hand-drawn diagrams. As Figure 7.1 illustrates, `DiagramNet` decomposes diagram

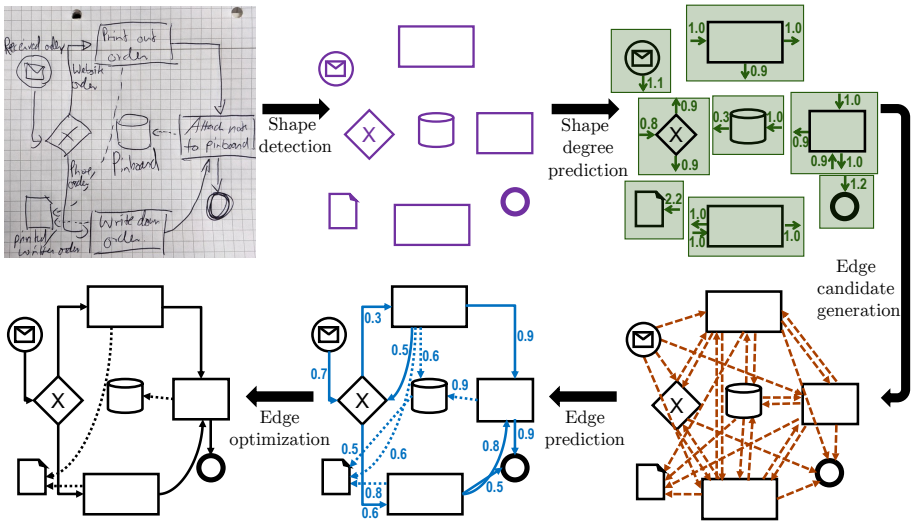


Figure 7.1: *DiagramNet overview*: Given an image, our approach first detects shapes. The local shape context (green) is then used to predict in- and out-degrees for each direction. Next, a pruned graph of edge candidates is generated. The edge prediction network classifies each candidate and predicts the drawn arrow path. Last, the edge optimization procedure uses the predicted shape degrees and edge scores to determine the final diagram.

recognition into a sequence of five stages. Below we introduce each stage in detail.

7.1.1 Shape Detection

We frame shape detection as an object detection task and use the Faster R-CNN [77] framework to detect the set of shapes V in an image. Each detected shape $v \in V$ is associated with a predicted bounding box $\mathbf{b}_v \in \mathbb{R}^4$, probability p_v, and class $c_v \in \mathcal{C}$, where $\mathcal{C}$ is the set of shape types of a modeling language. Our definition of $\mathcal{C}$ also includes shape types that are not connected to other shapes through arrows, such as the lane element in BPMN.

We follow prior work [82] and use Faster R-CNN with the feature pyramid network (FPN) extension [57]. During inference, we keep all shapes with $p_v \geq 0.7$. To eliminate duplicate detections, we also apply non-maximum suppression (NMS) with an intersection over union (IoU) threshold of 0.8 over all shape classes except lane. We exclude lanes since we observe a large bounding box overlap between lanes and their enclosing pools. During training, Faster R-CNN uses a multi-task loss, where the bounding box regression loss L_{loc} is weighted higher than the classification loss L_{cls}. Since we consider accurate classification equally important for shape detection, we decrease the weight for L_{loc} by a factor of 2.5.

As a result, the localization and classification losses are in the same range during training.

7.1.2 Shape Degree Prediction

Given the set of detected shapes V from the previous step, we predict the out-degrees $\mathbf{d}_v^+ \in \mathbb{R}^4$ and in-degrees $\mathbf{d}_v^- \in \mathbb{R}^4$ for each shape v. The vector $\mathbf{d}_v^+$ represents the predicted number of outgoing edges for shape v in directions left, top, right, and bottom. Figure 7.1 conceptually shows the predicted in- and out-degrees for significant shape directions. We use the predicted shape degrees in all three subsequent edge-centric steps:

1. *Edge candidate generation*: prune edge candidates whose source or target shape has an insignificant shape degree in the corresponding direction
2. *Edge prediction*: generate more targeted arrow-relation bounding boxes
3. *Edge optimization*: use shape degrees to create a globally coherent solution

We formulate degree prediction as a regression task and propose a *degree prediction network* that predicts the degrees for each shape given the visual shape features. Arrows are not always properly connected to their source and target shapes, i.e. sometimes there is some distance between the drawn arrow and the shapes they connect. Therefore, we pad each shape bounding box with 50px local context. Given the padded shape bounding box, we use RoIAlign [35] to extract a $28\times28\times256$ feature from the image features. We concatenate a 28×28 binary mask that encodes the location of the shape bounding box within the padded box. The resulting $28\times28\times257$ context-enriched shape representation is used as input for the degree prediction network.

Our degree prediction network is inspired by the MobileNet [38] architecture and contains a sequence of 6 depthwise separable convolutions. The spatial resolution of each feature map is downsampled twice by a factor of 2 using strided convolutions in the third and last depthwise convolution. An average pooling layer projects the resulting $7\times7\times256$ feature to a 256-d vector. After two fully connected layers with 256 dimensions each, two linear heads predict the direction out-degrees $\mathbf{d}_v^+ \in \mathbb{R}^4$ and in-degrees $\mathbf{d}_v^- \in \mathbb{R}^4$. As in [38], each network layer is followed by batch normalization and ReLU nonlinearity, with the exception of the final fully connected layer. Subsequent steps of our approach also require the total shape out- and in-degree. We define this as the sum over all directions, and denote it as $d_v^+ = \|\mathbf{d}_v^+\|_1$ and $d_v^- = \|\mathbf{d}_v^-\|_1$.

We train the degree prediction network using mean squared error loss. During training, we randomly shift each side of the ground-truth shape bounding box and vary the context padding to improve generalization.

7.1.3 Edge Candidate Generation

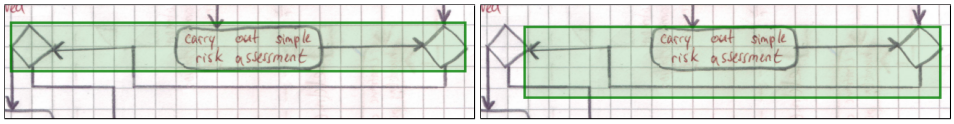
Given the set of detected shapes V, the fully-connected graph $V \times V$ has $O(|V|^2)$ directed edge candidates. To avoid considering all shape pairs, we prune edges by leveraging (1) syntactical rules of the modeling language and (2) predicted shape degrees. The syntactical rules in a modeling language govern how elements of the language can be combined. As we want to recognize unfinished diagrams or diagrams with modeling errors as drawn, we do not apply all syntactical rules of the modeling language. Instead, we empirically only consider rules that are not violated in the training set, which correspond to edges that are very unintuitive to be modeled as such. As an exemplary rule for the BPMN language, we prune all edge candidates between gateway (diamond) and business object (file and database) shapes, as illustrated in Figure 7.1. Given the initial fully-connected graphs of the `hdBPMN` test set, syntactical pruning eliminates 31.8% of all candidate edges, and the resulting graphs have a mean graph density of 0.75.

We further prune the candidate graph using the predicted shape degrees from the previous step. We prune each edge (u, v) where $d_u^+ < \alpha$ or $d_v^- < \alpha$, i.e. we prune edges where the degree of at least one shape is below a threshold. We find that $\alpha = 0.05$ sufficiently balances precision and recall. Our degree-based pruning approach is also illustrated in Figure 7.1, where the outgoing edges for the data object (file icon) have been pruned. On the `hdBPMN` test set, degree pruning eliminates an additional 11.3% of all candidate edges and further reduces the mean graph density to 0.62.

7.1.4 Edge Prediction

Given the set of candidate edges from the previous step, our edge prediction approach is concerned with three major aspects: First, we identify the arrow region for each shape pair, which we refer to as *arrow-relation bounding box*. Second, during *edge classification*, we classify each candidate into one of the edge classes (including a background class). Third, we predict the arrow path as a sequence of k equidistant points, which we refer to as *path prediction*. In the following, we present the details of each aspect.

Arrow-relation bounding box. As discussed in Section 2.4, visual relationship detection methods commonly define the relation region for an object pair as the union of their bounding boxes. This approach is not well suited for arrow-relation recognition since (1) the respective arrow is not necessarily located within this region and (2) the shape bounding boxes in their entirety are not required to recognize the arrow. Therefore, we propose an approach that leverages the predicted shape degrees introduced in Section 7.1.2. For each edge (u, v), we use the predicted



(a) union-based (standard) **(b)** direction-based (ours)

Figure 7.2: *Arrow-relation bounding box.* (a) shows the arrow-relation bounding box generated as the union of both diamond shape boxes. In (b), the bottom side of the box has been padded, and the left side has been shrunk. The box of our approach captures the hand-drawn arrow more closely.

shape out-degrees $\mathbf{d}_u^+$ and in-degrees $\mathbf{d}_v^-$ to (1) *pad* shape boxes on sides with predicted arrows and (2) *shorten* shape boxes on irrelevant sides. We then generate the arrow-relation bounding box as the union of the transformed shape boxes. Before we go into further detail, Figure 7.2 shows an example of this approach.

An in- or outgoing arrow of a shape that does not point toward its opposing shape is a strong indicator that the edge is not drawn as a straight arrow. Therefore, we *pad* the shape bounding box on sides with a predicted arrow. Since we are only interested in the existence of in- and outgoing arrows in each direction, we binarize the predicted shape degrees using a threshold $T = 0.3$. Given an edge candidate (u, v), we pad u with local context on sides where $\mathbf{d}_u^+ > T$ and pad v on sides where $\mathbf{d}_v^- > T$. We find that a local context of 50px sufficiently covers the distribution of diverse arrow types such as curved and elbow arrows. In Figure 7.2, the arrow between the two diamond gateway shapes exits the source shape in the bottom direction. After padding the source shape in the bottom direction, the arrow-relation box fully contains the drawn arrow.

Further, we *shorten* the shape bounding boxes on sides where there is no predicted arrow on the side itself and its adjacent sides. The left diamond gateway in Figure 7.2 has only one incoming edge from the right side. Thus, we shorten its left side for arrow relations with this shape as target. As a result, the arrow-relation box in the right image does not contain the target shape. We also shorten BPMN pool shape boxes, which differ from other arrow-connected shapes in that their larger side usually spans most of the diagram. To avoid overly large arrow-relation boxes for shape pairs that involve a pool, we limit the longer side of pools to 100px.

Edge classification and path prediction. We propose an *edge prediction network* that classifies each edge candidate and predicts the drawn arrow path. The network is similar to the shape degree prediction network introduced in Section 7.1.2. We extract a 28×28×256 feature representation using the previously defined arrow-relation bounding box. Next, we concatenate two 28×28 binary masks, one for the source and one for the target shape. These binary masks indicate the spa-

tial proximity in which to expect the arrowhead and tail. For shapes that are not contained in the arrow-relation box, we set the border pixels closest to the shape to one. We use the same MobileNet-inspired architecture to project the resulting 28×28×258 feature to a 256-d visual feature vector. The predicted shape classes are then concatenated as one-hot encoded vectors. This semantic information is added so that the network predicts edge classes consistent with the source and target shape classes. After two fully connected layers, a linear layer with a softmax function predicts the edge probability p_e and edge class $c_e \in \mathcal{R}$, where the set of edge classes $\mathcal{R}$ includes a negative "background" class. A second linear layer predicts the encoded coordinates of k arrow keypoints. Following our `Arrow R-CNN` method presented in Chapter 4, we encode arrow keypoints relative to the relation bounding box.

We train the edge classifier using cross-entropy and train the arrow keypoint predictor using smooth L_1 loss as in [83]. Since our model requires a constant number of arrow keypoints, we sample k equidistant keypoint targets from the ground-truth arrow path. We find that $k = 5$ captures the majority of arrow drawing styles, and use this configuration throughout all experiments. We average the loss over all keypoints and multiply the keypoint loss by a factor of 10 to ensure that the classification and keypoint losses are in the same range. During training, we randomly sample 80% of the ground-truth edges as positive edge candidates and add twice as many negative candidates from the set of candidate edges from the previous step. As in Section 7.1.2, we randomly shift each side of the ground-truth shape bounding boxes and the generated arrow-relation bounding box to improve generalization.

7.1.5 Edge Optimization

Given the predicted probability p_e for each edge candidate $e \in E$ from the previous step, a standard postprocessing method is to use a score threshold T and keep each edge e where $p_e \geq T$. However, we can improve upon this baseline by also considering the predicted shape degrees d_v^+ and d_v^- introduced in Section 7.1.2. Consequently, we employ a global optimization that minimizes the difference between predicted shape degrees and the number of accepted edges that connect to each shape. We denote $E^+(v)$ as the outgoing and $E^-(v)$ as the incoming edges of shape v. Let $\mathbf{x} \in \{0,1\}^{|E|}$ be a binary vector where $x_e = 1$ indicates that edge e is accepted. We define the degree penalty terms $\deg^+(v)$ and $\deg^-(v)$ as

$$\deg^+(v) = \left(d_v^+ - \sum_{e \in E^+(v)} x_e \right)^2 \quad \deg^-(v) = \left(d_v^- - \sum_{e \in E^-(v)} x_e \right)^2 \tag{7.1}$$

Each term measures the squared difference between the predicted degree of v and the number of accepted edges that connect to v. We formulate the optimization problem as

$$\mathbf{x}^* = \underset{\mathbf{x}}{\operatorname{argmax}} \left[\sum_{e \in E} (p_e - T)\, x_e - \lambda \sum_{v \in V} \deg^+(v) - \lambda \sum_{v \in V} \deg^-(v) \right] \tag{7.2}$$

The score threshold T is now a soft threshold, and λ is a hyperparameter for weighting the degree terms. We set $T = 0.4$ and $\lambda = 0.4$ after experimenting on the `hdBPMN` validation set and use these in our evaluation. To reduce the number of edge candidates, we already reject all candidates with $p_e < 0.05$ prior to optimization. We obtain three sets of disjunct edges from our proposed edge optimization procedure: the accepted edges $E[\mathbf{x}^*]$, the rejected edges $E[\neg\mathbf{x}^*]$, and the edges rejected before optimization $E[p_e < 0.05]$. The bottom-left diagram in Figure 7.1 conceptually shows the final recognition result with the accepted edges after optimization.

7.2 Evaluation

In this section, we evaluate the performance of `DiagramNet` on several datasets. We describe the evaluation setup in Section 7.2.1, before presenting the results in Section 7.2.2.

7.2.1 Evaluation Setup

Datasets. As in the previous chapter, we use a preliminary version of `hdBPMN` with 502 instead of 704 images. In the experiments of the previous chapter, we found that the classification of the exact event type is a major challenge in `hdBPMN`. As this work focuses on edge recognition, we mitigate this recognition challenge by reducing the set of event categories. Specifically, we only distinguish between untyped-, message-, and timer events. To demonstrate the general applicability of `DiagramNet`, we also compare its performance to existing methods on two popular online flowchart datasets, `FC_A` and `FC_B`.

Implementation Details. We implement DiagramNet using the detectron2 [104] framework. We use a ResNet50 with FPN [57] as the backbone network and use models pretrained on the COCO dataset. We sum the losses of all DiagramNet components and jointly train the network. We use the same training procedure as in [82], except that we multiply the learning rate with 0.4 after iterations 50k and 70k. Our augmentation pipeline is also similar to [82], we resize images to a

longer size of 1333px and randomly shift, scale, rotate and flip the image. As in the previous chapter, in order to simulate the properties of camera-based images, we randomly add Gaussian noise, change brightness and contrast, and randomly shift hue, saturation, and value.

Arrow R-CNN Baselines. We compare our approach to the state-of-the-art method Arrow R-CNN [82]. To that end, we reimplemented Arrow R-CNN using the detectron2 framework. The image augmentation methods in [82] target online diagram datasets. Thus, we use two Arrow R-CNN baselines, one with the original augmentations and backbone used in [82], and one with our proposed augmentations and backbone for fair comparison. The edges in `hdBPMN` are annotated as a sequence of waypoints. Since Arrow R-CNN requires edge bounding boxes, we define the edge bounding box as the smallest possible box that includes all waypoints. For straight arrows drawn parallel to an axis, this results in bounding boxes with a width or height of 1. In early experiments, we noticed that this bounding box definition has a large negative effect on Arrow R-CNN's recognition performance. We thus introduce a minimum arrow bounding box width and height of 20px for an image resized to a longer side of 1000px.

Metrics. Prior works use bounding-box centric methods to evaluate shape and edge recognition in hand-drawn diagrams and require at least 80% bounding box overlap between predicted and ground-truth objects for both shapes and edges. In addition, edges need to be matched to the correct source and target shapes, which we refer to as the shape-match criterion. The problem with evaluating arrows based on bounding boxes becomes clear when considering a straight horizontal arrow with a height of e.g. 4 pixels. Using the common metric, a predicted arrow bounding box that is off by 1 pixel would be considered false positive. We argue that it is more intuitive to evaluate edge recognition only based on the shape-match criterion, and thus do not consider the arrow bounding box. The shape-match criterion has one disadvantage, though: consider a shape with 5 connected edges that has been accurately localized but misclassified. With the shape-match criterion, all edges connected to this shape are inevitably evaluated as false positives. To separate edge and shape recognition performance, we also report the edge recognition metrics when using ground-truth shapes instead of predicted shapes.

For measuring precision, recall, and F_1-score, we keep all shapes with $p_v \leq 0.7$, and use the set of accepted edges from the edge optimization procedure. Since both shape and edge classes are imbalanced, we report macro scores by taking the average over the individual class scores. We also measure mean average precision (mAP) for shape and edges. Since average precision requires a consolidated edge ranking, we concatenate the sets $E[\mathbf{x}^*]$, $E[\neg\mathbf{x}^*]$, and $E[p_e < 0.05]$ introduced in Section 7.1.5 in that order.

Table 7.1: Overall results on the `hdBPMN` test set.

	Shape				Edge			
Method	**Prec.**	**Rec.**	**F_1**	**mAP**	**Prec.**	**Rec.**	**F_1**	**mAP**
Faster R-CNN [77]	93.1	89.5	90.9	88.7	–	–	–	–
Arrow R-CNN [82]	96.6	90.1	92.9	89.8	86.1	83.3	84.6	79.0
Arrow R-CNN† [82]	96.4	92.0	94.0	91.7	87.2	83.7	85.4	80.1
DiagramNet (ours)	96.1	93.7	**94.7**	**94.8**	92.2	88.9	**90.5**	**90.8**
Evaluation with ground-truth shapes								
Arrow R-CNN [82]	–	–	–	–	91.0	88.1	89.5	87.7
Arrow R-CNN† [82]	–	–	–	–	92.1	88.3	90.1	89.5
DiagramNet (ours)	–	–	–	–	96.0	92.0	**93.9**	**95.8**

†Uses our proposed image augmentations and backbone for fair comparison

Table 7.2: Edge prediction ablation study to quantify the impact of syntax pruning (SP), keypoint prediction (KP), degree prediction (DEG), and direction prediction (DIR).

				Shape				Edge			
SP	**KP**	**DEG**	**DIR**	**Prec.**	**Rec.**	**F_1**	**mAP**	**Prec.**	**Rec.**	**F_1**	**mAP**
✓	-	-	-	96.0	93.1	94.4	95.5	74.9	69.0	71.6	75.2
✓	✓	-	-	96.2	94.0	95.0	**95.8**	84.6	80.6	82.5	85.1
✓	✓	✓	-	96.0	94.6	**95.2**	95.0	90.9	85.4	88.0	88.3
✓	✓	-	✓	96.1	93.7	94.7	94.8	92.2	88.9	**90.5**	**90.8**
-	✓	-	✓	95.7	93.0	94.2	96.1	92.1	87.7	89.8	89.2

7.2.2 Results

Overall results. Table 7.1 shows that DiagramNet has a slightly better shape recognition performance than the Arrow R-CNN baselines, and significantly outperforms Arrow R-CNN in edge recognition. The edge evaluation using ground-truth shapes confirms this finding. For comparison, we also include the shape recognition results of a Faster R-CNN object detector that uses our backbone and standard image augmentations (horizontal flipping and random resizing). For a qualitative assessment of our method, Figure 7.3 shows the recognized diagram for a test set image. As demonstrated, DiagramNet is able to recognize a diagram from a camera-based image that includes background objects. The total inference time per image is 319ms on average, measured on a Tesla V100 GPU with 16GB memory. This includes the proposed edge optimization procedure, which takes 129ms on average.

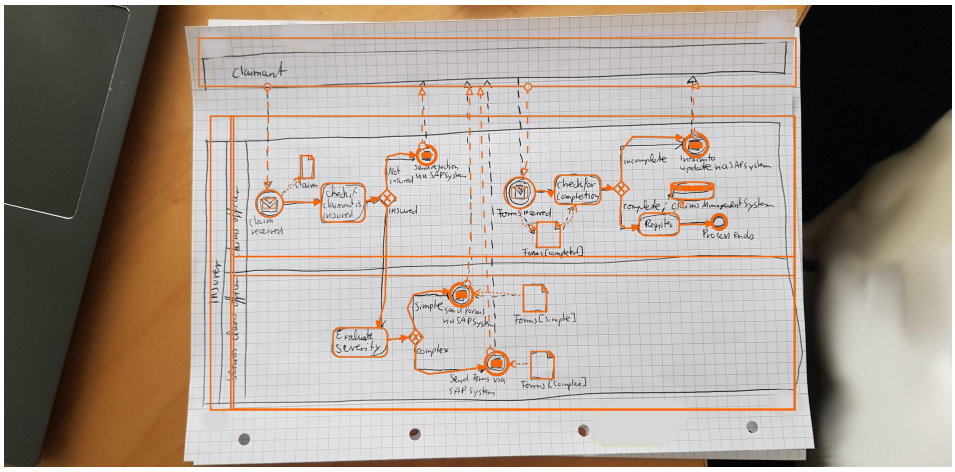


Figure 7.3: `hdBPMN` test set image with predicted diagram overlay (orange). All shapes and edges have been correctly recognized. The BPMN shapes are rendered using the predicted shape classes and bounding boxes. The line segments of each edge are visualized by connecting the predicted arrow keypoints.

The ablation study in Table 7.2 shows how each edge component contributes to the end result. Since we only vary edge-related components, the shape recognition performance is similar in all settings. In the first experiment, we create the arrow-relation bounding boxes as the union of both shape boxes, which is standard in visual relationship detection. Also, we predict neither arrow keypoints nor shape degrees. We observe that this results in a relatively low edge F_1 score (71.6). Training with arrow keypoint prediction (KP) improves edge F_1 considerably from 71.6 to 82.5, even though the edge metric only takes edge classification into account. This indicates that the edge predictor greatly benefits from the keypoints as additional supervision targets. We also conduct a degree ablation study (DEG) in which the shape degree predictor only predicts the overall shape in- and outdegree, instead of predicting the degree for each direction. Like in our proposed method, we use these predicted degrees to prune candidate edges and perform global edge optimization. As a result, we observe that edge F_1 improves from 82.5 to 88.0. Next, our proposed shape direction predictor (DIR) allows us to generate more targeted arrow-relation bounding boxes. We observe that this further boosts edge F_1 from 88.0 to 90.5. In the final experiment, we disable syntax pruning. As expected, we observe that this leads to a small decrease in shape and edge F_1.

To demonstrate the general applicability of our method, we also trained and evaluated DiagramNet and Arrow R-CNN on two popular online flowchart datasets. The results in Table 7.3 show that DiagramNet slightly outperforms the state-of-

Table 7.3: Overall results on the online flowchart datasets FC_A [3] and FC_B [11].

	FC_A		FC_B	
Method	**Shape F_1**	**Edge F_1**	**Shape F_1**	**Edge F_1**
Arrow R-CNN [82]	**99.8**	96.1	98.8	96.7
DiagramNet (ours)	**99.8**	**96.7**	**99.2**	**97.5**

Table 7.4: Shape and edge recognition results

Group	Class	Precision	Recall	F_1	AP	Count
Activity	Task	97.7	99.5	98.6	99.6	560
	Subprocess	93.8	71.4	81.1	79.3	21
Event	Untyped Event	96.6	99.4	98.0	99.9	170
	Message Event	96.2	99.2	97.7	98.8	126
	Timer Event	100.0	91.7	95.7	95.8	24
Gateway	Exclusive Gateway	98.7	98.7	98.7	99.9	157
	Parallel Gateway	95.2	97.5	96.4	97.7	122
	Event-based Gateway	90.9	90.9	90.9	82.6	11
Collaboration	Pool	99.2	99.2	99.2	100.0	125
	Lane	91.9	91.0	91.5	94.2	100
Data element	Data Object	96.9	97.5	97.2	98.6	161
	Data Store	95.7	88.0	91.7	91.0	25
Edges	Sequence Flow	92.9	94.6	93.7	94.3	1,216
	Message Flow	89.8	84.7	87.2	87.4	177
	Data Association	93.9	87.3	90.5	90.6	315

the-art method Arrow R-CNN on flowcharts. However, the near-perfect shape results and the edge F_1 scores of at least 96.7 also motivate the need for more difficult datasets such as hdBPMN.

Shape and edge recognition. Table 7.4 provides detailed results per shape and edge class. For 8 out of 12 shape classes, DiagramNet achieves an F_1 score of at least 95. For three out of the remaining shape classes, the number of data points is very low (Subprocess, Event-based Gateway, Data Store). For lanes, we observe that the rectangle shapes are often not fully drawn, as illustrated in Figure 1.2. The edge class results show that sequence flows (F_1 of 93.7) are easier to recognize than data associations (90.5) and message flows (87.2). For message flows, besides the smaller number of training data points, the challenge is that they often connect

shapes far from each other, as illustrated in Figure 7.3. Also, message flows and data associations are both dashed arrows, and we observe a large variety of drawing styles for the dashed lines.

We also measure the accuracy of our shape degree prediction network. To this end, we compare the four ground-truth degrees of each correctly detected shape with the rounded predicted degrees. On the test set, we observe a higher accuracy for ingoing edges (97.8%) than for outgoing edges (96.9%), which suggests that arrow heads are easier to recognize than arrow tails.

7.3 Conclusion

In this chapter, we proposed DiagramNet, a new method for recognizing the shapes and edges of hand-drawn diagrams. DiagramNet includes a shape degree prediction and an edge prediction network. The generated predictions are used to 1) prune the candidate graph, 2) generate informed arrow-relation boxes, and 3) formulate edge recognition as a global optimization problem. We also propose a new metric to assess edge recognition performance independent of arrow bounding boxes. The experimental results demonstrate that DiagramNet considerably outperforms Arrow R-CNN in edge recognition on the challenging `hdBPMN` dataset. The ablation study shows that this is partially due to our direction-based approach for creating arrow-relation bounding boxes. This approach allows us to also capture arrows that connect their respective shapes with a detour.

DiagramNet has also been used by Schumacher in 2021 to recognize hand-drawn UML diagrams [87]. In his bachelor thesis, Schumacher modified DiagramNet such that it can recognize undirected edges and conducted an evaluation on a small dataset of UML class diagrams. The fact that DiagramNet can be applied to other types of arrow-connected diagrams with minimal changes demonstrates its general applicability.

Of course, our work is subject to several limitations. First, the edge recognition approach of DiagramNet reaches its limits when the aforementioned detour is too large, and it also creates overly large relation boxes for arrows that directly connect two shapes. This leads to the research question of whether a complementary approach, which combines the strengths of Arrow R-CNN and DiagramNet, would provide even better results. In such an approach, the object detector would detect arrow instances, and the bounding box of an arrow could then be used to create a more fitting arrow-relation box, which is then provided as input to an edge prediction network. Second, this chapter leaves aside the text modality to focus on the challenges associated with shape and edge recognition. Here, the research question is how to design a method that can effectively detect and recognize textblocks,

and relate textblocks with shapes and arrows. We address both research questions in the next chapter, where we present `Sketch2Process`, our unified diagram recognition method that combines the strengths of the methods presented in this and previous chapters.

Chapter 8

Sketch2Process

This chapter introduces Sketch2Process, the first approach that provides end-to-end recognition of hand-drawn BPMN models from images. Unlike the methods presented in the last two chapters, which focus on recognizing shapes and edges, Sketch2Process also targets label recognition. Apart from the expanded scope, Sketch2Process also achieves a better performance in recognizing shapes and edges. For edge recognition, we improve the visual relationship detection component presented in the last chapter. Specifically, the component proposed in this chapter is designed such that it can also recognize arrows that connect two shapes with a large detour. In addition, we improve the overall recognition performance through a crop augmentation procedure, which allows our approach to better handle model layouts not encountered during training.

The research presented in this chapter is based on the paper "*Sketch2Process: End-to-end BPMN Sketch Recognition Based on Neural Networks*" by Bernhard Schäfer, Han van der Aa, Henrik Leopold, and Heiner Stuckenschmidt [86]. As part of this research, we also substantially extended the hdBPMN dataset. In particular, we manually annotated all labels and individual words in the dataset, and also increased the size of the dataset from 502 to 704 images.

The remainder of this chapter is organized as follows. Section 8.1 presents our Sketch2Process approach. Section 8.2 describes the evaluation procedure and the experimental results, and we conclude this chapter in Section 8.3.

8.1 The Sketch2Process Method

This section introduces Sketch2Process, our approach for recognizing a hand-drawn BPMN model from an image. As visualized in Figure 8.1, our approach consists of three main steps: object detection, edge recognition, and label recognition.

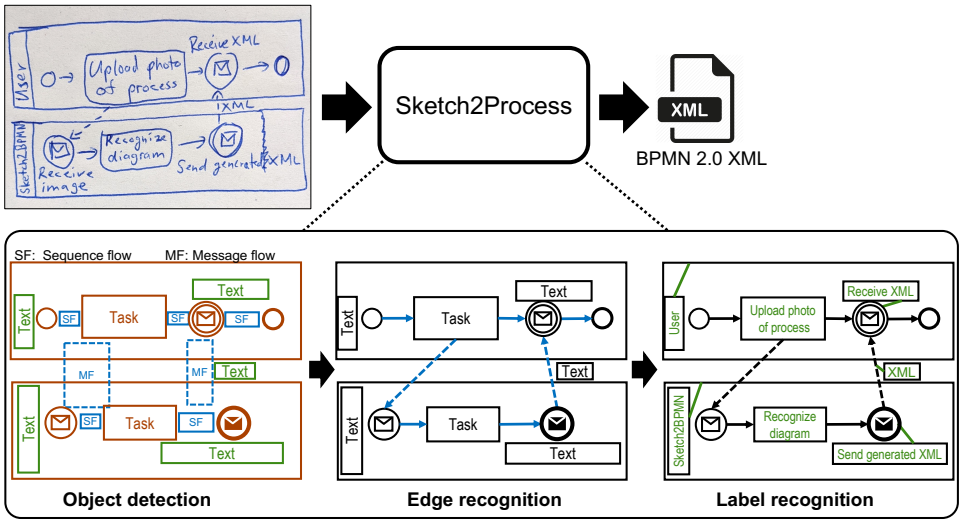


Figure 8.1: `Sketch2Process` overview: Given an image, we first detect shapes, arrows, and textblocks (object detection). Then, we identify the drawn arrow path and the shapes that each arrow connects (edge recognition). Next, we decode the textual content within each textblock and identify the shape or edge that each textblock labels (label recognition). Finally, we generate a BPMN 2.0 XML file suitable for process modeling tools.

The *object detection* step aims to detect all objects (shapes, arrows, and textblocks) that are part of the input image, characterizing each object as a bounding box and a predicted object class. While this step completes the recognition of shapes in the drawing, detected arrow objects are further processed in the *edge recognition* step. In this step, we identify the source and target shapes that each arrow connects, along with the drawn path of the arrow. Afterward, the *label recognition* step decodes the textual content of each textblock and assigns it as a label to a corresponding shape or edge. Finally, `Sketch2Process` generates and returns a BPMN 2.0 XML file that captures the detected BPMN model.

In the following, we provide details on the individual approach steps (Sections 8.1.1 to 8.1.3) and the subsequent output generation (Section 8.1.4). Finally, given that we employ three neural networks throughout our approach, we show how these networks are connected and jointly trained (Section 8.1.5).

8.1.1 Object Detection

In this first step, our approach aims to detect all objects (shapes, arrows, and textblocks) contained in a given image. As in the previous chapters, we use *Faster R-CNN* [77] as our object detector of choice. During inference, we divide the de-

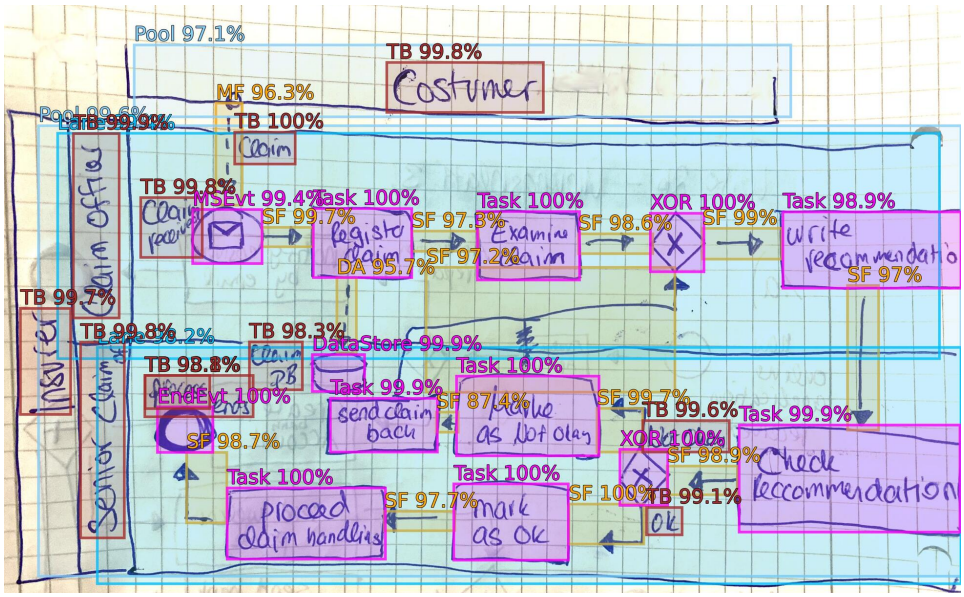


Figure 8.2: *Object detection* output: the network has detected shape (blue), arrow (orange), and textblock (brown) objects.

tected objects into the (disjoint) sets of shapes S, arrow objects A, and textblocks T, based on their predicted object classes. Note that concerning shapes, set S only includes those objects with a classification score s equal to or above a score threshold for shapes, τ_s (we use 0.5 as default and test further values in our evaluation experiments).

Figure 8.2 shows the outcome of this step. As can be seen, the object detector aims to detect bounding boxes that as closely encompass shapes and textblocks as possible. For example, the bounding boxes of the *Check Recommendation* activity and *Costumer* [sic] textblock leave little space between the box and the actually drawn element. In contrast, for arrow objects in A, the object detector is trained to establish bounding boxes so that the box connects the actual source and target shapes of the arrow, rather than just encompassing the arrow's drawn path. We describe the ground truth used for this purpose in Section 8.1.5. As, e.g., can be seen for the sequence flow arrows surrounding the XOR choices in Figure 8.2, this can result in a bounding box that is considerably larger than the drawn arrow. By detecting arrow objects in this manner, it is considerably easier to turn them into edges that connect shapes, as done next.

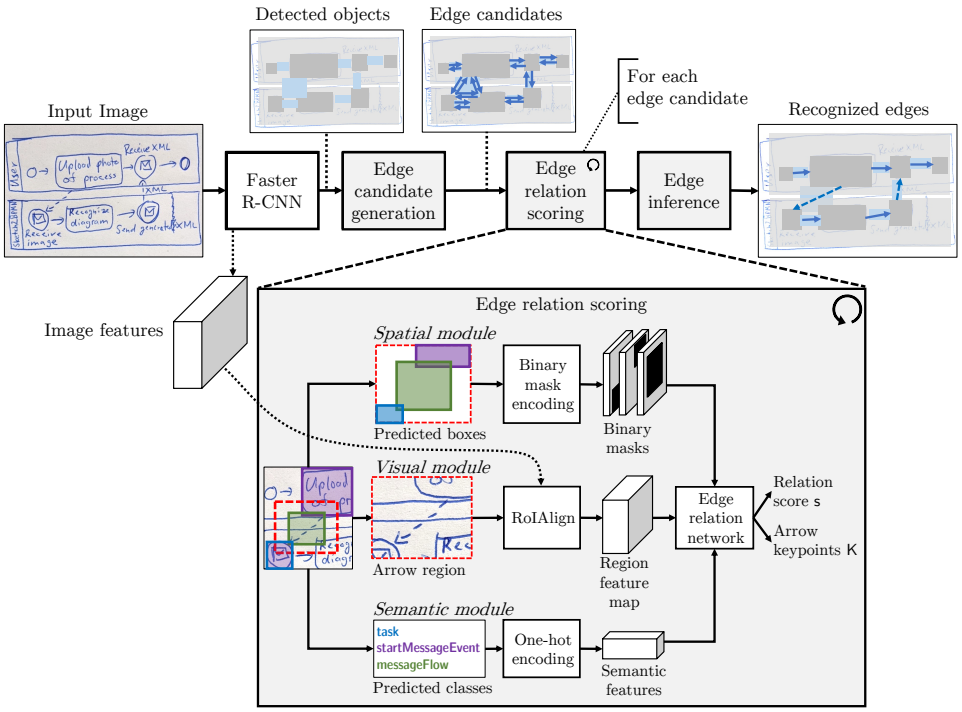


Figure 8.3: *Edge recognition*: Given the objects detected by Faster R-CNN, *edge candidate generation* produces a set of edge candidates for every arrow. For each edge candidate $src \xrightarrow{a} tgt$, the *edge relation scoring* procedure predicts the relation score and arrow keypoints from the features extracted by three modules. Finally, the *edge inference* procedure identifies the most likely edge candidate for each arrow and eliminates duplicate edges.

8.1.2 Edge Recognition

In this step, our approach aims to recognize the BPMN edges indicated by the drawn arrows detected in the previous step. The input for this step consists of the detected shapes S and arrow objects A, and the image features $feat$, capturing the image representation learned by Faster R-CNN. For each arrow object $a \in A$, our approach strives to recognize the two shapes that the edge connects and the edge's drawn arrow path. This is captured in the form of a tuple $e = (a, src, tgt, s, K)$, where $src, tgt \in S$ are the source and target shapes that the arrow connects, s its score, and K the drawn path of the arrow, represented as a sequence of keypoints. The output of this step, then, is a set of recognized edges E.

As visualized in Figure 8.3, we decompose edge recognition into three stages. First, *edge candidate generation* produces a set of edge candidates for every detected arrow $a \in A$. Each edge candidate $src \xrightarrow{a} tgt$ relates an arrow a to a pair of

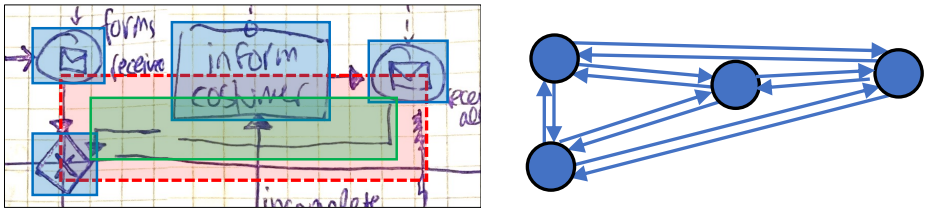


Figure 8.4: *Edge candidate generation*: Given an arrow (green), each shape that intersects with the arrow region (red) is considered a candidate. After, we create $O(n^2)$ edge candidates (blue arrows) for the arrow and its n candidate shapes.

possible source and target shapes in its proximity. Second, we use a trained *edge relation network* to process each edge candidate $src \xrightarrow{a} tgt$ and predict the likelihood s that arrow a indeed connects src and tgt, as well as the arrow path K that was drawn to connect src and tgt. Lastly, given the set of edge candidates detected for all arrows E_C, the *edge inference* procedure determines the final set of edges $E \subseteq E_C$. This procedure involves finding the most likely edge candidate for each arrow and eliminating duplicate edges. In the following, we introduce each stage in detail.

Edge candidate generation. Given an arrow $a \in A$, we first identify all potential pairs of source and target shapes that the arrow might connect. To that end, we take all shapes that are considered to be in proximity to the arrow's bounding box. Although the object detector already aims to connect an arrow's bounding box to its source and target shapes (see Section 8.1.1), we still need to account for minor prediction inaccuracies that could result in an arrow's bounding box not being fully connected to its corresponding shapes.

Therefore, we determine if an arrow a and a shape s are in proximity to each other by expanding the arrow's bounding box $a.b$ in a manner relative to its width and height. Specifically, we pad each side of $a.b$ by an arrow padding percentage pad_a of the box's weight and height, respectively.[1] We refer to this extended arrow bounding box as the (context-enriched) *arrow region*, $a.r$. Figure 8.4 shows an example, with the arrow's bounding box in green and the arrow region in red. Based on the experiments reported on in Section 8.2.2, we found that a padding of $pad_a = 10\%$ is best suited for this purpose.

The set of candidate shapes for an arrow a then includes all shapes that overlap with the arrow region, i.e., it contains any shape $s \in S$ of which its bounding box $s.b$ intersects with the arrow's region $a.r$. For instance, in Figure 8.4, the arrow region overlaps with the bounding boxes of all shapes, thus resulting in four shape

[1]To account for overly small or large arrows, we enforce a minimum padding value of 15px and a maximum of 100px.

candidates for the arrow. There are two exceptions to this procedure, related to the specifics of collaboration shapes in BPMN. First, we only consider pool shapes as candidates for arrows classified as message flow, as the other edge types do not connect to pools in BPMN. Second, we do not consider lane shapes, given that no edge type connects to these shapes at all. Removing invalid collaboration shape candidates greatly reduces the total number of candidates to evaluate, as, in the example of sequence flows, we would otherwise create a candidate shape for both the lane and pool that the arrow belongs to.

Given n candidate shapes, we create $n * (n-1)$ directed edge candidates of the form $src \xrightarrow{a} tgt$, with $src \neq tgt$, i.e., we consider all pair-wise combinations as the potential source and target shapes for arrow a, as illustrated in Figure 8.4. We intentionally do not apply any heuristics to further prune the set of shape pair candidates. For example, in Figure 8.4, it seems unlikely that the predicted arrow bounding box connects the two leftmost shapes. However, in some cases, arrows are drawn in such a way that they connect two shapes with a large detour. Instead of applying heuristics to detect such scenarios, we, therefore, opt for a learning-based approach that can exploit these spatial bounding box correlations to decide if an arrow connects a shape pair, as detailed in the second stage below.

While the number of edge candidates is quadratic with respect to the number of shape candidates, this is generally not problematic in practice. In particular, we observe that the majority of arrows (e.g., 71% in the training split of `hdBPMN`) have two candidate shapes, which results in only two edge candidates, one per direction. This is, e.g., seen for the two short sequence flow arrows in Figure 8.4.

Edge relation scoring. In the second stage, we use an edge relation network to predict the likelihood that an edge candidate $src \xrightarrow{a} tgt$ indeed connects shape src to tgt, and to predict the drawn path of the arrow (captured as a sequence of keypoints K). As shown in Figure 8.3, the network is provided an edge candidate as input, comprising the bounding boxes and class predictions of the three objects, and the arrow region $a.r$. As depicted, the edge relation scoring procedure uses three modules to analyze different kinds of features. This modular approach is inspired by existing works [19,118], where relationships between objects in images are also detected using a spatial, a semantic, and a visual module. In the following, we describe the three modules in detail and explain how the edge relation network predicts the edge relation score and keypoints given the encoded features.

Spatial module. The spatial module encodes spatial features for each edge candidate, i.e., the (relative) locations of the bounding boxes of the arrow and the two associated shapes. For each predicted box of the three objects, the spatial module generates a 28×28 binary mask that indicates the location of the box within the arrow region $a.r$, as illustrated in Figure 8.3. Each binary mask is initialized

with zeros and then filled with ones for each bounding box pixel that is located within $a.r$. Specifically, we first compute the intersection bounding box of each shape and $a.r$. Next, we normalize the intersection boxes by dividing their x and y coordinates by the width and height, respectively, of $a.r$. Then, we multiply each coordinate by 28 and obtain boxes in the range $[0, 28]$, which we finally convert into binary masks.

The binary masks instruct the network which task it is supposed to solve, i.e., they indicate the source and target shape between which the network should try to recognize an edge. In addition, the spatial features can be used by the network to assess the likelihood that arrow a connects the source and target shapes, src and tgt. For instance, spatial features capture that corresponding source and target shapes are typically located on opposite sides of an arrow's bounding box, as seen for the example in Figure 8.3. However, to actually recognize an edge, the network also needs the visual features of the arrow region. These features are generated by the visual module, as we describe next.

Visual module. The visual module generates a learned $28 \times 28 \times 256$ visual feature representation of the arrow region $a.r$. These visual features can be used to assess the likelihood that an arrow connects to particular shapes, e.g., by considering the proximity and direction of a drawn arrow, whereas they are also used to identify the keypoints in the drawing, i.e., an arrow's start, end, and notable points in between. As discussed in Section 2.2.2, Faster R-CNN uses the *RoIAlign* mechanism to cut out the part of the image features that correspond to the bounding box of an object proposal. Figure 8.3 shows that the visual module uses the same mechanism to extract the features for $a.r$.

As discussed in the previous section, the network needs both spatial and visual features to solve the edge recognition task. Given just the visual features, the network lacks information about the shape pair it should evaluate. With both features, the network can learn to figure out if there is an arrow whose tail is in proximity to the source shape, whose head is in proximity to the target shape, and whose bounding box equals the provided arrow bounding box. Both the spatial and visual modules rely on the bounding boxes of the edge candidate objects but do not take into account the predicted classes. The next section, therefore, presents the semantic module, which leverages the predicted classes.

Semantic module. The semantic module provides the network with an encoded representation of the predicted classes of an edge. The network can use these features to learn class-specific modeling rules and conventions, e.g., that control-flow edges connect activities, events, and gateways, whereas data associations involve at least one data element. Given the large number of shape classes, many combinations of source and target shape classes have only a few training samples, and we observed

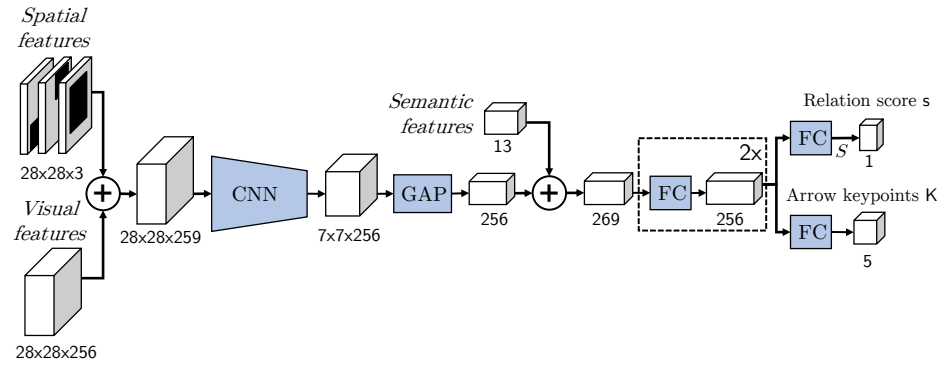


Figure 8.5: *Edge relation network*: The concatenated visual and spatial features are processed by a convolutional neural network (CNN). Next, the feature vector obtained from the global average pooling (GAP) layer is fused with the semantic features. After two fully-connected layers (FCs), a network branch with a sigmoid function (S) predicts the edge relation score s, and a second branch predicts the arrow keypoints K.

that this leads to overfitting during training. Therefore, instead of directly using the predicted shape classes, we map the 21 shape classes into five more general shape groups: *activity*, *event*, *gateway*, *collaboration*, and *data elements*. From a process modeling perspective, this is reasonable, as the majority of modeling rules are applied on the level of shape groups. For example, the rule that a data association always connects a data store or data object can be simplified to a rule on data element level.

The predicted arrow class and the mapped shape groups are converted into vectors using one-hot encodings, which results in a vector of size 3 for the arrow, and two vectors of size 5 for the shapes. Finally, the three vectors are concatenated into a semantic feature vector of size 13.

Edge relation network. Given the input stemming from the three aforementioned modules, we use a neural network to predict the edge relation score and arrow path, as illustrated in Figure 8.5. The architecture of our network largely follows our earlier work [84], which is in turn based on a network designed to predict relationships between form elements in a small document image dataset [19]. We first combine the visual and spatial features by concatenating the three binary masks as additional channels to the visual features and obtain a $28 \times 28 \times 259$ feature representation. The combined visual and spatial features are processed by a convolutional neural network (CNN). We use the same CNN architecture as in our earlier work [84], which consists of six depth-wise separable convolutional layers [38]. The spatial resolution of the features is downsampled twice by a factor of two, us-

ing strided convolutions in the third and last depth-wise convolution. This results in features of size 7×7×256. Next, we convert the three-dimensional features to a vector using global average pooling (GAP), which computes the mean over each of the 256 channels. We then concatenate the semantic vector, leading to an intermediate vector of size 269. We subsequently integrate these semantic features by applying two fully-connected layers (FCs), resulting in a final feature vector of size 256.

Given the final feature vector, the network predicts two outputs. First, a binary classification layer with a sigmoid function predicts the edge relation score s. A second linear layer predicts the arrow path as a sequence of arrow keypoints K, where each $(x, y) \in K$ represents a point within the image. Following our earlier work [84], we choose $|K| = 5$, and encode the arrow keypoint coordinates relative to the arrow region. The first and last keypoints in K indicate the arrow tail and head positions, and the three intermediate points capture the drawn path. The number of intermediate keypoints required to describe an arrow varies, e.g., an elbow arrow can be described with just one intermediate keypoint. Therefore, we remove superfluous intermediate keypoints in the last step of our approach, as we describe in Section 8.1.4.

Given the edge relation score s and the arrow keypoints K, we obtain a tuple $e = (a, src, tgt, l, K) \in E_C$ for each edge candidate. The next section details how we use the set of edge tuples E_C to determine the final set of edges.

Edge inference. The edge inference procedure determines the final set of edges $E \subset E_C$. For each arrow a, we choose the edge candidate e with the highest edge score $e.s$. In other words, we reduce the number of edge candidates to the number of arrows by only keeping the most likely edge per arrow. Next, we aggregate the edge score $e.s$ with the object detector score $e.a.s$ by taking the minimum of both, i.e., we set $e.s = \min(e.s, e.a.s)$.

Next, we remove all edges whose aggregated score is lower than a threshold τ_e (we use 0.5 as default and test further values in our evaluation experiments), which results in $E'_C \subset E_C$. Last, we identify edges with the same connection, i.e., edges $e_1, e_2 \in E'_C$ with $e_1.src = e_2.src$ and $e_1.tgt = e_2.tgt$. We resolve these duplicates by only keeping the edge with the higher aggregated score $e.s$. As a result, we obtain the final set of edges E.

8.1.3 Label Recognition

In this step, our approach aims to recognize the BPMN labels indicated by the previously detected textblocks T (Section 8.1.1). The input for this step consists of the sets of shapes S, edges E, and textblocks T, as well as the image features $feat$. The output of this step is a set of labels L, where each label $l = (t, tgt, s, txt) \in L$

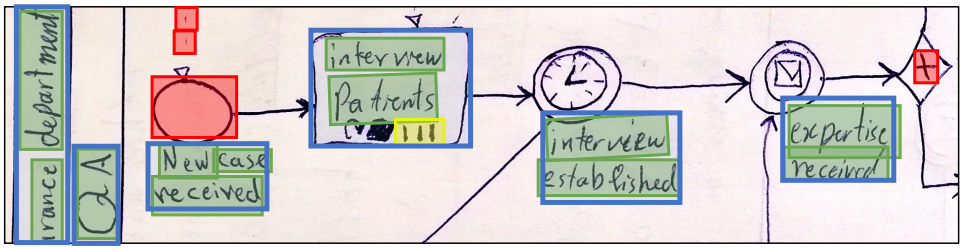


Figure 8.6: *Textblock handwriting recognition*: For each textblock (blue), the label is obtained by detecting the reading order of its contained words (green), except for notation words (yellow).

relates a textblock t, to a target shape or edge $tgt \in S \cup E$. Further, s represents the consolidated label score, and txt the textual content.

To achieve this, we decompose label recognition into two main parts. First, *textblock handwriting recognition* decodes the textual content within each textblock. Second, *textblock relation detection* strives to identify the shape or edge that each textblock labels, and, to eliminate duplicate textblocks that both relate to the same shape or edge. Besides the textblocks in T, our approach also decodes the textual content within the activities that are part of S. Therefore, we create a pseudo textblock for each activity. The pseudo textblock receives the same bounding box as its associated activity shape but does not participate in textblock relation detection since its target shape is already known. We refer to the set of regular and pseudo textblocks as T'.

Below, we introduce the two parts to obtain L in detail.

Textblock handwriting recognition. Given a textblock $t \in T'$, the textblock handwriting recognition procedure tries to decode the textual content txt within the textblock. We decompose textblock handwriting recognition into two stages. First, in *image word recognition*, we use an off-the-shelf OCR service to recognize all words in an image. Second, the *textual content decoding* procedure identifies the words that belong to each textblock and combines the words into a word sequence that represents the textual content of the textblock.

Image word recognition. In this stage, we try to identify all handwritten word objects $w = (b, d, txt) \in W$ within a given image. Each word w is defined by a bounding box b, the degree of its rotation angle d, and the word's textual content txt. To accomplish this, we leverage an off-the-shelf OCR service that supports handwritten text. Given a raw image as input, the OCR service returns a set of text lines, where each line consists of a sequence of words. As words can be arbitrarily rotated, the word bounding boxes also indicate the angle d. Since OCR services are commonly optimized towards handwritten documents, we observe that the returned

text lines often combine lines of multiple textblocks, e.g., when two textblocks are next to each other. In Figure 8.6, the service might recognize the text line "interview expertise", even though both words belong to different textblocks. Therefore, we discard the text line information and only keep the returned words W. Next, we describe our approach for identifying the word sequences that represent the textual content of each textblock.

Textual content decoding. Given the textblocks O'_T and words W, we propose a procedure that decodes the textual content of each textblock. To this end, we first identify the words $W^t \subset W$ that belong to each textblock t, and, then, combine the words W^t to obtain the textual content txt.

To identify the words that belong to a textblock, we first compute the intersection over area (IoA) between all words and textblocks in the image. The IoA quantifies to what fraction of a word w is contained in a textblock t, and is defined as the overlap area of $w.b$ and $t.b$ divided by the area of $w.b$. We match each word to the textblock with the highest IoA, while only keeping words whose textblock IoA exceeds 50%. We use this threshold to account for minor word and textblock localization errors. In Figure 8.6, the green boxes illustrate the words that have been assigned to their enclosing textblock. Further, the red boxes show that the OCR service returns many (false positive) words that actually are part of drawn shapes or edges. Among others, we frequently observe recognized "X" characters for exclusive gateway symbols, or multiple returned "-" or "I" characters for horizontal and vertical arrows. These false positives are discarded by only keeping words within textblocks. However, we also observe false positive words located within textblocks that often correspond to notational elements of BPMN, as illustrated by the yellow box in Figure 8.6. We try to eliminate these with a filter list of common notation words, which includes recognized words for parallel task (e.g., "III", "111") and collapsed sub-process markers (e.g., "+"). Applying both the matching and filtering procedure yields the final set of words W^t for each textblock t.

In the second stage, a reading order detection algorithm decodes the textual content txt of each textblock t. Given the words W^t, the algorithm uses the bounding box and rotation angle d of each word to identify which words form a line, and in which order the lines should be read. The algorithm details can be found in Appendix A.

Textblock relation detection. Similar to edge recognition (Section 8.1.2), we formulate textblock relation detection as a relationship detection problem and decompose it into three stages. First, *relation candidate generation* produces a set of shape and edge candidates for every textblock $t \in T$. Second, *relation scoring* predicts the score for each candidate pair, where a candidate pair consists of a textblock and a related shape or edge. Last, the *relation inference* procedure tries

to find the most likely shape or edge for each textblock and to eliminate duplicate labels, i.e., multiple textblocks that have been related to the same shape or edge.

Relation candidate generation. Given a textblock t, we first identify all shapes and edges that the textblock might label. In edge candidate generation (cf. Section 8.1.2), the set of candidates includes all shapes that overlap with the arrow region. Here, the arrow region corresponds to the arrow bounding box extended proportionally to its width and height. We follow a similar procedure to generate textblock relation candidates but also account for the following specifics of BPMN labels. First, we observe a large variance in textblock bounding box sizes, since the textual content can range from single-digit to multi-line text phrases. Therefore, we do not pad each textblock relative to its dimensions. Instead, we compute the median size of all textblocks in the image, which we refer to as M. Here, the size of a textblock is defined as the mean of its width and height. To obtain the textblock region, we then pad the side of each textblock by a factor $pad_t * M$. Based on our experimental results described in Section 8.2.2, we set $pad_t = 1$.

In order to reduce the number of false positive candidates to evaluate, we also leverage two patterns regarding the positions of textblocks relative to the shape or edge that they label. First, we observe that edge bounding boxes often do not closely capture the drawn path of the arrow, especially for diagonal or elbow arrows. For example, even though the drawn path of the sequence flow in the center of Figure 4.8 is far away from the "Not Okay" textblock, their two bounding boxes are in proximity. Therefore, we only consider an edge as a relation target if its drawn path, identified by its keypoints K, intersects the textblock region. Second, we observe that pool and lane labels are commonly located near the boundary of the detected shape, as illustrated in Figure 8.2. Therefore, we only consider a pool or lane as a relation target if its border intersects the (extended) textblock region. This way, we avoid creating a candidate for every textblock that actually labels a shape or edge within the pool.

Figure 8.7 shows an exemplary textblock in an image, along with the related candidate shapes (blue) and edges (orange) that intersect with the textblock region (red). As indicated, we do not consider activity shapes as relation targets, since we know the relation targets of the (pseudo) textblocks that we created for each activity. Further, we exclude data association arrows, as not a single such arrow is labeled in the `hdBPMN` dataset. In the next step, we use a network to evaluate each textblock relation candidate (t, tgt).

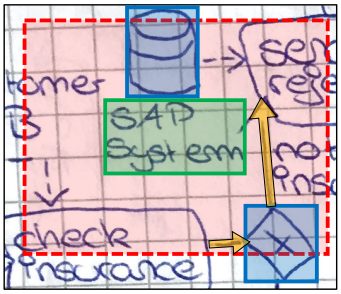


Figure 8.7: Relation candidate generation

Relation scoring. In this second stage, we use a textblock relation network to determine the score of a relation candidate (t, tgt), i.e., the likelihood that textblock t labels a shape or edge, tgt. As in edge recognition, the network consists of three modules and outputs a relation score. The main difference is that the textblock relation network operates on object pairs instead of triplets. As a consequence, the spatial module generates two instead of three binary masks. Further, since the source object of the textblock relation is always a textblock, the semantic module only encodes the predicted class of one object, namely the target shape or edge. Finally, the visual module extracts the features from the textblock region using RoIAlign, and thus works in the same way as with edge recognition. The network details can be found in Appendix A.

Relation inference. The relation inference procedure determines the final set of textblock relations, and as part of this, also eliminates textblocks that have been identified as duplicates. This procedure is, again, performed in a similar manner as its corresponding part in the edge recognition step. For each textblock t, we choose the relation candidate (t, tgt) with the highest relation score s. We then aggregate the relation and the object detector score by taking their minimum and remove all textblocks whose aggregated score is lower than a threshold τ_l (we use 0.5 as a default and test further values in our evaluation experiments). Last, we identify textblock duplicates, i.e., multiple textblocks that have been matched to the same shape or edge, and we resolve these cases by only keeping the textblock with the highest score.

8.1.4 Approach Output

The last step in our approach takes the final shapes S, edges E, and labels L to create a process model in the BPMN 2.0 XML format. The XML format consists of two main schemata: the actual process model and the BPMN DI schema, which defines the shape and label bounding boxes and the waypoints of edges. For each predicted shape, edge, and label, we create a respective element in the XML file. When creating a BPMN DI edge element for each $e \in E$, we follow the typical convention and define the first and last waypoint as the points that intersect with the edge's source ($e.src$) and target ($e.tgt$) shapes, respectively. To that end, we shift the first and last predicted keypoint of $e.K$ to the nearest point on the bounding box boundary of the connecting shapes, except for gateways, where we connect the keypoint to the closest of the four diamond corner points. For the intermediate keypoints, we remove superfluous points with the Douglas-Peucker line-simplification algorithm [23]. Concretely, we remove every point where the distance between the original and induced simplified path is less than 5% of the total arrow length, resulting in a smoother representation as the final output of `Sketch2Process`.

8.1.5 Training

To operationalize `Sketch2Process`, we need to train and validate all neural network components on a dataset with accompanying ground truth, which we split into a training and validation set. In this section, we describe the format of this ground truth, the overall training procedure, and details on our employed image augmentation pipeline.

Ground truth. Given an image containing a hand-drawn BPMN model, our approach uses a ground-truth annotation that captures the shapes, edges, and labels contained in it. The formats for shapes and labels are straightforward and follow directly from the annotation procedure. Specifically, each *ground-truth shape* is a tuple (c, b), with c as the shape's class and b its bounding box, whereas each *ground-truth label* is a tuple (txt, tgt, b), with txt as the label's text, tgt its target shape or edge, and b its bounding box.

The ground truth of edges is slightly more intricate. Particularly, each *ground-truth edge* is a tuple (src, tgt, K, b), where src and tgt are the source and target shapes that the edge connects, K a sequence of keypoints (x and y coordinates, corresponding to BPMN waypoints) that are used to capture the edge's drawn path, and b the edge's bounding box. The first and last keypoints of K, respectively, correspond to the tail and head of the edge, whereas additional keypoints may be used to indicate an edge's bending points. Here, it is important to note that the tail and head keypoints should correspond to the points where the edge intersects or is supposed to intersect with its source and target shapes, rather than to the actually drawn start and endpoints of the edge. In this manner, we account for incompletely drawn edges that do not connect with their corresponding shapes (see issue e2 in Figure 4.8). Then, the object detector trained on these annotations will strive to predict where an arrow bounding box *should* have ended (or started) if the edge had been drawn properly, rather than predicting the point where the edge ends (or starts) in the drawing (see also Section 8.1.1). Finally, an edge's bounding box b is automatically computed as the smallest box that contains all keypoints in K.

Training procedure. In the context of deep learning, which encompasses the neural networks we employ, the goal of model training is to find network parameters (i.e., weights) that maximize the performance on the validation set. We follow the de facto standard in deep learning and train the networks with stochastic gradient descent (SGD) [29]. SGD is an iterative method, where, at each step, k examples (images in our case) are randomly sampled from the training set. Here, k is referred to as the batch size, and the set of sampled examples is called a mini-batch. During each SGD iteration, a *loss function* quantifies the difference between the ground truth and the predicted outputs of the neural networks. We apply the loss function per image and obtain a mini-batch loss by averaging over the k image

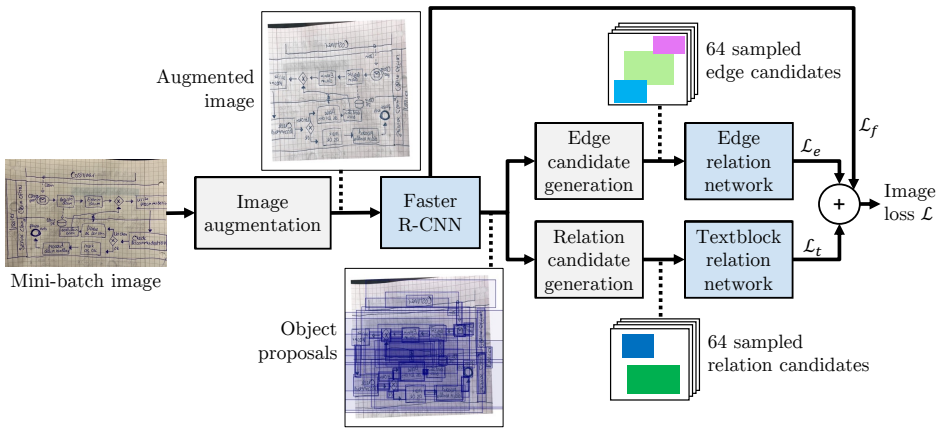


Figure 8.8: *Training overview*: Given the augmented image, Faster R-CNN produces a set of object proposals. Edge candidate generation identifies all proposals that sufficiently overlap with a ground-truth shape or edge. It then generates 64 edge candidates, for which the edge loss $\mathcal{L}_e$ is computed by comparing the relation network outputs to the respective ground truths. The textblock relation loss $\mathcal{L}_t$ is computed on 64 relation candidates using a similar procedure. Finally, the image loss $\mathcal{L}$ combines the losses of all networks.

losses. Although we defer the technical details to Appendix A, the intuition of how we compute an image loss is as follows.

As depicted in Figure 8.8, we first apply an *image augmentation* pipeline (described below) and obtain a randomly augmented version of the input image. As mentioned in Section 2.2.2, Faster R-CNN uses the (augmented) image to produce the image features $feat$. Given these features, it generates a set of object proposals in the first stage. In the second stage, a box-head network uses the proposals and image features to predict the detected objects. Following the standard training procedure of this object detector [77], we compute the aggregated Faster R-CNN loss $\mathcal{L}_f$ by comparing both the proposals and the detected objects against the ground-truth objects. Concretely, this means that Faster R-CNN uses a ground truth consisting of all shapes, as well as the bounding boxes and classes of edges and labels (representing arrow and textblock objects, as described in Section 8.1.1).

As Figure 8.8 illustrates, we use the object proposals to train our edge and textblock relation networks, which follows related work that extends Faster R-CNN [35]. To this end, we randomly sample 64 edge candidates (for edge recognition) and 64 relation candidates (for label recognition) from the object proposals. We then compute an edge loss $\mathcal{L}_e$, which consists of a relation loss and a keypoint loss. For the *relation loss*, we use the ground-truth information on both shapes and edges to assess if an edge candidate is considered true positive or not, which

results in a binary ground-truth relation score. We then compare the predicted and the ground-truth relation score using binary cross entropy. For the *keypoint loss*, we compute the mean squared error between the predicted and ground-truth keypoints K of an edge (though the latter are resampled in an equidistant manner). The label relation loss $\mathcal{L}_l$ is calculated in the same manner as for edge relations, except that we are using the entire ground truth, as textblocks can be used to label both shapes and edges. Finally, we compute the image loss, $\mathcal{L} = \mathcal{L}_f + \mathcal{L}_e + \mathcal{L}_l$, as a weighted combination of the losses of the three networks.

Image augmentation pipeline. During training, we have to account for the particularities of the `hdBPMN` dataset in terms of its size (hundreds rather than many thousands of images) and diversity with respect to the means used to create and digitize the hand-drawn models, such as the type of paper and drawing implement (see also Section 4.3).

Therefore, given a mini-batch image, we apply a randomly selected set of augmentation methods that are specifically suited to the data at hand. A code listing capturing the exact pipeline we use for this, based on the Albumentations library [12], is provided in the supplementary material, whereas we here focus on a description of the augmentation methods and their relevance.

In particular, we employ augmentation methods that randomly rotate, flip, and scale training images, as well as add Gaussian noise. These methods have been shown to work well for smaller datasets used in flowchart recognition [82]. On top of those, we also employ methods that are particularly aimed at the camera-based images in our dataset, which we introduced in a previous work [85]. This involves altering varying properties to reflect different kinds of camera-based images, for which we randomly change the brightness and contrast of the image, and shift the hue, saturation, and value color scale.

Finally, to improve generalization, we further introduce a new crop augmentation in this work, which cuts out a random part of the diagram and then enlarges the crop to the scale of the original diagram. Concretely, we crop an image patch whose scale compared to the original image is between 0.2 and 1.0, and whose ratio is between 0.5 and 1.5. Then, we keep all objects from the ground truth where at least 70% of the bounding box is located within the cropped image. To avoid having empty (cropped) images, we reapply the entire augmentation pipeline until the augmented image contains at least 3 objects. Overall, we observe that random cropping improves our recognition results, as during training the object detector sees variations of the same image that have been (1) stretched in horizontal or vertical direction and (2) where only a subset of the diagram symbols are visible.

Figure 8.9 shows some augmented images obtained by applying our proposed pipeline on the running example.

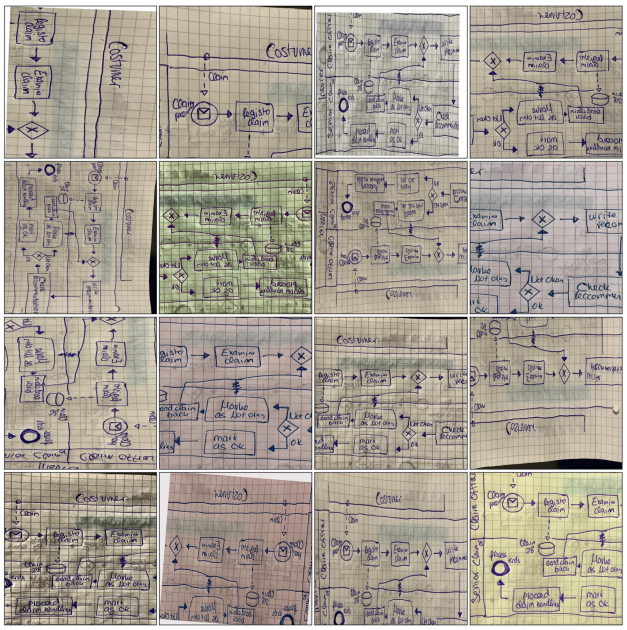


Figure 8.9: *Image Augmentation Example*: Our augmentation pipeline produces different variations of the same input image

8.2 Evaluation

To demonstrate the capability of our approach, we trained and optimized it using the training and validation set of `hdBPMN`, and conducted an evaluation using its test set. The evaluation results clearly demonstrate that our approach can reliably recognize hand-drawn BPMN models from images and, hence, remove undesirable friction in the modeling workflow.

8.2.1 Evaluation Setup

Below we elaborate on the details of our employed implementation and parameter settings, as well as the metrics and baselines used to evaluate our approach.

Implementation. Our neural network implementation is based on the Detectron2 [104] framework, which provides an extensible Faster R-CNN implementation based on PyTorch [69]. For OCR, we use the Microsoft Azure OCR service,[2] version v3.2. The service is asynchronous, i.e., we submit the image via a pro-

[2] `https://docs.microsoft.com/azure/cognitive-services/computer-vision/overview-ocr`

cessing request, and then periodically check if the results are available. In order to optimize the overall runtime of our approach, we perform network inference while waiting for the OCR results. Readers are invited to try out their own sketches using our public demo:
`http://sketch2bpmn.informatik.uni-mannheim.de/`.

Parameter settings. `Sketch2Process` has several parameters, for which we analyze the effect of different choices. During inference, we use three score thresholds, one for shapes (τ_s), one for edges (τ_e), and one for labels (τ_l), to decide which elements to keep. For each score threshold, we explored choices in the range from 0.05 to 0.95. In addition, both our edge and label recognition components have a region size parameter, which we consider the most important parameter of the respective components, as they are used to identify elements in proximity during candidate generation, and to extract the visual features in the scoring procedures. For the arrow region size, we explored arrow padding percentages pad_a in the range from 0% to 50%, with a step size of 10%, and also evaluated 5% when we found that 10% performs best. For the textblock region size, we explored textblock padding factors pad_t in the range from 0.0 to 2.0, with a step size of 0.25. Finally, the *textual content decoding* step, which is part of the label recognition component, has two additional parameters. For matching words to textblocks, we explore IoA thresholds τ_{ioa} in the range from 0.001 to 1.0. For determining which words belong to the same line as part of our reading order detection algorithm, we explore line distance thresholds τ_{ld} in the range from 0.0 to 1.0.

Besides the mentioned parameters, Detectron2 has several training configurations, for which we apply the settings from previous works [82, 84]. Faster R-CNN can be equipped with different backbone networks. We use the ResNet-50-FPN [57] backbone throughout the experiments, which is relatively fast and provides a good speed-accuracy trade-off. For training, we largely follow the default Detectron2 configuration for training Faster R-CNN with a ResNet-50-FPN backbone. We use stochastic gradient descent with a batch size of 4, and a learning rate of 0.005. We train the model for 90k iterations, which means that the model sees 360k augmented images (90k batches of size 4). This takes about 32 hours on a Tesla V100 GPU with 16GB memory. During training, we multiply the learning rate after 50k and 70k iterations by a factor of 0.4. Here, we follow our previous work [84], where we found that the default factor of 0.1 decreases the learning rate too much. We initialize the model weights with models pretrained on the COCO dataset, which we obtain from the Detectron2 model zoo. Although the COCO dataset consists of images from a different domain than our target one, i.e., images of everyday scenes, we observed minor improvements with this transfer learning strategy in our previous works.

Finally, another important part of the training procedure is the image augmentation pipeline. To understand its impact on the overall results, we conducted an ablation study in which we compare a model trained using our full augmentation pipeline to two additional models: one with no augmentation at all, and one with just the default augmentations from the Detectron2 library (random resizing and horizontal flipping).

Metrics. To evaluate our approach, we compare a process model extracted by our approach to the one in the manually annotated image (see Section 4.2), i.e., the *ground truth*.

Object detection metrics. To quantify the *object detection* performance, we follow related work in diagram recognition [9, 11, 103]. A detected shape, edge, or textblock is considered a true positive if it has the correct class and its bounding box overlaps sufficiently with its ground-truth counterpart. Particularly, following other work [103], we consider this overlap sufficient if the bounding boxes have an overlap that exceeds an IoU threshold of 50%, which accounts for annotation inaccuracies in the ground-truth bounding boxes. To quantify object detection performance, we then use this notion of true positives to match the ground truth to the predicted objects and compute the standard *precision*, *recall*, and F_1 scores. For infrequent shape classes, there can be zero predicted objects, in which case the recall is zero. As this leads to a division by zero in the calculation of the precision score, we handle this edge case by reporting n/a for both precision and F_1 scores. Similarly, we report a recall of n/a for classes that appear in the training, but not in the test set.

As the object detection step completes the recognition of shapes, we compute the macro shape F_1 score as the mean F_1 over all shape classes. Similarly, the micro shape F_1 score is computed as the F_1 over all shape predictions. This means that the micro score weights each class according to its relative frequency, and thus the score is biased towards more frequent classes.

Edge recognition metrics. To quantify the performance for edge recognition, we follow related work by determining if our approach is able to relate detected arrow objects to the correct (i.e., ground-truth) source and target shapes [9, 11]. We, again, compute precision, recall, and F_1 scores for this, where we consider an edge to be a true positive if both the source and target shapes are correctly identified.

Note that this means that edge recognition is affected by object detection accuracy for arrows, as well as for shapes: if an arrow object was not correctly detected, we cannot relate it to any shapes, whereas if a shape was not correctly detected, we cannot relate any edges to it, either.[3]

[3]Note that an edge is still considered correct if its associated shapes are incorrectly classified.

Label recognition metrics. We consider label recognition performance with respect to its two parts: textblock relation detection, for which we assess how well our approach relates textblocks to their corresponding shapes and edges, and textblock handwriting recognition, for which we evaluate the actual textual content detected in these textblocks.

For *textblock relation detection*, we follow the same evaluation procedure as for edge recognition, i.e., a label is a true positive if its textblock has been correctly detected and the associated shape or edge has been correctly identified. Note that, our conceptual contributions focus on this part of label recognition, whereas the rest primarily depends on an employed OCR service. Therefore, the overall label recognition performance that we report for our approach only includes these non-textual aspects.

For *textblock handwriting recognition*, we assess its two steps. We evaluate the *image word recognition* step by comparing the OCR performance to the ground-truth words annotated in an image. We consider a predicted word a true positive if its IoU score exceeds 50%, allowing us to report on precision, recall, and F_1 scores for the OCR service. In addition, we compare the predicted and ground-truth texts of the true positive words using the character error rate (CER), a common metric to evaluate HWR methods [80]. The CER is defined as the Levenshtein distance at character level between the prediction and ground truth, normalized by the ground-truth length. Prior to computing the CER, we lowercase the texts and replace line breaks with whitespaces, as we observe inconsistent annotations for both. Finally, for the *textual content decoding* step, we assess how accurately our approach assigns words to the textblocks (and thus to labels). We evaluate this based on the ground-truth words, reflecting the accuracy of the decoding step in isolation, and based on the actually detected words, reflecting the end-to-end performance of the textblock handwriting recognition procedure.

Baselines. To demonstrate the efficacy of our approach, we compare its performance to related works. To this end, we train and evaluate the following systems on the `hdBPMN` dataset: BPMN-Redrawer [1], Arrow R-CNN [82], Sketch2BPMN [85], and DiagramNet [84] (see also Chapter 3). For the BPMN-Redrawer comparison, we use the training configurations provided in the open-source implementation. For Arrow R-CNN, we use its default image augmentation methods, which are targeted at diagrams drawn on a white background. We also compare to the Sketch2BPMN approach, which extended Arrow R-CNN with additional augmentation methods and an improved rule-based edge relation procedure. Last, we compare to the DiagramNet [84] approach. DiagramNet does not predict arrow bounding boxes. Instead, the arrow bounding box is generated as the smallest bounding box that contains all predicted arrow keypoints. In the paper, the edge recognition

Table 8.1: Overall approach results

	Shape		Edge		Label
Approach	**Micro F_1**	**Macro F_1**	**Micro F_1**	**Macro F_1**	**F_1**
BPMN-Redrawer [1]	89.2	73.7	63.1	47.5	—
Arrow R-CNN [82]	94.4	86.5	87.7	82.9	—
Sketch2BPMN [85]	94.8	87.3	90.7	86.2	—
DiagramNet [84]	95.6	**88.5**	85.5	78.8	—
Sketch2Process	**95.8**	88.3	**93.0**	**90.5**	**92.6**

evaluation metric considers neither arrow bounding boxes nor keypoints, and instead only considers if an arrow has been matched to the correct source and target shape. Our hypothesis is that, while DiagramNet is comparable to our approach in edge relation detection performance, it produces less accurate keypoints and bounding boxes. To verify this hypothesis, we compare our approach to DiagramNet using both evaluation procedures, the one proposed in our work, and the one used to evaluate DiagramNet originally [84].

8.2.2 Results

This section presents the results of our evaluation for the `hdBPMN` test set, first in terms of overall results, before taking a detailed look at the results per BPMN element class. In addition, we present a sensitivity analysis of the major parameters and the measured runtime of `Sketch2Process` and its major components.

Overall results and baselines. The overall results are presented in Table 8.1. As the cells with missing label F_1 scores indicate, our approach is the first work that addresses the recognition of all three diagram components. The results reveal that, for shape recognition, our approach is on par with DiagramNet, and outperforms all other approaches, especially on macro F_1. Note that micro and macro measures differ because certain classes (e.g., *Tasks*) are much more common and easier to recognize than others (e.g., specific kinds of events). However, the overall trends are consistent across the two. For edge recognition, our approach considerably outperforms all other approaches, achieving both a micro and a macro F_1 score above 90. The rather low performance of the BPMN-Redrawer approach highlights that, especially for the recognition of hand-drawn edges, a dedicated network architecture is required.

As mentioned above, we also compared our edge recognition performance against DiagramNet [84] using the true positive definition from that work. Here, we observe that our approach has the highest macro F_1 (85.4), followed by Dia-

gramNet (83.8) and Sketch2BPMN (80.8). These results indicate that, while DiagramNet underperforms all other approaches in edge recognition with our bounding box-based evaluation, it performs much better when only considering whether an edge actually exists between two shapes.

Regarding label recognition, we observe that our approach achieves an F_1 score of 92.6 (as indicated in Section 8.2.1, this corresponds to performance on textblock relation detection). Given that, to our knowledge, we are the first approach that addresses label recognition, we do not have prior work to compare to.

Object detection. Table 8.2 provides detailed insights into the performance of our object detector, by depicting the results obtained per shape, arrow, and text class. The table shows that our approach correctly recognizes the vast majority of objects, achieving an F_1 score of at least 90 for all arrow, text, and 14 out of 21 shape classes. For some shape types, the number of data points is too low (in both the training and the test set) to sufficiently cover the spectrum of factors such as drawing styles and, therefore, too low to provide reliable evaluation results.

A post hoc analysis of the results reveals that the most difficult task for our object detector is the correct classification of certain kinds of events. This comes as no surprise, though, since the difference between some of the eight kinds of events may only be due to marginal differences, such as a change in line thickness (start events), as well as different kinds of tiny envelopes (message events) and clocks (timer events). Especially in light of the diverse shapes in our dataset, as highlighted in Figure 4.6, identifying such differences in hand-drawn models can already be highly complex for humans, let alone for an automated approach that lacks sufficient training examples for some of the rarer classes.

Edge recognition. The edge recognition results in Table 8.3 again demonstrate the overall strong performance of our approach, as well as that *sequence flows* (F_1 of 94.7) are easier to recognize than message flows (88.1) and data associations (88.8). To some extent, this can be attributed to the commonality of sequence flows and the fact that the latter two classes use dashed rather than continuous lines. However, it is also highly interesting to consider the different roles of these edges from a process modeling perspective. Particularly, message flows connect (elements in) different pools, which are often placed relatively far from each other. This results in longer edges, which may also cross more nodes, and are, therefore, harder to analyze for an automated approach. For example, we observe that the median arrow path length is more than twice as high for message flows (447 pixels) as for sequence flows (152). For data associations, it is important to consider that elements related to the data perspective are often drawn last [24, p.177], whereas they also often are connected to multiple shapes, scattered throughout a model. These two factors thus commonly result in data associations that cross other edges

Table 8.2: Object detection results per class obtained for the test set

Group	Class	Prec.	Rec.	F_1	Count
Activity	Task	97.8	99.6	98.7	763
	Subprocess (collapsed)	96.2	80.6	87.7	31
	Subprocess (expanded)	n/a	0.0	n/a	3
	Call Activity	100.0	100.0	100.0	1
Event	Start Event	94.4	97.1	95.8	70
	Intermediate Event	n/a	n/a	n/a	0
	End Event	97.4	97.4	97.4	190
	Message Start Event	93.9	86.8	90.2	106
	Message Interm. Catch Event	84.0	94.3	88.9	106
	Message Interm. Throw Event	79.6	70.9	75.0	55
	Message End Event	81.1	76.9	78.9	39
	Timer Start Event	100.0	93.3	96.6	15
	Timer Intermediate Event	93.1	94.7	93.9	57
Gateway	Exclusive Gateway	97.6	98.0	97.8	247
	Parallel Gateway	96.1	96.8	96.4	126
	Inclusive Gateway	n/a	0.0	n/a	1
	Event-based Gateway	91.9	94.4	93.2	36
Collaboration	Pool	96.6	99.0	97.8	203
	Lane	91.9	94.6	93.2	111
Data Elements	Data Object	98.9	97.3	98.1	185
	Data Store	100.0	94.3	97.1	35
Arrow	Sequence Flow	97.6	95.6	96.6	1,887
	Message Flow	94.4	89.3	91.8	346
	Data Association	96.7	86.9	91.5	367
Text	Textblock	96.8	94.0	95.4	1,538
Overall	Macro avg.	94.3	84.8	89.3	6,518
	Micro avg.	95.9	95.1	95.5	6,518

Table 8.3: Edge recognition results per class obtained for the test set

Class	Precision	Recall	F_1	Count
Sequence Flow	95.9	93.5	94.7	1,887
Message Flow	91.0	85.3	88.1	346
Data Association	93.9	84.2	88.8	367
Macro avg.	93.6	87.7	90.5	2,600
Micro avg.	95.0	91.1	93.0	2,600

Table 8.4: Textblock handwriting recognition results obtained for the test set

Recognition task	Mean CER
Image word recognition	5.5
Textual content decoding (GT Words)	0.5
Textual content decoding (OCR Words)	8.8

or even shapes, which complicates their recognition.

By comparing the arrow object detection results in Table 8.2 with the edge recognition results in Table 8.3, we can quantify the effectiveness of our edge relation detection procedure. Concretely, the difference between the object detection and the edge recognition measures are arrows that were correctly detected, but where either the source or target shape was not correctly identified. Here, we observe that the F_1 delta is lowest for sequence flows (-1.9), and increases slightly for data associations (-2.7) and message flows (-3.7). For comparison, we also computed these differences for our rule-based approach in Sketch2BPMN [85]. Here, we find that the F_1 delta is much higher for sequence flows (-3.5) and message flows (-10.4), and slightly lower for data associations (-2.5). Overall, this shows the effectiveness of our improved edge recognition method, especially for complex arrows such as message flows.

Label recognition. In the following, we report the evaluation results of both parts of our label recognition procedure.

Textblock relation detection. Table 8.5 shows how well our approach is able to relate textblocks to corresponding shapes and edges, for each of the element groups. The table shows that event labels (F_1 of 96.6) and data element labels (F_1 of 96.3) are easier to detect and relate than collaboration labels (F_1 of 90.7) and edge labels (F_1 of 87.6). Since we do not (need to) detect activity labels through dedicated textblocks, we do not report results for activities.

Table 8.5: Textblock relation detection results per group obtained for the test set

	Predicted objects			Ground-truth objects			
Group	**Prec.**	**Rec.**	**F_1**	**Prec.**	**Rec.**	**F_1**	**Count**
Event	97.2	96.1	96.6	99.2	98.7	99.0	536
Gateway	n/a	0.0	n/a	n/a	0.0	n/a	3
Data element	98.1	94.5	96.3	100.0	98.6	99.3	220
Collaboration	93.8	87.8	90.7	100.0	98.7	99.4	312
Edge	92.4	83.3	87.6	97.5	92.7	95.1	467
Macro avg.	95.4	90.4	92.8	99.2	97.2	98.2	1,538
Micro avg.	95.3	90.1	92.6	99.0	96.7	97.8	1,538

To assess the accuracy of the relation detector independent of object detection errors, we also evaluate it against the ground-truth objects. Here, the table shows near-perfect results for four out of five category groups, which demonstrates the effectiveness of our textblock relation detector. Overall, the recognition of edge labels is the most difficult task for our approach, which can be attributed to the fact that there are often multiple arrows in proximity to a textblock. For collaboration shapes (pools and lanes), the near-perfect results when using the ground-truth objects indicate that the errors are largely due to textblock and collaboration object detection errors. Here, a post-hoc analysis shows that the most difficult task is the detection of nested lanes and their labels, where the bounding boxes of parent and child lane are very similar (IoU of up to 97%).

Textblock handwriting recognition. Turning to the textual content within the detected textblocks, we assess both steps of the handwriting recognition part:

For the *image word recognition* step, the OCR service achieves a precision of 69.6% and a recall of 90.0% when it comes to detecting words in images (irrespective of their textual content). The low precision is largely due to single-character, false positive "words" that actually correspond to parts of drawn shapes or edges (e.g., "X" characters stemming from exclusive gateway symbols and dashes from dashed arrows), as mentioned in Section 8.1.3. Such false positives are not an issue for our approach, though, since they are eliminated during the textual content decoding step, given that they are not located within a textblock. For the true positive words identified by the OCR service, we observe a mean CER of 5.5%, as shown in Table 8.4.

For the *textual content decoding* step, the mean CER of 0.5% obtained when using the ground-truth words (GT words) shows that this step is very accurate in isolation. This means that if the OCR service would be perfect, the labels of the

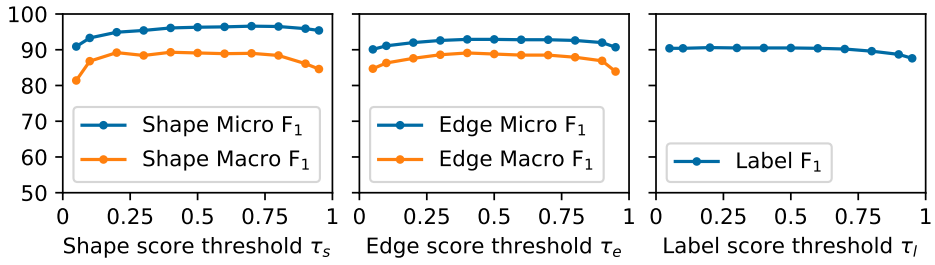


Figure 8.10: Score threshold analyses conducted on the validation set

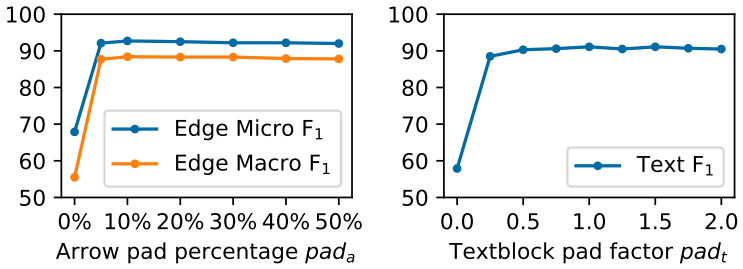


Figure 8.11: Region size analyses conducted on the validation set

detected textblocks would be near-perfect as well. In terms of end-to-end performance of textblock handwriting recognition, we observe a CER of 8.8% when using the actually detected OCR words as input for textual content decoding. On top of character recognition errors made by the OCR service in true positive words (captured in the 5.5% CER for image word recognition), the 8.8% CER is also caused by words that the OCR service failed to detect, i.e., false negatives.

In summary, the vast majority of errors that happen during textblock handwriting recognition directly or indirectly stem from errors of the OCR service. It is, furthermore, worth highlighting that this does not imply that major corrections to the generated models are required. For instance, in the running example (cf. Figure 4.3), only 18 of the 31 shapes and edges actually have labels, and among those, the median edit distance is 1, which means that only minor edits are required to fix the labels in the recognized model.

Sensitivity analysis. As mentioned, we analyze the sensitivity of the main parameters of `Sketch2Process`, including the settings for the different score thresholds, the arrow and textblock region sizes, the textual content decoding, and the image augmentation methods.

Score thresholds. Figure 8.10 shows the results for the score thresholds τ_s, τ_e,

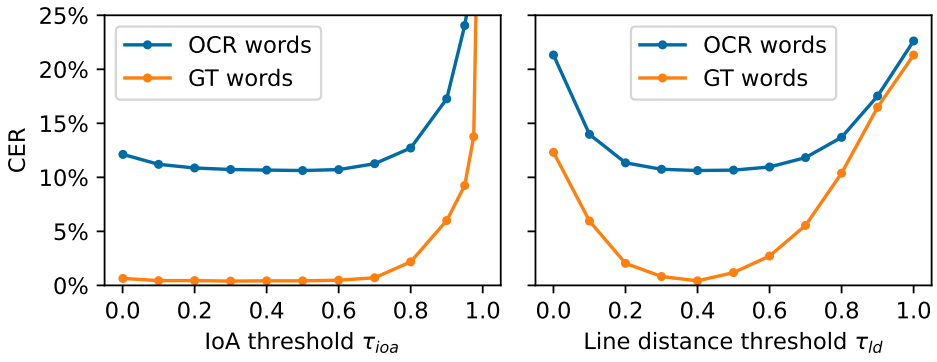


Figure 8.12: Textual content decoding analyses conducted on the valid. set

and τ_l, of which we apply each in one component. We find that each threshold obtains similar results in the range from 0.4 to 0.7, where the difference between the highest and lowest F_1 score is at most 0.6. The performance only degrades substantially when using a very small or a very large threshold. This shows that our approach is very robust to minor score threshold changes. Given the analysis, we decided to simply set all three thresholds to 0.5.

Region sizes. Figure 8.11 shows the results for the region size configurations. For the arrow pad percentage pad_a, we observe that the edge F_1 scores are substantially lower when applying only the minimum padding, are almost identical in the range from 10% to 30%, and then slightly drop at 40% and 50%. As the number of edge candidates increases with the arrow region size, which leads to increased inference time, we use $pad_a = 10\%$ in our approach. Regarding the textblock region size, we observe that the results for the pad factor pad_t are very similar in the range from 0.75 to 2.0, with an F_1 between 90.5 and 91.1. As expected, the F_1 drops substantially when using no padding ($pad_t = 0$), as the majority of textblocks (53% in the training set) do not intersect with the shape or edge that they label. Similar to the arrow region size, the number of relation candidates increases with the textblock region size. Therefore, we employ $pad_t = 1.0$ for our approach.

Textual content decoding. Figure 8.12 shows the results for the IoA and line distance threshold configurations. For the IoA threshold τ_{ioa} that we apply when assigning the OCR words to textblocks, we observe that the CER is lowest for $\tau_{ioa} = 0.5$, which is why we use this as a default value in our approach. Overall, we observe very similar results for τ_{ioa} in the range from 0.3 to 0.6, though. As expected, the CER increases dramatically when τ_{ioa} approaches 1.0. In the extreme case of $\tau_{ioa} = 1.0$, only words are matched whose bounding box is fully contained within the corresponding textblock. When matching the ground-truth words to the

Table 8.6: Image augmentation ablation study conducted on the validation set

Augmentation	Shape		Edge		Label
	Mi. F_1	Ma. F_1	Mi. F_1	Ma. F_1	F_1
No augmentation	92.2	81.8	86.1	78.2	86.0
Detectron2 (resize & hor. flip)	94.2	85.3	89.3	83.1	88.5
Sketch2Process (ours)	**95.7**	**87.9**	**92.8**	**88.0**	**90.6**

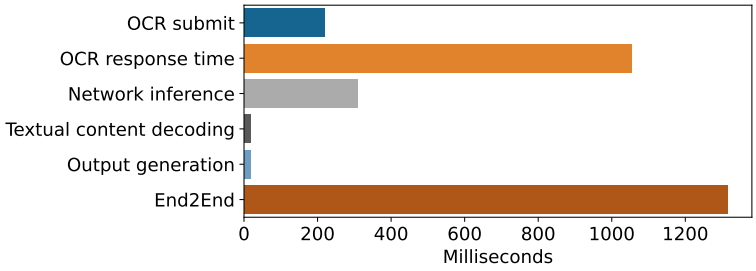


Figure 8.13: Median runtime measures obtained for the validation set

textblocks, the CER is below 0.5% for τ_{ioa} in the range from 0.1 to 0.6, which demonstrates the effectiveness of our simple IoA-based procedure.

For the line distance threshold τ_{ld} that we apply in our word reading order detection algorithm, we observe the lowest CER for both the OCR and ground-truth word evaluation at $\tau_{ld} = 0.4$, which is why we employ this configuration in our approach. In addition, the results are similar for τ_{ld} in the range from 0.3 to 0.5. Outside of that range, the CER increases the further τ_{ld} deviates from 0.4, as the words tend to be grouped in either too few lines ($\tau_{ld} < 0.3$) or too many lines ($\tau_{ld} > 0.5$).

Image augmentation. The results of our ablation study in Table 8.6 show the benefits of using image augmentation. Compared to using no augmentation, the performance of our approach consistently improves when using the default object detection augmentations provided by the Detectron2 library (random resizing and horizontal flipping), leading to F_1-score increases between 2.0 and 4.9. A further performance improvement of the same magnitude can be observed with the full `Sketch2Process` image augmentation pipeline, leading to further F_1-score improvements between 1.5 and 4.9. Notably, the recognition of the challenging edge categories (message flow and data association) benefits most from augmentation, which leads to the large edge macro F_1-score improvements (from 78.2 to 88.0).

Runtime. Finally, Figure 8.13 shows the median runtime of `Sketch2Process`

for both the individual components and the system end-to-end. Given an image to be processed, we first send the image to the OCR service (220 ms). We find that the time until the OCR results are available (1,125 ms) always exceeds the network inference time. Therefore, while waiting for the OCR results, we compute the results of all networks, i.e., the object detection, edge relation detection, and textblock relation detection networks, which on average takes 308 ms on a Tesla V100 GPU. After network inference, we periodically check for the OCR results, which on average takes 695 ms. Once the OCR results are available, we run the textual content decoding procedure (17 ms) to identify the textual content of each textblock and create the final BPMN XML as described in Section 8.1.4 (17 ms). On average, our approach processes an image in 1.3 seconds, most of which is spent waiting for the OCR results.

8.3 Conclusion

In this chapter, we presented `Sketch2Process`, a comprehensive approach for the recognition of hand-drawn BPMN models. Sketch2Process takes an image of a hand-drawn BPMN model as input and automatically transforms it into a format compatible with existing process modeling tools. Our approach is the first method that addresses all three subtasks of diagram label recognition: textblock detection, textblock handwriting recognition, and textblock relation detection. For its training and evaluation, we extended the `hdBPMN` dataset, which now consists of 704 hand-drawn and manually annotated BPMN models. Experiments conducted on this dataset demonstrated that `Sketch2Process` consistently outperforms all shape and edge recognition methods proposed in previous chapters.

Naturally, our work is subject to limitations. For example, our label recognition component relies on an off-the-shelf OCR service, which can lead to errors when dealing with handwriting (as seen in Section 6.2.2). However, this is not a critical issue for the conceptual validity or practical usefulness of our approach. Specifically, our work is independent of a particular service, which means that the employed OCR service can be replaced with improved versions in the future. Furthermore, the mistakes from this component can be quickly fixed manually or through a post-processing step that, for instance, builds on a dictionary. We provide additional conclusions in the next chapter, where we summarize the main results and highlight directions for future research.

Chapter 9

Conclusion

This chapter concludes the doctoral thesis. Section 9.1 summarizes the main findings and Section 9.2 gives an outlook on directions for future research.

9.1 Summary of Results

This thesis addressed the automated recognition of hand-drawn diagrams contained in images. In particular, we focused on the recognition of business process diagrams drawn on paper. To this end, we collected a dataset of hand-drawn BPMN models, and we developed several methods that we evaluated on this dataset. To show the general applicability of our methods, we also evaluated them on existing diagram datasets, which depict either flowcharts or finite automata diagrams. We can summarize the main results of this thesis as follows:

1. *Collection and publication of a dataset that consists of diagrams sketched on paper*: In Chapter 3, we surveyed existing work in diagram recognition. One finding in this chapter is that prior to our work, there were no existing offline datasets of hand-drawn diagrams. Therefore, the few existing offline works evaluated their methods on images generated from online datasets, which are much easier to recognize. In Chapter 4, we addressed this gap by presenting the `hdBPMN` dataset, which consists of 702 business process diagrams drawn on paper that we collected from 107 participants. In order to facilitate research in offline hand-drawn diagram recognition, we have made the images and BPMN annotations publicly available.[1] In addition, we have published a Python library to convert the images and annotations into a common computer vision format.[2]

[1] `https://github.com/dwslab/hdBPMN`
[2] `https://github.com/dwslab/pybpmn-parser`

Finally, we have open-sourced a tool that we developed to manually annotate images of hand-drawn BPMN diagrams.[3]

2. *Overview of challenges in recognizing camera-captured diagrams sketched on paper*: The diagrams in all existing datasets were collected by providing participants with computer-generated diagram templates and asking them to draw these diagrams on a tablet. As illustrated in Chapter 3, this procedure results in well-organized drawings, with black strokes on a white background. In Chapter 4, we identified the key challenges in recognizing diagrams obtained from conceptual pen-on-paper modeling sessions, as found in the `hdBPMN` dataset. We found that solving a conceptual modeling task on paper not only leads to diagrams with chaotic layouts, but also results in a wide variety of paper types, pens, and image-capturing methods. Overall, we outlined a large number of recognition challenges in Chapter 4, most of which are not present in existing datasets. For shapes, these challenges include incompletely drawn shapes, different shape types with a similar graphical notation, and multiple shapes with a significant bounding box overlap. For edges, among others, we observed long-range arrows with frequent direction changes, arrows that cross each other, and arrows drawn with an incorrect notation (e.g., using a dashed instead of a solid line). Finally, for labels, the challenges include illegible handwriting, nearby textblocks that are difficult to distinguish, and textblocks with multiple shapes or edges in proximity.

3. *Development of an end-to-end system for recognizing business process diagrams*: In Chapter 8, we introduced Sketch2Process, the first end-to-end system for recognizing sketched BPMN diagrams. Sketch2Process consists of the following steps. Given an image of a hand-drawn BPMN model, we first detect shape, arrow, and textblock object instances (object detection). Then, we identify the drawn path and the shapes that each arrow connects (edge recognition). Next, we decode the textual content within each textblock and identify the shape or edge that each textblock labels (label recognition). Finally, we generate a BPMN XML file suitable for process modeling tools. The system is implemented by extending an object detector with neural network components for edge and label recognition. The evaluation showed that Sketch2Process outperforms existing methods not only on hand-drawn but also on computer-generated BPMN datasets. We have created a public demo of Sketch2Process,[4] which has been showcased to several customers of *SAP Signavio*, a major vendor of business process modeling software.

[3] `https://github.com/dwslab/bpmn-image-annotator`
[4] `https://sketch2bpmn.informatik.uni-mannheim.de/`

4. *Recognition of diagram edges using keypoint and visual relationship detection methods*: Even though edges are a crucial component of arrow-connected diagrams, their recognition in hand-drawn diagram images has received little attention in research. The edge recognition methods presented in this thesis addressed this gap. Chapter 5 introduced Arrow R-CNN, which extends Faster R-CNN with a network branch for detecting each arrow's head and tail keypoints. The keypoints are used to identify the connected shapes of an arrow. Chapter 6 presented Sketch2BPMN, which adapts Arrow R-CNN to the specifics of BPMN and proposes rules to identify invalid edges. Both methods only require a minor modification to the underlying Faster R-CNN object detector, and thus, incur only a negligible overhead to the inference time of the object detector. To address the inflexibility of the rule-based method, Chapter 7 proposed DiagramNet. DiagramNet introduces an edge prediction network that directly predicts whether two given shapes are connected through an edge. Finally, Chapter 8 presented Sketch2Process, which improves the edge prediction network such that it can also recognize arrows that connect two shapes with a large detour. The evaluation results showed that Sketch2Process considerably outperforms existing methods in edge recognition, at the cost of an increased inference time compared to the approaches based on Arrow R-CNN. However, we observed that the higher inference time has no effect on the overall system runtime, which is dominated by the response time of the external OCR service.
5. *Recognition of diagram labels using a three-step approach*: In Chapter 3, we found that the recognition of diagram labels is hardly considered in existing works. In Chapter 8, we thus proposed the first method that addresses all three subtasks of diagram label recognition: textblock detection, textblock handwriting recognition, and textblock relation detection. For textblock detection, we train an object detector to identify textblocks outside of activities. For textblock handwriting recognition, we use an off-the-shelf OCR service to recognize all words in an image. We found that the OCR service returns many false-positive words that are actually part of drawn shapes or edges, e.g., recognized "X" characters for exclusive gateway symbols or multiple returned "-" or "I" characters for horizontal and vertical arrows. These false-positive words are eliminated as part of our textual content decoding procedure, which, given the textblock bounding boxes and the recognized words, identifies the words that belong to each textblock and combines the words into the actual label. For textblock relation detection, we proposed a network that predicts whether a textblock belongs to a candidate shape or edge. An evaluation on the `hdBPMN` dataset demonstrated that our approach achieves promising results. In particular, it correctly detects and relates more than 90% of the textblocks.

9.2 Future Research

This thesis primarily focused on the recognition of BPMN diagrams sketched on paper. Future work could expand on our work in two directions. First, the scope of our recognition methods could be extended, e.g., by adapting them to other types of arrow-connected diagrams. Second, options to further improve the performance of our proposed recognition methods could be explored.

9.2.1 Recognition Scope

With respect to the recognition scope, future research could focus on (i) process models sketched on surfaces other than paper, (ii) process models generated with digital diagramming tools, and (iii) other modeling notations that can be expressed as arrow-connected diagrams.

(i) The `hdBPMN` dataset presented in Chapter 4 contains BPMN models sketched on paper. In practice, process models are also sketched on other physical surfaces, including whiteboard or brown paper. In addition, haptic tools, i.e., tools that rely on physical objects, might be used to make the modeling experience more engaging [24, p. 85]. Such haptic tools can range from simple post-its to magnetic BPMN elements out of a tool box [33]. In future work, it would be interesting to investigate the characteristics of process models obtained in such scenarios. Such an effort could result in a dataset of process model images sourced from organizations.

(ii) In Chapter 8, we have shown that our Sketch2Process approach is also highly accurate on computer-generated BPMN diagrams, as evaluated on the BPMN-Redrawer dataset [1]. However, this dataset has two limitations. First, it is limited in size. To improve the validity of our study, the recently published SAP-SAM dataset [89], which contains more than 100,000 BPMN models, could be used for an additional evaluation. Second, the images of the dataset have been generated by a single BPMN editor. As BPMN editors use different ways to graphically represent the same BPMN element, it would be interesting to collect a dataset of images produced by various editors. From an application perspective, a model trained on a more diverse dataset could be used in cases where only an image of a process model is available and the recreation of an editable BPMN model is requested. To support standardization efforts, another possible direction for future work is the recognition of process models created using free-form diagramming tools, such as Microsoft PowerPoint and Visio [64].

(iii) Last, it would be interesting to explore how well our approaches can be

adapted to other modeling notations. For example, the bachelor thesis by Schumacher [87] showed that DiagramNet can also be applied to recognize hand-drawn UML class diagrams. To improve the results and the validity of this study, it would be interesting to collect a larger UML dataset. Besides adapting our approaches to other diagram types, it would be interesting to explore to what extent generic aspects (e.g., recognizing arrowheads) can be learned and transferred from the `hdBPMN` dataset.

9.2.2 Recognition Method

Our evaluation in Chapter 8 showed very promising results for Sketch2Process. However, it is reasonable to explore other recognition methods to further improve the results. In the following, we conclude this thesis by providing three possible directions for future work, which involve (i) introducing a diagram detector, (ii) addressing the limitations of two-stage object detectors, and (iii) using a model that does not rely on an external OCR service.

(i) In Chapter 4, we have shown that the `hdBPMN` diagrams have a wide variety in terms of how they were captured as images. Most of the diagrams were photographed from a short distance and thus take up most of the image. However, some diagrams were photographed at a greater distance, which means that they occupy only a small part of the image. As common in object detection, our methods operate on images that have been resized to a longer side of 1,333px. For diagrams photographed from a distance, each element in the resized images becomes very small, which makes their recognition difficult. This problem could be addressed by decomposing diagram recognition into two steps. In the first step, an object detector could be trained to detect the diagrams in a given image. Given the detected bounding boxes, each diagram could then be cropped from the raw image, resulting in an image that tightly encloses the diagram. In the second step, each cropped image could be used as input for the actual diagram recognition model.

(ii) An inherent assumption of two-stage object detectors such as Faster R-CNN is that objects of the same class do not significantly overlap. This assumption plays a key role in the design of many components, such as the anchor generation procedure, the heuristic that assigns target boxes to anchors, and the non-maximum suppression procedure (cf. Section 2.2). However, this assumption does not always hold in diagrams, where two shapes or arrows can significantly overlap. In BPMN, this particularly affects nested lanes, where we observe an IoU of up to 97% (cf. Section 8.2.2). In the last years, transformer-based object detectors [13] have gained popularity, which do

not rely on hand-designed components that encode prior knowledge. These approaches have also been adapted to scene graph generation [55]. As mentioned in Section 2.4, there is a task overlap between recognizing scene graphs and diagrams in images. Therefore, it would be interesting to explore if transformer-based methods can be adapted to diagram recognition.

(iii) Finally, the exploration of diagram recognition approaches that do not rely on an external OCR service could be an interesting direction for future research. In the domain of visual document understanding (VDU), there has been a recent trend of pre-training multi-modal transformers on a large dataset and then fine-tuning them for downstream tasks such as invoice information extraction [2, 44, 94, 107, 109]. Most of these VDU approaches rely on an off-the-shelf OCR service. As the OCR results are part of the model input, the OCR service needs to run first, resulting in a slow end-to-end inference speed. To address this shortcoming, Kim et al. [44] propose the Donut approach, which does not require an OCR service but still achieves state-of-the-art performance on various VDU tasks in terms of both speed and accuracy. It would be interesting to explore if Donut can be fine-tuned for diagram recognition. As Donut formulates all downstream tasks as a JSON prediction problem, this would require designing a JSON-based representation for arrow-connected diagrams.

Bibliography

[1] Alessandro Antinori, Riccardo Coltrinari, Flavio Corradini, Fabrizio Fornari, Barbara Re, and Marco Scarpetta. BPMN-Redrawer: From Images to BPMN Models. In *Proceedings of the Demonstration Resources Track, Best BPM Dissertation Award, and Doctoral Consortium at BPM 2022*, page 5. CEUR Workshop Proceedings, September 2022.

[2] Srikar Appalaraju, Bhavan Jasani, Bhargava Urala Kota, Yusheng Xie, and R. Manmatha. DocFormer: End-to-End Transformer for Document Understanding. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 993–1003, 2021.

[3] Ahmad-Montaser Awal, Guihuan Feng, Harold Mouchère, and Christian Viard-Gaudin. First Experiments on a new Online Handwritten Flowchart Database. In *Document Recognition and Retrieval XVIII*, page 78740A, January 2011.

[4] Omar Badreddin, Khandoker Rahad, Andrew Forward, and Timothy Lethbridge. The evolution of software design practices over a decade: A long term study of practitioners. *Journal of Object Technology*, 20(2):2:1–19, 2021.

[5] Showmik Bhowmik, Ram Sarkar, Mita Nasipuri, and David Doermann. Text and non-text separation in offline document images: A survey. *Int. Journal on Document Analysis and Recognition*, 21(1-2):1–20, June 2018.

[6] M. Bresler, T. V. Phan, D. Prusa, M. Nakagawa, and V. Hlavác. Recognition System for On-Line Sketched Diagrams. In *2014 14th International Conference on Frontiers in Handwriting Recognition*, pages 563–568, September 2014.

[7] M. Bresler, D. Prùa, and V. Hlavác. Modeling Flowchart Structure Recognition as a Max-Sum Problem. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1215–1219, August 2013.

[8] M. Bresler, D. Prusa, and V. Hlavàc. Detection of Arrows in On-Line Sketched Diagrams Using Relative Stroke Positioning. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 610–617, January 2015.

[9] M. Bresler, D. Průša, and V. Hlaváč. Recognizing Off-Line Flowcharts by Reconstructing Strokes and Using On-Line Recognition Techniques. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 48–53, October 2016.

[10] Martin Bresler, Daniel Pruša, and Václav Hlavác. Simultaneous segmentation and recognition of graphical symbols using a composite descriptor. In *18th Computer Vision Winter Workshop*, volume 13, pages 16–23, 2013.

[11] Martin Bresler, Daniel Průša, and Václav Hlaváč. Online recognition of sketched arrow-connected diagrams. *International Journal on Document Analysis and Recognition (IJDAR)*, 19(3):253–267, September 2016.

[12] Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and Flexible Image Augmentations. *Information*, 11(2):125, February 2020.

[13] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-End Object Detection with Transformers. In *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 213–229. Springer International Publishing, 2020.

[14] C. Carton, A. Lemaitre, and B. Coüasnon. Fusion of Statistical and Structural Information for Flowchart Recognition. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1210–1214, August 2013.

[15] Suranjan Chakraborty, Saonee Sarker, and Suprateek Sarker. An exploration into the process of requirements elicitation: A grounded approach. *Journal of the association for information systems*, 11(4):1, 2010.

[16] Peter C-H. Cheng. Why Diagrams Are (Sometimes) Six Times Easier than Words: Benefits beyond Locational Indexing. In *Diagrammatic Representation and Inference*, Lecture Notes in Computer Science, pages 242–254. Springer, 2004.

[17] Mauro Cherubini, Gina Venolia, Rob DeLine, and Andrew J. Ko. Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. In

Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI '07, pages 557–566, 2007.

[18] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1, June 2005.

[19] B. Davis, B. Morse, S. Cohen, B. Price, and C. Tensmeyer. Deep Visual Template-Free Form Parsing. In *ICDAR*, pages 134–141, September 2019.

[20] Brian Davis, Bryan Morse, Brian Price, Chris Tensmeyer, and Curtis Wiginton. Visual FUDGE: Form Understanding via Dynamic Graph Editing. In *Document Analysis and Recognition – ICDAR 2021*, Lecture Notes in Computer Science, pages 416–431. Springer International Publishing, 2021.

[21] D. Doermann, Jian Liang, and Huiping Li. Progress in camera-based document image analysis. In *Int. Conf. on Document Analysis and Recognition*, pages 606–616 vol.1, 2003.

[22] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021.

[23] David H Douglas and Thomas K Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica*, 10(2):112–122, December 1973.

[24] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo Reijers. *Fundamentals of Business Process Management*. Springer, 2 edition, 2018.

[25] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *Int J Comput Vis*, 88(2):303–338, June 2010.

[26] Jiaqi Fang, Zhen Feng, and Bo Cai. DrawnNet: Offline Hand-Drawn Diagram Recognition Based on Keypoint Prediction of Aggregating Geometric Characteristics. *Entropy*, 24(3):425, March 2022.

[27] Philippe Gervais, Thomas Deselaers, Emre Aksan, and Otmar Hilliges. The DIDI dataset: Digital Ink Diagram data. *arXiv:2002.09303 [cs]*, February 2020.

[28] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 580–587, 2014.

[29] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[30] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour. *arXiv:1706.02677 [cs]*, June 2017.

[31] Mark D. Gross. Recognizing and interpreting diagrams in design. In *Proceedings of the Workshop on Advanced Visual Interfaces*, AVI '94, pages 88–94. Association for Computing Machinery, June 1994.

[32] Mark D. Gross and Ellen Yi-Luen Do. Ambiguous intentions: A paper-like interface for creative design. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, UIST '96, pages 183–192. Association for Computing Machinery, November 1996.

[33] Alexander Grosskopf, Jonathan Edelman, and Mathias Weske. Tangible Business Process Modeling – Methodology and Experiment Design. In *Business Process Management Workshops*, Lecture Notes in Business Information Processing, pages 489–500. Springer Berlin Heidelberg, 2010.

[34] Tracy Hammond and Randall Davis. Tahuti: A Geometrical Sketch Recognition System for UML Class Diagrams. In *Proceedings of AAAI Spring Symposium on Sketch Understanding*, pages 59–68, 2002.

[35] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask R-CNN. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2961–2969, 2017.

[36] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[37] Jorge-Ivan Herrera-Camara and Tracy Hammond. Flow2Code: From Hand-drawn Flowcharts to Code Execution. In *Proceedings of the Symposium on Sketch-Based Interfaces and Modeling*, SBIM '17, pages 3:1–3:13, 2017.

[38] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *arXiv:1704.04861 [cs]*, April 2017.

[39] Ramesh Jain, Rangachar Kasturi, and Brian G. Schunck. *Machine Vision*. McGraw-Hill, Inc., 1995.

[40] Timothy E. Johnson. Sketchpad III: A computer program for drawing in three dimensions. In *Proceedings of the May 21-23, 1963, Spring Joint Computer Conference*, AFIPS '63 (Spring), pages 347–353. Association for Computing Machinery, May 1963.

[41] F. D. Julca-Aguilar and N. S. T. Hirata. Symbol Detection in Online Handwritten Graphics Using Faster R-CNN. In *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, pages 151–156, April 2018.

[42] Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, and Nina S. T. Hirata. A general framework for the recognition of online handwritten graphics. *International Journal on Document Analysis and Recognition (IJDAR)*, January 2020.

[43] Qingyi Kang, Yong Yu, and Fei Dai. Automatic Recognition of Business Process Images. In *2019 IEEE International Conference on Computer Science and Educational Informatization (CSEI)*, pages 104–113, August 2019.

[44] Geewook Kim, Teakgyu Hong, Moonbin Yim, JeongYeon Nam, Jinyoung Park, Jinyeong Yim, Wonseok Hwang, Sangdoo Yun, Dongyoon Han, and Seunghyun Park. OCR-Free Document Understanding Transformer. In *Computer Vision – ECCV 2022*, Lecture Notes in Computer Science, pages 498–517. Springer Nature Switzerland, 2022.

[45] Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen, Yannis Kalantidis, Li-Jia Li, David A. Shamma, Michael S. Bernstein, and Li Fei-Fei. Visual Genome: Connecting Language and Vision Using Crowdsourced Dense Image Annotations. *International Journal of Computer Vision*, 123(1):32–73, May 2017.

[46] Praveen Krishnan and C. V. Jawahar. HWNet v2: An efficient word image representation for handwritten documents. *International Journal on Document Analysis and Recognition (IJDAR)*, 22(4):387–405, December 2019.

[47] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[48] James A. Landay and Brad A. Myers. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '95, pages 43–50. ACM Press/Addison-Wesley Publishing Co., May 1995.

[49] Jill H. Larkin and Herbert A. Simon. Why a Diagram is (Sometimes) Worth Ten Thousand Words. *Cognitive Science*, 11(1):65–100, January 1987.

[50] Hei Law and Jia Deng. CornerNet: Detecting Objects as Paired Keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[51] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

[52] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

[53] Aurélie Lemaitre, Harold Mouchère, Jean Camillerapp, and Bertrand Coüasnon. Interest of Syntactic Knowledge for On-Line Flowchart Recognition. In *Graphics Recognition. New Trends and Challenges*, Lecture Notes in Computer Science, pages 89–98, 2013.

[54] Henrik Leopold, Jan Mendling, and Oliver Günther. Learning from quality issues of bpmn models from industry. *IEEE software*, 33(4):26–33, 2015.

[55] Rongjie Li, Songyang Zhang, and Xuming He. SGTR: End-to-End Scene Graph Generation With Transformer. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 19486–19496, 2022.

[56] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1–1, 2018.

[57] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection.

In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2117–2125, 2017.

[58] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. In *Computer Vision – ECCV 2014*, Lecture Notes in Computer Science, pages 740–755, 2014.

[59] Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothee Cour, Kai Yu, Liangliang Cao, and Thomas Huang. Large-scale image classification: Fast feature extraction and SVM training. In *CVPR 2011*, pages 1689–1696, June 2011.

[60] Cewu Lu, Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Visual Relationship Detection with Language Priors. In *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 852–869. Springer International Publishing, 2016.

[61] U.-V. Marti and H. Bunke. The IAM-database: An English sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5(1):39–46, November 2002.

[62] Francisco Massa and Ross Girshick. Maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark, 2018.

[63] Roland Morzinger, Rene Schuster, Andras Horti, and Georg Thallinger. Visual Structure Analysis of Flow Charts in Patent Images. In *Working Notes for CLEF 2012 Conference*, page 8. CEUR Workshop Proceedings, September 2012.

[64] Debdoot Mukherjee, Pankaj Dhoolia, Saurabh Sinha, Aubrey J. Rembert, and Mangala Gowri Nanda. From Informal Process Diagrams to Formal Process Models. In *Business Process Management*, Lecture Notes in Computer Science, pages 145–161. Springer Berlin Heidelberg, 2010.

[65] Nicholas Negroponte. Recent advances in sketch recognition. In *Proceedings of the June 4-8, 1973, National Computer Conference and Exposition*, AFIPS '73, pages 663–675. Association for Computing Machinery, June 1973.

[66] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked Hourglass Networks for Human Pose Estimation. In *Computer Vision – ECCV 2016*, Lecture Notes in Computer Science, pages 483–499, 2016.

[67] Matt Notowidigdo and Robert C Miller. Off-line sketch interpretation. In *AAAI Fall Symposium*, pages 120–126. Arlington, VA, 2004.

[68] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[69] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32*, volume 32, pages 8024–8035. Curran Associates, Inc., 2019.

[70] Jakob Pinggera, Pnina Soffer, Dirk Fahland, Matthias Weidlich, Stefan Zugal, Barbara Weber, Hajo A. Reijers, and Jan Mendling. Styles in business process modeling: An exploration and a model. *Software & Systems Modeling*, 14(3):1055–1080, July 2015.

[71] R. Plamondon and S. N. Srihari. Online and off-line handwriting recognition: A comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, January 2000.

[72] Pablo Pytel, Matías Almad, Rocío Leguizamón, Cinthia Vegega, and Ma Florencia Pollo-Cattaneo. Object Detection Based Software System for Automatic Evaluation of Cursogramas Images. In *Applied Informatics*, Communications in Computer and Information Science, pages 39–54. Springer International Publishing, 2021.

[73] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016.

[74] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6517–6525. IEEE, July 2017.

[75] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv preprint arXiv:1804.02767*, April 2018.

[76] Khaled S. Refaat, Wael N. Helmy, AbdelRahman H. Ali, Mohamed S. AbdelGhany, and Amir F. Atiya. A new approach for context-independent handwritten offline diagram recognition using support vector machines. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 177–182, June 2008.

[77] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28*, pages 91–99, 2015.

[78] Marçal Rusiñol, Lluís-Pere de las Heras, and Oriol Ramos Terrades. Flowchart recognition for non-textual information retrieval in patent search. *Inf Retrieval*, 17(5):545–562, October 2014.

[79] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *Int J Comput Vis*, 115(3):211–252, December 2015.

[80] Joan Andreu Sánchez, Verónica Romero, Alejandro H. Toselli, Mauricio Villegas, and Enrique Vidal. ICDAR2017 Competition on Handwritten Text Recognition on the READ Dataset. In *ICDAR*, pages 1383–1388, 2017.

[81] J. Sauvola and M. Pietikäinen. Adaptive document image binarization. *Pattern Recognition*, 33(2):225–236, February 2000.

[82] Bernhard Schäfer, Margret Keuper, and Heiner Stuckenschmidt. Arrow R-CNN for handwritten diagram recognition. *IJDAR*, February 2021.

[83] Bernhard Schäfer and Heiner Stuckenschmidt. Arrow R-CNN for Flowchart Recognition. In *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, page 7, September 2019.

[84] Bernhard Schäfer and Heiner Stuckenschmidt. DiagramNet: Hand-Drawn Diagram Recognition Using Visual Arrow-Relation Detection. In *Int. Conf. on Document Analysis and Recognition*, pages 614–630, 2021.

[85] Bernhard Schäfer, Han van der Aa, Henrik Leopold, and Heiner Stuckenschmidt. Sketch2BPMN: Automatic recognition of hand-drawn BPMN models. In *Int. Conf. on Advanced Information Systems Engineering*, pages 344–360. Springer, 2021.

[86] Bernhard Schäfer, Han van der Aa, Henrik Leopold, and Heiner Stuckenschmidt. Sketch2Process: End-to-end BPMN Sketch Recognition Based on Neural Networks. *IEEE Transactions on Software Engineering*, pages 1–21, 2022.

[87] Philipp Nicolas Schumacher. *Ein Datensatz handgezeichneter UML-Klassendiagramme für maschinelle Lernverfahren*. Abschlussarbeit - Bachelor, Karlsruher Institut für Technologie (KIT) / Karlsruher Institut für Technologie (KIT), 2021.

[88] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, ICDAR '03, pages 958–, 2003.

[89] Diana Sola, Christian Warmuth, Bernhard Schäfer, Peyman Badakhshan, Jana-Rebecca Rehse, and Timotheus Kampik. SAP Signavio Academic Models: A Large Process Model Dataset, August 2022.

[90] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep High-Resolution Representation Learning for Human Pose Estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5693–5703, 2019.

[91] Lianshan Sun, Hanchao Du, and Tao Hou. FR-DETR: End-to-End Flowchart Recognition With Precision and Robustness. *IEEE Access*, 10:64292–64301, 2022.

[92] Xiao Sun, Bin Xiao, Fangyin Wei, Shuang Liang, and Yichen Wei. Integral Human Pose Regression. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 529–545, 2018.

[93] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Texts in Computer Science. Springer International Publishing, 2022.

[94] Zineng Tang, Ziyi Yang, Guoxin Wang, Yuwei Fang, Yang Liu, Chenguang Zhu, Michael Zeng, Cha Zhang, and Mohit Bansal. Unifying Vision, Text, and Layout for Universal Document Processing, December 2022.

[95] Maxim Tkachenko, Mikhail Malyuk, Andrey Holmanyuk, and Nikolai Liubimov. Label Studio: Data labeling software, 2020-2022. Open source software available from https://github.com/heartexlabs/label-studio.

[96] Alexander Toshev and Christian Szegedy. DeepPose: Human Pose Estimation via Deep Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1653–1660, 2014.

[97] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. Scikit-image: Image processing in Python. *PeerJ*, 2:e453, June 2014.

[98] Bintu G. Vasudevan, Sorawish Dhanapanichkul, and Rajesh Balakrishnan. Flowchart knowledge extraction on image processing. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 4075–4082, June 2008.

[99] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All you Need. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[100] Jagoda Walny, Sheelagh Carpendale, Nathalie Henry Riche, Gina Venolia, and Philip Fawcett. Visual thinking in action: Visualizations as used on whiteboards. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2508–2517, 2011.

[101] C. Wang, H. Mouchère, C. Viard-Gaudin, and L. Jin. Combined Segmentation and Recognition of Online Handwritten Diagrams with High Order Markov Random Field. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 252–257, October 2016.

[102] Chengcheng Wang, Harold Mouchère, Aurélie Lemaitre, and Christian Viard-Gaudin. Online flowchart understanding by combining max-margin Markov random field with grammatical analysis. *International Journal on Document Analysis and Recognition (IJDAR)*, 20(2):123–136, June 2017.

[103] Jie Wu, Changhu Wang, Liqing Zhang, and Yong Rui. Offline Sketch Parsing via Shapeness Estimation. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*, June 2015.

[104] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

[105] Bin Xiao, Haiping Wu, and Yichen Wei. Simple Baselines for Human Pose Estimation and Tracking. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 466–481, 2018.

[106] Danfei Xu, Yuke Zhu, Christopher B. Choy, and Li Fei-Fei. Scene Graph Generation by Iterative Message Passing. In *CVPR*, pages 5410–5419, 2017.

[107] Yang Xu, Yiheng Xu, Tengchao Lv, Lei Cui, Furu Wei, Guoxin Wang, Yijuan Lu, Dinei Florencio, Cha Zhang, Wanxiang Che, Min Zhang, and Lidong Zhou. LayoutLMv2: Multi-modal Pre-training for Visually-rich Document Understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2579–2591. Association for Computational Linguistics, August 2021.

[108] Yifan Xu, Weijian Xu, David Cheung, and Zhuowen Tu. Line Segment Detection Using Transformers without Edges. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4255–4264, June 2021.

[109] Yiheng Xu, Minghao Li, Lei Cui, Shaohan Huang, Furu Wei, and Ming Zhou. LayoutLM: Pre-training of Text and Layout for Document Image Understanding. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, pages 1192–1200. Association for Computing Machinery, August 2020.

[110] Jianwei Yang, Jiasen Lu, Stefan Lee, Dhruv Batra, and Devi Parikh. Graph R-CNN for Scene Graph Generation. In *ECCV*, pages 670–685, 2018.

[111] Yuhong Yu, A. Samal, and S.C. Seth. A system for recognizing a large class of engineering drawings. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(8):868–890, August 1997.

[112] Xiao-Long Yun, Yan-Ming Zhang, Jun-Yu Ye, and Cheng-Lin Liu. Online Handwritten Diagram Recognition with Graph Attention Networks. In *Image and Graphics*, Lecture Notes in Computer Science, pages 232–244, 2019.

[113] Manuel Zapp, Peter Fettke, and Peter Loos. Towards a Software Prototype Supporting Automatic Recognition of Sketched Business Process Models. *Wirtschaftsinformatik 2017 Proceedings*, January 2017.

[114] Rowan Zellers, Mark Yatskar, Sam Thomson, and Yejin Choi. Neural Motifs: Scene Graph Parsing With Global Context. In *CVPR*, pages 5831–5840, 2018.

[115] Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling Vision Transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12104–12113, 2022.

[116] Feng Zhang, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. Distribution-Aware Coordinate Representation for Human Pose Estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7093–7102, 2020.

[117] Ji Zhang, Kevin J. Shih, Ahmed Elgammal, Andrew Tao, and Bryan Catanzaro. Graphical Contrastive Losses for Scene Graph Parsing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, March 2019.

[118] Ji Zhang, Kevin J. Shih, Ahmed Elgammal, Andrew Tao, and Bryan Catanzaro. Graphical contrastive losses for scene graph parsing. In *Conf. on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[119] Zhuoyao Zhong, Lei Sun, and Qiang Huo. An anchor-free region proposal network for Faster R-CNN-based text detection approaches. *International Journal on Document Analysis and Recognition (IJDAR)*, 22(3):315–327, September 2019.

Appendix A

Sketch2Process Supplementary Material

This chapter contains the supplementary material of our `Sketch2Process` approach described in Chapter 8. Section A.1 details the training procedure of the edge relation network. Section A.2 gives a detailed overview of the textblock relation network architecture and training procedure. Section A.3 elaborates on the word reading order detection algorithm that we use as part of the procedure that decodes the textual content of each textblock. Section A.4 details the image augmentation pipeline that we apply in our work. Last, Section A.5 reports on an experiment that we conducted to evaluate our approach on computer-generated BPMN models.

A.1 Edge Relation Network Training

This section provides the details of our training procedure for the edge relation network. Given an augmented image, Faster R-CNN produces a set of object proposals, which is illustrated in Figure 8.8. As shown in Figure 8.8, edge relation network training involves generating the input (sampled edge candidates) and computing the output (edge loss) of the network. During inference, the edge candidate generation procedure receives the set of detected shapes and arrows. During training, we use a modified sampling-based *edge candidate generation* procedure that generates the input by sampling edge candidates from the set of object proposals. Regarding the output, we compute an aggregated *edge loss* over the sampled edge candidates. In the following, we explain these two steps in more detail.

A.1.1 Edge Candidate Generation

For sampling the edge candidates, we first identify all proposal candidates, i.e., all object proposals that sufficiently overlap with a ground-truth shape or edge. Concretely, a proposal is considered a candidate and assigned to a ground-truth shape or edge if the IoU between both boxes exceeds a threshold of 0.7. By default, Faster R-CNN keeps the top 1000 object proposals ranked by their objectness score. Further, it uses an IoU threshold of 0.5 to match proposals to ground-truth objects. We use a more strict IoU threshold to identify proposal candidates for our edge recognition use case, as we find that an IoU threshold of 0.5 leads to unstable training. This might be due to the specifics of our use case, where two arrow objects in proximity can have a large bounding box overlap. To ensure that the number of proposal candidates is still sufficiently high, we increase the number of Faster R-CNN object proposals from 1,000 to 2,000. Since the number of proposals is very large, the number of identified proposal candidates is typically much larger than the number of ground-truth objects. To give an example, the augmented image in Figure 8.8 contains 13 relevant ground-truth shape and 11 arrow objects, for which we identify 45 shape and 41 arrow proposal candidates. After identifying the proposal candidates, we use them as input for the edge candidate generation procedure to obtain the set of edge candidates. In the given example, edge candidate generation produces 2,019 edge candidates, of which 467 are positive and 1,552 are negative. Here, an edge candidate is considered positive if the ground-truth arrow associated with the arrow proposal indeed connects the ground-truth shapes associated with the shape proposals. Finally, we sample 64 edge candidates per image, with a positive ratio of 50%. In the rare case that the image has less than 32 positive candidates, the remaining slots are filled up with additional negative candidates.

Overall, our proposal-based sampling procedure has several advantages compared to directly using the set of ground-truth shapes and edges. First, as mentioned, the number of generated edge candidates is much larger, which means that we can sample from a much larger set. Second, the network learns to predict edges even in the presence of bounding box localization errors, which increases robustness. As the proposal boxes are also used to extract the spatial and visual features of the edge relation network, this greatly reduces overfitting, as it is much harder for the network to simply memorize an edge given the three bounding boxes of the edge candidate. Third, the sampling ratio effectively addresses the class imbalance problem. As most edge candidates are negative, the classifier would be biased without a sampling mechanism, which would make it more challenging to find a suitable score threshold during inference.

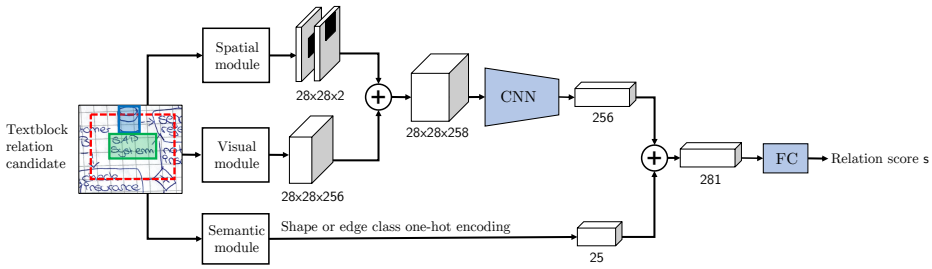


Figure A.1: Textblock relation network

A.1.2 Edge Loss

For each sampled edge candidate, the edge relation network predicts a relation score and arrow keypoints. We compute the edge relation loss using binary cross entropy. For the keypoint loss, we use the same keypoint encoding, loss method, and ground-truth keypoint formulation as in [84]. The keypoint loss is computed as the mean squared error between the encoded predicted and ground-truth keypoints, which we average over the number of keypoints. Further, the intermediate ground-truth keypoints are sampled in an equidistant manner from the annotated arrow path, i.e. the keypoints are chosen such that the distance between all successive keypoint pairs is the same. Since the network should learn to predict keypoints consistent with the source and target shape, we only compute the keypoint loss for positive edge candidates. Given the relation and keypoint loss, we compute the edge loss for a candidate as a weighted combination of the two. We find that a relation loss weight of 2 and a keypoint loss weight of 5 sufficiently balances these loss terms with the other losses. As mentioned, the keypoint loss is only defined for positive candidates, which is equivalent to setting it to 0 for negative candidates. Finally, we obtain the edge loss $\mathcal{L}_e$ as the mean loss over all sampled edge candidates.

A.2 Textblock Relation Network Training

This section provides the details of our proposed textblock relation network and addresses both its architecture and its training procedure.

A.2.1 Network Architecture

The architecture of our proposed textblock relation network is illustrated in Figure A.1. The network is very similar to the edge relation network but operates

on object pairs instead of triplets. As in edge recognition, the visual module uses the textblock region bounding box to extract a visual feature representation. The spatial module generates two binary masks that encode the bounding boxes of the textblock and the target shape or edge. The semantic module only one-hot encodes the predicted target shape or edge class, as the source of the relation is always a textblock. We combine the features of the different modules in the same way as in the edge relation network. Further, we use an almost identical network architecture. The network only differs in the dimensionality of the first CNN and the first FC layer, to account for the different number of binary masks and semantic features. The last layer of the network is a binary classification layer, which predicts the textblock relation score.

A.2.2 Network Training

The training procedures of the textblock and the edge relation network are very similar. Therefore, this section focuses on the differences between both.

Relation candidate generation. As in edge candidate generation, we sample proposal candidates from the Faster R-CNN object proposals. In edge candidate generation, we use a more strict IoU threshold to identify proposals candidates, in order to prevent unstable training. We find that this is not required for generating textblock relation candidates. Therefore, we match proposals to ground-truth objects using the standard IoU threshold of 0.5. Given the proposal candidates, we run the relation candidate generation procedure to produce a set of relation candidates. As in edge candidate generation, we sample 64 candidates per image, with a positive ratio of 50%.

Textblock relation loss. We compute the textblock relation loss $\mathcal{L}_t$ in the same way as the edge relation loss, and also use a relation loss weight of 2.

A.3 Word Reading Order Detection

Our `Sketch2Process` approach contains a two-stage *textual content decoding* step. In the first stage, we identify the words W^t that belong to each textblock t. In the second stage, given the words W^t assigned to a textblock t, we use a word reading order detection algorithm to determine the textual content txt of the textblock, which we detail in the following.

The convention in BPMN is to only use rotated text for pool and lane labels, where the text is rotated by 90° counter-clockwise, i.e., the reading direction is bottom-to-top. The exception to this convention are collapsed pools, where the textblock is commonly located in the center of the pool and not rotated. We observe

that textblocks in the hdBPMN dataset also largely adhere to this convention. Of the textblocks that label neither pools nor lanes, around 99% are written in the standard left-to-right direction. Overall, except for some rare cases, the textual content of the textblocks in the hdBPMN has three orientations: not rotated, 90° counter-clockwise, and 90° clockwise. Our word reading method, therefore, assumes that the text is rotated by a multiple of 90°. To this end, we discretize the word rotation angle $w.d$ to the nearest of the three orientations.

Given the bounding box, orientation, and textual content of all words associated with a textblock, our word reading order algorithm works as follows: First, we define the textblock orientation as the most common orientation among all words assigned to a textblock. Next, we compute an average word height W_h as the median word height of all words in the image that have the same orientation as the textblock. We find that averaging over all words in the image reduces variance and is thus more robust than only considering the textblock words. In order to separate words into lines, we compute a word distance matrix A, where each entry A_{ij} is defined by the absolute difference of the y-center coordinates (x-center for rotated textblocks) of the bounding boxes of the words w_i and w_j. We then consider two words w_i and w_j on the same line if their distance A_{ij} is lower than $\tau_{ld} * W_h$. Based on our experimental findings, we set the line distance threshold τ_{ld} to 0.4. After applying the criterion, we obtain a binary matrix A' of the matrix A. Next, we obtain the textblock lines by computing the connected components of A'. Finally, we obtain the textual content of the line by arranging the words by their x-center coordinate (y-center for rotated textblocks).

A.4 Image Augmentation

We use the Albumentations library to implement our augmentation pipeline [12]. Albumentations offers existing implementations for all image augmentation methods that we use in our approach. The code listing in Listing A.1 shows how to implement our proposed image augmentation pipeline in Albumentations.

A.5 Performance on Computer-generated BPMN Models

While Sketch2Process was designed to recognize hand-drawn BPMN models, it naturally can also be used to recognize computer-generated BPMN models. In this section, we report on an experiment we conducted to demonstrate the performance of Sketch2Process on computer-generated BPMN models.

Baseline. We compare Sketch2Process against BPMN-Redrawer [1], the state-of-the-art approach for computer-generated BPMN model recognition. From

```python
from albumentations import *

transforms = [
    OneOf([
        LongestMaxSize(p=0.7, max_size=1333),
        RandomResizedCrop(
            p=0.3, height=1333, width=1333,
            scale=(0.2, 1.0), ratio=(0.5, 2.0)
        ),
    ], p=1.0),
    RandomRotate90(p=0.3),
    HorizontalFlip(p=0.3),
    VerticalFlip(p=0.3),
    ShiftScaleRotate(
        p=0.3, shift_limit=0.03,
        scale_limit=(-0.2, 0.0), rotate_limit=5,
        border_mode=cv2.BORDER_CONSTANT, value=(255, 255, 255)
    ),
    GaussNoise(p=0.3),
    OneOf([
        CLAHE(),
        Sharpen(),
        RandomBrightnessContrast()
    ], p=0.3),
    HueSaturationValue(p=0.3, val_shift_limit=0),
]

aug = Compose(
    transforms=transforms,
    bbox_params=BboxParams(
        format="coco", min_area=10.0,
        min_visibility=0.7
    ),
    keypoint_params=KeypointParams(format="xy")
)
```

Listing A.1: Albumentations image augmentation pipeline

Table A.1: Average precision (AP) results for bounding boxes and keypoints obtained for the test set of the BPMN-Redrawer dataset

	Shape	**Edge**	
Approach	**AP-box**	**AP-box**	**AP-keypoints**
BPMN-Redrawer	96.0	73.9	89.9
Sketch2Process	**98.5**	**95.5**	**95.7**

an architectural perspective, the BPMN-Redrawer approach is very similar to Arrow R-CNN [82], but builds on two different neural networks, one to recognize shapes, and one to recognize arrows. For the recognition of *shapes*, they train the Faster R-CNN object detector. For recognizing *edges*, they first detect arrow objects and their keypoints by training the Mask R-CNN keypoint detector [35]. Then, they use the heuristics-based procedure of Arrow R-CNN to identify the shapes that each arrow connects. For the recognition of *labels*, they build on the Tesseract OCR engine, which can only recognize machine-printed characters.

Dataset. We use the publicly available dataset from the BPMN-Redrawer approach [1]. It consists of 828 BPMN models and corresponding computer-generated images. However, the BPMN-Redrawer approach does not use all these models for their evaluation. While the shape recognition performance is evaluated on all models, edge recognition is only evaluated on 492 models due to labeling issues. Therefore, we also limit ourselves to the same 492 models for evaluating edge recognition.

Metrics. To provide a fair comparison, we use the performance metrics of the BPMN-Redrawer approach. For shape recognition, they use average precision (AP), a popular metric in computer vision to assess object detection, which they report for each shape category. As we want to boil down the object detection performance to a single metric, we follow the common practice and compute the mean AP over all categories. For edge recognition, they also use AP to evaluate the arrow objects and use the keypoint AP to evaluate arrow keypoint detection. However, they do not evaluate edge recognition end-to-end. Concretely, they do not evaluate if the shapes that connect each arrow have been correctly identified. Since label recognition is not evaluated at all, we consider this out of scope for this experiment.

Results. Table A.1 shows the results of our conducted experiment. As indicated, `Sketch2Process` performs slightly better in shape recognition and outperforms BPMN-Redrawer substantially in both edge recognition metrics.

Appendix B

hdBPMN Modeling Tasks and Instructions

The following provides the modeling tasks and instructions from the exercise sheet and exam questions that were used to collect the hdBPMN dataset. The modeling instructions have been translated from German to English for clarity.

Exercise ex00

Model the following business process for opening a bank account:

When the bank receives a new online application for opening a bank account, the application is evaluated. If the application is rejected, the customer is notified by email and the process ends. If the application is approved, a new bank account is created. Before the process ends, the bank sends a welcome pack, a bank card, and a PIN number in separate letters to the customer.

Instructions:

- Only use the following BPMN elements: task, events, gateways, sequence flow

Exercise ex01

Model the following business process for assessing credit risks:

When a new credit request is received, the risk is assessed. If the risk is above a threshold, an advanced risk assessment needs to be carried out, otherwise a simple

risk assessment will suffice. Once the assessment has been completed, the customer is notified with the result of the assessment, and in the meantime the disbursement is organized.

For simplicity, you can assume that the result of an assessment is always positive.

Instructions:

- Only use the following BPMN elements: task, events, gateways, sequence flow

Exercise ex02

Model the following fragment of a business process for insurance claims.

When the insurer receives a claim of a customer, it is registered and examined by a claims officer who then writes a settlement recommendation. This recommendation is then checked by a senior claims officer who may mark the claim as "OK" or "Not OK". If the claim is marked as "Not OK", it is sent back to the claims officer and the recommendation is repeated. If the claim is "OK", the claim handling process proceeds.

Instructions:

- Model the relevant resources (pools, lanes) and business objects (data object, data store).

Exercise ex03

Model the following business process of an insurer for handling a claim. The claim is submitted by a claimant.

When a claim is received, a claims officer first checks if the claimant is insured. If not, the claimant is informed that the claim must be rejected by sending an automatic notification via an SAP system. Otherwise, a senior claims officer evaluates the severity of the claim. Based on the outcome (simple or complex claims), the relevant forms are sent to the claimant, again using the SAP system. Once the forms are returned, they are checked for completeness by the claims officer. If the forms provide all relevant details, the claim is registered in the claims management system, and the process ends. Otherwise, the claimant is informed to update the forms via the SAP system. Upon reception of the updated forms, they are checked again by the claims officer to see if the details have been provided.

Instructions:

- Model the relevant resources (pools, lanes) and business objects (data object, data store).

Exercise ex04

Model the following business process for damage compensation at rental properties:

If a tenant is evicted because of damages to the premises, a process needs to be started by the tribunal in order to hold a hearing to assess the amount of compensation the tenant owes to the owner of the premises. This process starts when a cashier of the tribunal receives a request for compensation from the owner. The cashier then retrieves the file for those particular premises and checks that the request is both acceptable for filing and compliant with the description of the premises on file. After these checks, the cashier needs to set a hearing date. Setting a hearing date incurs fees to the owner. It may be that the owner has already paid the fees with the request, in which case the cashier allocates a hearing date and the process completes. If the owner has not paid the required fees, the cashier produces a fees notice and waits for the owner to pay the fees before reassessing the document compliance.

For simplicity, you can assume that the request always passes the checks.

Instructions:

- Model the relevant resources (pools, lanes) and business objects (data object, data store).

Exercise ex05

Model the following business process for processing car damage claims:

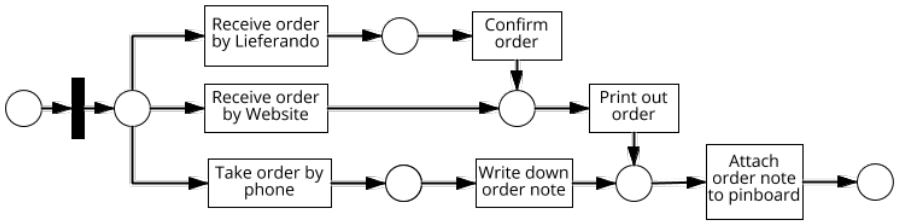
The motor claim handling process starts when a customer submits a claim with the relevant documentation. The notification department at the car insurer checks the documents upon completeness and registers the claim. Next, the Handling department picks up the claim and checks the insurance. Then, an assessment is performed. If the assessment is positive, a garage is phoned to authorize the repairs and the payment is scheduled (in this order). Otherwise, the claim is rejected. The claim is also immediately rejected if any of the previous checks fail. In any case (whether the outcome is positive or negative), a letter is sent to the customer and the process is considered to be complete.

Instructions:

- Model the relevant resources (pools, lanes) and business objects (data object, data store).

Exercise ex06

In a pizzeria, one person centrally records all incoming orders and attaches them to a pinboard. Translate the following Petri net of this process into a BPMN model:



Instructions:

- Use Business Objects where needed.

Exercise ex07

Model the following business process for rating doctors in a hospital:

The doctor rating workflow at a hospital is carried out by two different roles. The first one is a quality assurance (QA) specialist from the quality assurance department, while the second one represents the managing director of the hospital.

The QA specialist starts a new case regarding a certain doctor by interviewing patients. Since a patient interview workflow is already established, it is simply integrated in the new workflow. Meanwhile, the director asks an external expert to review the work of the doctor under rating. Unfortunately, since the expert only gets a low expenses fee, it can happen that the expert is not responding in time. If that happens, another expert has to be asked (who could also not respond in time, i.e. the procedure repeats). If an expert finally sends an expertise, it is received by the director and forwarded to the QA specialist. The QA specialist then files the results containing the patient interviews as well as the expertise and afterwards creates a report. While the QA specialist is doing this, the manager pays the expenses of the expert by filling a cheque and sending it to the expert.

Instructions:

- Model the relevant resources (pools, lanes)
- Modeling the relevant business objects (data object, data store) is not required.

Exercise ex08

Model the following process which describes the selection and allocation of elective courses at a school:
Version a):

Students must book two elective courses from the 5th school year onwards. Each year ten teachers are responsible for offering elective courses (each teacher one course). A course description is given by each teacher to the secretary's office at least one month before the start of the school year. The secretary collects all ten descriptions and enters the information into the course booking system (CBS). One week before the start of the school year, all responsible teachers receive a list of students who have registered for their course from the secretary. The teachers who have too many registered students choose which students can participate in the course. As a general rule, students in a higher year have priority. The (potentially empty) list of students who have not been accepted is handed over by each teacher to the secretary's office. The secretary then assigns these students to courses in which there are still free places and rebooks the students in the CBS.

Version b):

Students must book two elective courses from the 5th school year onwards. Each year twelve teachers are responsible for offering elective courses (each teacher one course). A course description is given by each teacher to the secretary's office at least two months before the start of the school year. The secretary collects all twelve descriptions and enters the information into the course information system (CIS). Two weeks before the start of the school year, all responsible teachers receive a list of students who have registered for their course from the secretary. The teachers who have too many registered students choose which students can participate in the course. As a general rule, students in a lower year have priority. The (potentially empty) list of students who have not been accepted is handed over by each teacher to the secretary's office. The secretary then assigns these students to courses in which there are still free places and rebooks the students in the CIS.

Instructions:

1. Model each resource as white-box pool, i.e. do not use black-box pools nor pools with more than one lane.
2. Modeling business objects is not required.

Exercise ex09

Model the following business process which describes the admission process of a PhD program at a university:
Version a):

To apply for the PhD program, students first fill in an online application form with their personal data. Online applications are recorded in an admission information system to which all staff members involved in the process have access. After a student has submitted the online form, a PDF document is generated and the student is requested to download it, sign it, and send it by post together with a transcript of grades and a letter of motivation. When these documents are received by the admissions office, the officer makes an initial assessment and rejects the application if the student has insufficient grades (such notifications of rejection are sent by email). In case of sufficient grades, the admissions office forwards the student documents by internal mail to the academic committee, which is responsible for deciding whether to offer admission or not. The committee meets once every month to examine all applications that are ready for academic assessment at the time of the meeting. At the end of the committee meeting, the chair of the committee notifies the admissions office of the selection outcomes. A few days later, the admissions office checks the selection outcomes and sends a rejection or admission email to each candidate.

Version b):

To apply for the PhD program, students first fill in an online application form with their personal data. Online applications are recorded in an application information system to which all members involved in the process have access. After a student has submitted the online form, a PDF document is generated and the student is requested to download it, sign it, and send it by post together with their diploma and a letter of motivation. When these documents are received by the administration office, the officer makes an initial assessment and rejects the application if the student's motivation letter is not convincing (such notifications of rejection are sent by letter). In case of a convincing letter of motivation, the administration office forwards the student documents by internal mail to the academic committee, which is responsible for deciding whether to offer admission or not. The committee

meets once every two months to examine all applications that are ready for academic assessment at the time of the meeting. At the end of the committee meeting, the committee notifies the administration office of the outcomes. A few weeks later, the administration office checks the outcomes and sends a rejection or admission letter to each candidate.

Instructions:

1. Model the relevant resources (pools, lanes). The process should be modeled from the university point of view, i.e. the student can be modeled as a black box.
2. Modeling business objects is not required.

Exercise ex10

Model the following business process between a supplier and a retailer:

After a retailer requests an offer from a supplier, the supplier prepares an offer and sends it to the retailer. Next, the supplier can receive an order confirmation, an order change, or an order cancellation from the retailer. It may happen that no response is received at all. If no response is received after 48 hours, or if an order cancellation is received, the supplier will cancel the order. If an order confirmation is received within 48h, the supplier will process the order normally. If an order change is received within 48h, the supplier will update the order and ask again the retailer for confirmation. The retailer is allowed to change an order at most three times. Afterwards, the supplier will automatically cancel the order.

Instructions:

- Model the relevant resources (pools, lanes)
- Modeling the relevant business objects (data object, data store) is not required.