Sketch2BPMN: Automatic Recognition of Hand-drawn BPMN Models

Bernhard Schäfer[1,2], Han van der Aa[2], Henrik Leopold[3,4], and Heiner Stuckenschmidt[2]

[1] Intelligent Robotic Process Automation, SAP SE, Walldorf, Germany
[2] Data and Web Science Group, University of Mannheim, Mannheim, Germany
{bernhard|han|heiner}@informatik.uni-mannheim.de
[3] Kühne Logistics University, Hamburg, Germany
henrik.leopold@the-klu.org
[4] Hasso Plattner Institute, University of Potsdam, Potsdam, Germany

Abstract. Despite the widespread availability of process modeling tools, the first version of a process model is often drawn by hand on a piece of paper or whiteboard, especially when several people are involved in its elicitation. Though this has been found to be beneficial for the modeling task itself, it also creates the need to manually convert hand-drawn models afterward, such that they can be further used in a modeling tool. This manual transformation is associated with considerable time and effort and, furthermore, creates undesirable friction in the modeling workflow. In this paper, we alleviate this problem by presenting a technique that can automatically recognize and convert a sketch process model into a digital BPMN model. A key driver and contribution of our work is the creation of a publicly available dataset consisting of 502 manually annotated, hand-drawn BPMN models, covering 25 different BPMN elements. Based on this data set, we have established a neural network-based recognition technique that can reliably recognize and transform hand-drawn BPMN models. Our evaluation shows that our technique considerably outperforms available baselines and, therefore, provides a valuable basis to smoothen the modeling process.

Keywords: process modeling, sketch recognition, hand-drawn process models

1 Introduction

In many organizations, business process modeling has become an integral activity to document and analyze business processes, as well as to collect requirements in software development projects [1, 2]. This importance has led to the availability of a large number of specialized process modeling tools, which support modelers while creating, checking, and maintaining business process models and even entire collections thereof. Although these tools can generally be considered indispensable, it is important to note that the development of most process models does not start with software. For example, pen and paper is a suitable approach for an initial sketch [12, p.85]. In collaborative settings, process models are typically drawn on whiteboard or brown paper, which stimulates the engagement of process stakeholders [9, 12].

However, starting with a hand-drawn model also introduces the need to subsequently convert it into a digital counterpart using a modeling tool [4]. Since this task currently needs to be performed manually, this transformation step is associated with considerable time and effort and, furthermore, creates undesirable interruptions and friction in the modeling process. Some existing methods for collaborative modeling have tried to circumvent this transformation problem by providing tool support for collaborative modeling [3, 7]. Nonetheless, these tools do not allow users to freely sketch processes, but require them to stick to predefined constructs and functionality, which actually mitigates the benefits associated with the use of hand-drawn models.

Therefore, recognizing the importance and benefits of hand-drawn process models, as well as the effort involved in their manual transformation, we use this paper to propose an approach that automates this step by transforming sketch process models into digital BPMN models. By expanding state-of-the-art work from the area of diagram recognition [20], our neural network-based approach, `Sketch2BPMN`, takes a hand-drawn BPMN (Business Process Model and Notation) model as input and produces a respective BPMN XML file, suitable for process modeling tools. Aside from this, a core contribution of our work is the introduction of the publicly available `hdBPMN` dataset, consisting of 502 manually annotated, hand-drawn BPMN models, covering 25 kinds of BPMN elements and varying considerably in their characteristics. Our experiments conducted on this dataset demonstrate the high accuracy achieved by our approach.

In the remainder, Section 2 illustrates the challenges that come with the recognition of hand-drawn BPMN models. Section 3 describes the `Sketch2BPMN` approach. Section 4 introduces our `hdBPMN` dataset and Section 5 the evaluation in which it is used. Finally, Section 6 reflects on related work before Section 7 concludes the paper.

2 Problem Illustration

To motivate the importance of our work, we illustrate the challenges associated with the automated recognition of hand-drawn process models using a real-world example from our `hdBPMN` dataset. Fig. 1 depicts a BPMN model concerned with a claims-handling process. Although the depicted model is syntactically sound and correct, the figure illustrates several challenges that must be overcome by an automated recognition approach. Issue (1) shows that model elements may be drawn *incompletely*, such as the *Costumer* (sic) pool. Issue (2) points to one of the many instances where an edge is *not properly connected* to its source and/or target shapes. Other issues relate to overlaps between model components, such as *text that interrupts a line* (3) or lines that *cross each other* (4). We also observe remains of corrected mistakes, i.e., *crossed out parts* (5) and the use of curved rather than straight lines (6).

Aside from the drawing itself, additional difficulties can be attributed to the paper that has been used and the way it was digitized. First, the use of *grid paper* adds numerous additional lines to the image that may obscure the distinction between lines of the paper and those drawn as part of a model element (e.g., a *resource pool* or *lane*). Second, the fact that the image has been captured with a camera instead of a scanner introduces additional quality issues, such as curved paper, blur, and perspective distortion.

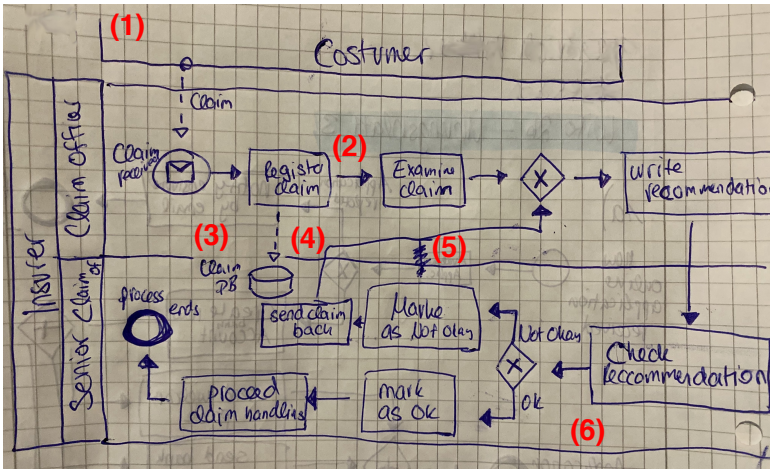


Fig. 1: Example of a hand-drawn BPMN model with highlighted issues

In summary, these issues make it harder for an automated approach to properly interpret a hand-drawn BPMN model. Although the above only relates to a single example, the spectrum of hand-drawn BPMN models is even more complex, due to the combination of various drawing styles, paper types, pens, and digitization methods. As a result, the dataset we introduce in this paper contains a remarkably high degree of diversity, which a successful BPMN-recognition approach must be able to deal with.

3 Approach

This section introduces `Sketch2BPMN`, our automated approach for recognizing a hand-drawn BPMN model from an image. As visualized in Fig. 2, `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.

3.1 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

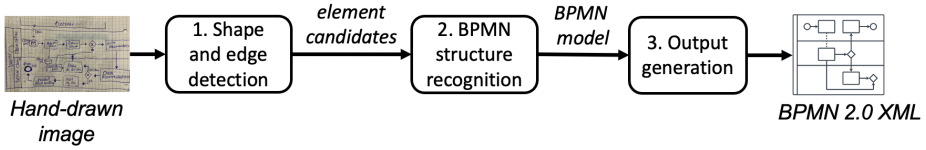


Fig. 2: `Sketch2BPMN`: overview of the main steps

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. Fig. 3 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* [20], 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 [20] is a method we previously defined to recognize flowcharts. At its core, Arrow R-CNN builds on Faster R-CNN [19], a popular deep neural network approach for object detection. Arrow R-CNN detects objects in an image through the aforementioned bounding boxes, depicted in Fig. 3. For each bounding box, Arrow R-CNN assigns a predicted class (from a set of predefined classes) and a likelihood between 0 and 1. While such bounding-box detection is suitable for shapes, the proper detection of edges is more complex. Specifically, standard object detection, such as provided by Faster R-CNN, cannot be used to identify the source and target shapes of an edge. Therefore, Arrow R-CNN extends Faster R-CNN with an arrow keypoint detection network that predicts the arrowhead and tail of an edge. This means that for edges, Arrow R-CNN provides a (predicted) start and end point along with the bounding box. During training, the keypoint detector loss is averaged over a sample of edges and then integrated into the joint Faster R-CNN loss term. This allows Arrow R-CNN to train all components simultaneously, and it also reduces the risk of overfitting on small datasets, which makes Arrow R-CNN especially suited for small datasets. Given its general applicability for the recognition of arrow-connected diagrams, we

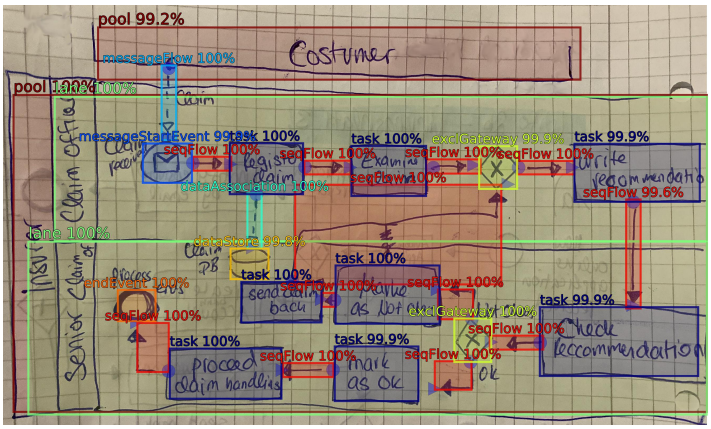


Fig. 3: *Step 1*: Candidate shapes and edges have been classified and localized through bounding boxes, in addition edges have predicted arrow head (▷) and tail (○) keypoints.

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 that support improved identification of BPMN models in our scenario.

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 (see issue 2 in Fig. 1). 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 Fig. 4. 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 particular, 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 Fig. 4, 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.

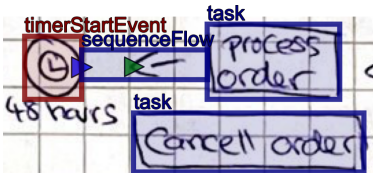


Fig. 4: Improved keypoint detection due to annotating edge intersection (blue ▷) instead of drawn arrow head (green ▷).

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 [8]. They have been shown to be particularly valuable when training an approach on a dataset with only a few hundred images [20], 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.

3.2 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 Fig. 5 for two categories of timer events.

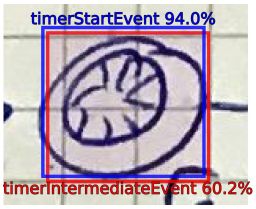


Fig. 5: Duplicate shape candidates

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 Fig. 5, 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 $e.c =$ *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. Fig. 6 depicts the outcome of this step for the

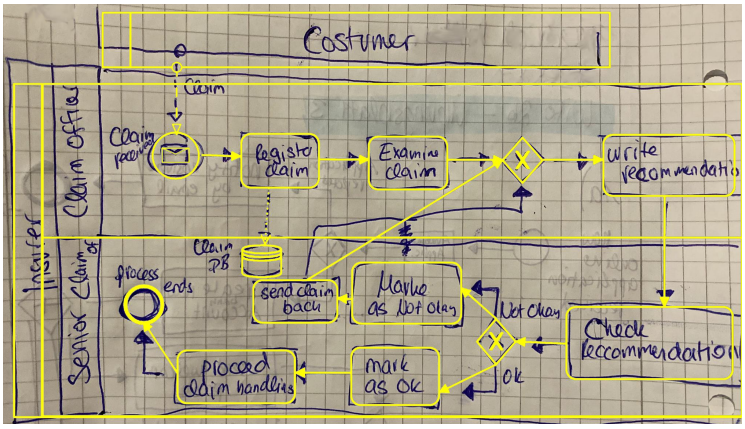


Fig. 6: *Recognized BPMN Model*: the recognized BPMN model has been converted to an image and overlaid over the hand-drawn sketch. The entire process from the input image to the final BPMN model is automated.

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.

3.3 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.

4 Dataset

This section discusses the collection, annotation, characteristics, and splits of `hdBPMN`, the dataset we established for our work. The original and annotated images are publicly available at: https://github.com/dwslab/hdBPMN.

4.1 Collection Procedure

We collected 502 images of hand-drawn BPMN models from 107 participants, all students at the University of Mannheim. Each image corresponds to a solution that was

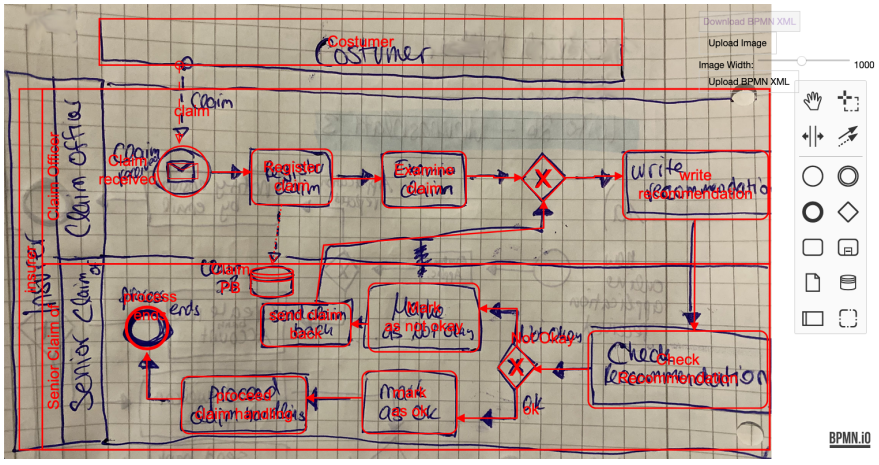


Fig. 7: Example of an annotated hand-drawn model in the *BPMN image annotator*. Shapes are sized and positioned to match their hand-drawn counterparts, while edges are modeled using waypoints to resemble the handwritten arrows as much as possible.

submitted by a student for a graded assignment in an exercise sheet or exam. The obtained models stem from 10 modeling tasks, 9 of which involved the establishment of a BPMN model on the basis of a textual process description, while the other involved the conversion of a Petri net into a BPMN model. Students were asked to hand-draw their models on paper and afterwards embed the scan or photo of their drawing in the submitted exercise sheet or exam PDF, with the only constraint that that the models should be readable.[5]

For each received submission, we used the `pdfimages` command-line utility to extract the images from the PDFs. We split images manually into multiples ones when they spanned different modeling tasks. Note that we deliberately did not further crop images, occasionally resulting in the inclusion of background objects. Finally, we used image editing software to conceal any personal details (e.g., names and student IDs). The resulting images were assigned filenames that follow a `taskID_participantID` convention, allowing one to identify images contributed by the same participant.

4.2 Annotation

To train and evaluate our BPMN recognition approach, we annotated the hand-drawn shapes and edges in each image. To this end, we developed a BPMN image annotation tool based on the open-source `bpmn-js`[6] BPMN viewer and editor, which we have made publicly available.[7] Fig. 7 depicts the annotation tool in action. Note that in order to allow us to also annotate images that contain modeling errors, we allow our

[5] See also the public repository for the modeling tasks and provided instructions.
[6] https://github.com/bpmn-io/bpmn-js
[7] https://github.com/dwslab/bpmn-image-annotator

annotation tool to violate certain correctness rules enforced by `bpmn-js`, e.g., an *end event* with an outgoing *sequence flow*. Upon completion, the annotation is exported as a *BPMN 2.0 XML* file. Since the image resolution during annotation is exported as a comment into this file, the location and size of each shape and edge in the BPMN model can be linked back to the location and size in the image.

4.3 Dataset Characteristics

The 502 annotated images contain more than 20,000 annotated elements. As shown in Table 1, the models in the dataset are highly expressive, spanning 25 types of BPMN elements, including 4 types of activity shapes, 9 types of events, 4 types of gateways, and 4 types of edges. 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, number of types covered). The models resulting from the ten different modeling tasks have up to 15 activities and 13 events, and between 0 and 8 gateways, 0 and 8 resources, and 0 and 11 data elements. Some tasks result in simpler models (e.g., task1: 11.0 shapes and 11.9 edges on average) and others in more complex ones (e.g., task3: 26.7 shapes and 28.2 edges). Note that the running example depicted in, e.g., Fig. 1, represents a task of intermediate complexity.

Table 1: BPMN elements included in the 502 annotated images

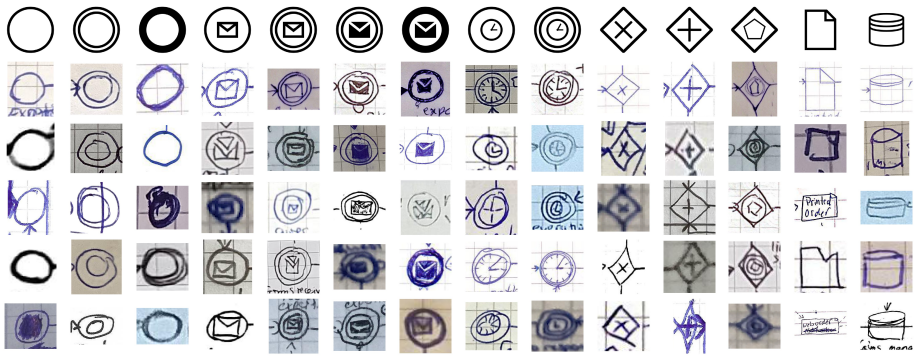
Type	Group	Elements and their frequencies
Shape	Activities	task (2,906), subprocess (collapsed) (88), subprocess (expanded) (4), call activity (10)
	Events	start (373), intermediate throw (16), end (587), message start (248), message intermediate catch (220), message intermediate throw (139), message end (52), timer start (71), timer intermediate catch (72)
	Gateways	exclusive (822), parallel (651) inclusive (1), event-based (67)
	Resources	pool (657), lane (523)
	Data elements	data object (707), data store (141)
Edge	Flow elements	sequence flow (6,456), message flow (852), data association (1,395) annotation association (65)
Text	Annotations	text annotation (69)
	Element labels	label (2,971)

Aside from the complexity of a particular process, the recognition difficulty of individual images is affected by various other aspects:

- *Paper type.* The type of paper on which a model is drawn influences the amount of noise that is present in an image, since, e.g., lined or squared paper introduces additional lines that may be hard to distinguish from model-related content in an image. The vast majority of images in our dataset is drawn on squared (about 330) and blank paper (about 140), whereas the remaining used lined or dotted paper.
- *Drawing implement.* The type of drawing implement employed affects the thickness, clarity, and consistency of lines in a drawing. Our dataset primarily contains

models drawn by pen (about 430), whereas the remainder were drawn using pencils, fineliners, or a mixture (e.g., pencil to draw shapes and pen for text).
- *Model quality issues.* The hand-drawn models have various kinds and degrees of quality issues, in line with those discussed in Section 2, such as models with incomplete, crossing, and curved lines, or with crossed-out elements. Furthermore, the models differ greatly in the spacing that is used between shapes, e.g., some models are spacious, whereas others pack various shapes into a small area.
- *Image capturing issues.* Camera-based image capturing introduces a series of quality issues [11]. This includes images that are rotated or blurry, as well as those that include content beyond the paper or actually cut part of it off (e.g., Fig. 1).

Overall, the 502 images in the publicly-available `hdBPMN` dataset thus depict BPMN models that span a broad range of BPMN elements and have a high degree of diversity, caused by variation in terms of the employed paper, drawing implement, and image capturing method. Fig. 8 visualizes this diversity by showcasing some of the different manners in which various kinds of shapes were drawn.



(a) (b) (c) (d) (e) (f) (g) (h) (i) (j) (k) (l) (m) (n)

Fig. 8: Examples of hand-drawn *events* (start 8a, intermediate 8b, end 8c, message start 8d, message intermediate catch 8e, message intermediate throw 8f, message end 8g, timer start 8h, timer intermediate 8i), *gateways* (exclusive 8j, parallel 8k, event-based 8l) and *data elements* (data object 8m, data store 8n).

4.4 Dataset Splits

Following related hand-drawn diagram datasets [6,14], we split up the dataset into publicly available training, validation, and test parts. Each participant in the dataset contributed between one and nine 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 308/102/92 diagrams from 65/21/21 participants, respectively.

5 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.

5.1 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* [23] object detection framework. To operationalize our extended augmentations, we use the *Albumentations* library [8]. 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 [20].

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 Section 4.2), referred to as the *ground truth*. To quantify the performance, we follow related work in diagram recognition [5, 6, 22] 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 [22], 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.[8]

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 3.1. Configuration C2 reflects

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

our full-fledged approach, including the proposed BPMN-specific processing components. Note that we employ the same shape-disambiguation procedure (proposed in Section 3.2) for all four systems, in order to ensure a fair comparison in terms of recall.

5.2 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 2 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.

Table 2: Overall approach results

Configuration	Shape		Edge	
	Micro F_1	**Macro F_1**	**Micro F_1**	**Macro F_1**
BL1: Faster R-CNN [19]	92.7	77.5	—	—
BL2: Arrow R-CNN [20]	93.2	80.8	85.5	76.0
C1: Arrow R-CNN + Extended Augm. (EA)	**95.7**	**86.2**	90.0	84.8
C2: Arrow R-CNN + EA + BPMN processing	**95.7**	**86.2**	**91.8**	**87.4**

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 3 provides detailed insights into the performance of our approach (with configuration C2,) by depicting the results obtained per shape 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 post-hoc 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,

Table 3: 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 Intermediate Catch Event	90.9	93.0	92.0	43
	Message Intermediate 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	1216
	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	3307
	Micro avg.	94.8	92.7	93.7	3307

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 diversity of shapes in our dataset, as highlighted in Fig. 8, 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 3 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 only highly interesting to consider the different role 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 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 [12, p.177], whereas they also often are connected to a 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 Related Work

Our research mainly relates to the area of *hand-drawn diagram recognition*, which has its roots in the graphics recognition area, where several techniques for flowchart [5, 16, 21, 22] and finite automata [6] recognition have been proposed. Most of these are so-called *online* recognition techniques, which depend on data about the order and extent of individual strokes, i.e., the drawing sequence. This means they can only be applied to diagrams drawn on digital devices, like tablets. *Offline* recognition, which we target in this paper, is more complex, because such sequence information is not available and individual strokes cannot be reliably reconstructed from camera-based images of pen and paper drawings. Outside graphics recognition, techniques for hand-drawn diagram detection have been mainly adopted in the area of software engineering [10, 15]. More recently, also the first recognition techniques for hand-drawn process models were proposed [17, 24]. However, these techniques only target the recognition of shapes, which means that they do not result in complete models.

Our research also relates to work on *collaborative process modeling*, which mainly investigates and supports modeling efforts that involve several people. Works on the former aspect typically use empirical methods to develop a better understanding of how certain factors, such as collaborative tools or participant interaction, affect the modeling outcome [13, 18]. Research on the latter aspect provides methods and tools to support collaborative modeling efforts. These include works that enable collaborative process modeling through design storyboards [3] and virtual worlds [7]. Our work also supports collaborative process modeling, since it does not require modelers to follow a particular procedure or use a specific tool to obtain an initial model. Instead, modelers can freely draw a model, which our approach can then transform into a digital counterpart.

7 Conclusion

In this paper, we proposed `Sketch2BPMN`, a neural network-based approach for automatically recognizing hand-drawn BPMN models from images. Moreover, we introduced `hdBPMN`, a publicly available dataset consisting of 502 manually annotated, hand-drawn BPMN models, covering 25 different BPMN elements. By using `hdBPMN` to train, validate, and test our approach, we demonstrated that `Sketch2BPMN` considerably outperforms available baselines from the area of object and flowchart recognition and, therefore, provides a valuable basis for the automated conversion of hand-drawn BPMN diagrams. We would like to highlight that this paper conceptually targets the recognition of BPMN shapes and edges and, hence, abstracts from recognizing the handwritten textual labels in of BPMN models. However, this can be achieved using off-the-shelf handwriting recognition solutions. In future work, we plan to further improve the practical value of our approach by 1) matching the recognized handwritten textual labels to the respective shapes and edges, and 2) by recognizing the intended rather than the actually drawn model in order to directly fix certain drawing errors.

References

1. Aagesen, G., Krogstie, J.: Analysis and design of business processes using BPMN. In: Handbook on Business Process Management 1, pp. 213–235. Springer (2010)
2. Allweyer, T.: BPMN 2.0: Introduction to the standard for business process modeling. BoD–Books on Demand (2016)
3. Antunes, P., Simões, D., Carriço, L., Pino, J.A.: An end-user approach to business process modeling. Journal of Network and Computer Applications **36**(6), 1466–1479 (2013)
4. Bartelt, C., Vogel, M., Warnecke, T.: Collaborative creativity: From hand drawn sketches to formal domain specific models and back again. In: MoRoCo@ ECSCW. pp. 25–32 (2013)
5. Bresler, M., Průša, D., Hlaváč, V.: Recognizing Off-Line Flowcharts by Reconstructing Strokes and Using On-Line Recognition Techniques. In: (ICFHR). pp. 48–53 (2016)
6. Bresler, M., Průša, D., Hlaváč, V.: Online recognition of sketched arrow-connected diagrams. International Journal on Document Analysis and Recognition **19**(3), 253–267 (2016)
7. Brown, R., Recker, J., West, S.: Using virtual worlds for collaborative business process modeling. Business Process Management Journal (2011)
8. Buslaev, A., Iglovikov, V.I., Khvedchenya, E., Parinov, A., Druzhinin, M., Kalinin, A.A.: Albumentations: Fast and Flexible Image Augmentations. Information **11**(2), 125 (2020)
9. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J.: Let's Go to the Whiteboard: How and Why Software Developers Use Drawings. In: SIGCHI. pp. 557–566 (2007)
10. Damm, C.H., Hansen, K.M., Thomsen, M.: Tool support for cooperative object-oriented design: gesture based modelling on an electronic whiteboard. In: SIGCHI. pp. 518–525 (2000)
11. Doermann, D., Jian Liang, Huiping Li: Progress in camera-based document image analysis. In: ICDAR. pp. 606–616 vol.1 (2003)
12. Dumas, M., Rosa, M.L., Mendling, J., Reijers, H.A.: Fundamentals of Business Process Management. Springer, 2 edn. (2018)
13. Forster, S., Pinggera, J., Weber, B.: Toward an understanding of the collaborative process of process modeling. In: CAiSE Forum. pp. 98–105 (2013)
14. Gervais, P., Deselaers, T., Aksan, E., Hilliges, O.: The DIDI dataset: Digital Ink Diagram data. arXiv:2002.09303 [cs] (2020)
15. Hammond, T., Davis, R.: Tahuti: A geometrical sketch recognition system for uml class diagrams. In: ACM SIGGRAPH 2006 Courses, pp. 25–es (2006)
16. Julca-Aguilar, F., Mouchère, H., Viard-Gaudin, C., Hirata, N.S.T.: A general framework for the recognition of online handwritten graphics. IJDAR (2020)
17. Polančič, G., Jagečić, S., Kous, K.: An empirical investigation of the effectiveness of optical recognition of hand-drawn business process elements by applying machine learning. IEEE Access (2020)
18. Recker, J., Mendling, J., Hahn, C.: How collaborative technology supports cognitive processes in collaborative process modeling: A capabilities-gains-outcome model. Information Systems **38**(8), 1031–1045 (2013)
19. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In: NeurIPS. pp. 91–99 (2015)
20. Schäfer, B., Keuper, M., Stuckenschmidt, H.: Arrow R-CNN for handwritten diagram recognition. International Journal on Document Analysis and Recognition (IJDAR) (Feb 2021)
21. Schäfer, B., Stuckenschmidt, H.: Arrow R-CNN for Flowchart Recognition. In: International Conf. on Document Analysis and Recognition Workshops (ICDARW) (2019)
22. Wu, J., Wang, C., Zhang, L., Rui, Y.: Offline Sketch Parsing via Shapeness Estimation. In: IJCAI (2015)
23. Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. https://github.com/facebookresearch/detectron2 (2019)
24. Zapp, M., Fettke, P., Loos, P.: Towards a Software Prototype Supporting Automatic Recognition of Sketched Business Process Models. Wirtschaftsinformatik 2017 (2017)