International Journal on Document Analysis and Recognition (IJDAR) (2021) 24:3–17
https://doi.org/10.1007/s10032-020-00361-1

SPECIAL ISSUE PAPER

Arrow R-CNN for handwritten diagram recognition

Bernhard Schäfer[1,2] · **Margret Keuper**[2] · **Heiner Stuckenschmidt**[2]

Received: 6 February 2020 / Revised: 19 October 2020 / Accepted: 8 December 2020 / Published online: 2 February 2021

Abstract
We address the problem of offline handwritten diagram recognition. Recently, it has been shown that diagram symbols can be directly recognized with deep learning object detectors. However, object detectors are not able to recognize the diagram structure. We propose Arrow R-CNN, the first deep learning system for joint symbol and structure recognition in handwritten diagrams. Arrow R-CNN extends the Faster R-CNN object detector with an arrow head and tail keypoint predictor and a diagram-aware postprocessing method. We propose a network architecture and data augmentation methods targeted at small diagram datasets. Our diagram-aware postprocessing method addresses the insufficiencies of standard Faster R-CNN postprocessing. It reconstructs a diagram from a set of symbol detections and arrow keypoints. Arrow R-CNN improves state-of-the-art substantially: on a scanned flowchart dataset, we increase the rate of recognized diagrams from 37.7 to 78.6%.

Keywords Diagram recognition · Offline recognition · Object detection · Graphics recognition · Symbol recognition · Flowchart

1 Introduction

Graphical modeling languages are a long-used and intuitive device to visualize algorithms, business process models, and software systems. There are various formalized modeling notations, including flowchart, UML use case, and event-driven process chain diagrams. Initial diagrams are typically sketched on non-digital devices such as whiteboard or paper [10]. If a user decides they want to continue working on the sketch in a modeling software, they are required to manually recreate the diagram in its entirety. Research in handwritten diagram recognition addresses this gap, by providing an automated method that converts a scan or photograph of a diagram into a structured model. It is thus concerned with two main tasks: (1) the local recognition and localization of symbols and (2) the recognition of the global structure.

✉ Bernhard Schäfer
bernhard.schaefer@sap.com

Margret Keuper
keuper@uni-mannheim.de

Heiner Stuckenschmidt
heiner@informatik.uni-mannheim.de

1 Intelligent Robotic Process Automation, SAP SE, Walldorf, Germany

2 Data and Web Science Group, University of Mannheim, Mannheim, Germany

Handwritten diagram recognition methods can be categorized into two approaches: online and offline recognition [7]. In online recognition, the diagrams are drawn with an ink input device such as a tablet. This input device captures the drawing as a temporal sequence of strokes. Online diagram recognition has received a lot of attention in research, especially in the area of flowcharts [1–6,9,14,18,36,37,40]. Yet, those approaches are of limited applicability if the original stroke data are not available (e.g., hand-drawn diagrams on paper). While offline recognition directly allows to tackle this more general scenario, it has attracted much less attention in the past. Most offline approaches rely on traditional image processing methods to reconstruct the strokes of a diagram, and use feature engineering to derive a set of distinctive stroke features [7,24]. The revival of deep convolutional neural networks (CNNs) has caused a paradigm shift from "feature engineering" to "feature learning". For detecting object instances in an image, two-stage object detectors popularized by Faster R-CNN [26] are state-of-the-art. In [15] it was demonstrated that Faster R-CNN can be effectively used to recognize the symbols in a flowchart image. Even though the utilized flowchart dataset has only 200 training images, the evaluation shows very good results for recognizing node shapes. The model mostly struggles with recognizing arrows and text phrases due to their varying form and size. We agree with their motivation and thus propose an offline handwrit-

ten diagram recognition approach which builds upon Faster R-CNN for symbol recognition.

This work focuses on handwritten diagrams, where each diagram contains node symbols of varying shapes, arrows, and optionally text phrases. Arrow recognition methods are also used to identify regions of interests in medical images and to interpret mechanical drawings and circuit diagrams [27–29]. While the recognition of computer-generated arrows in mentioned examples is important, this work focuses on handwritten diagrams, where each arrow connects two nodes, and each text phrase annotates either a node or an arrow. Although this structure is simple, it is sufficiently powerful to describe graphical modeling languages from various domains. We follow the terminology in [6] and use the term arrow-connected diagram to describe this class of diagrams. Structure recognition in arrow-connected diagrams is concerned with specifying the nodes each arrow joins and the edge direction. While an object detector can classify and localize the symbols of a diagram through bounding boxes, the bounding box information is insufficient for structure recognition. We show that we can leverage arrow keypoint information for diagram structure recognition.

In this paper, we propose a system for recognizing arrow-connected diagrams. Our contributions are the following:

- We demonstrate how a Faster R-CNN object detector can be extended with a lightweight arrow keypoint predictor for diagram structure recognition.
- We identify data augmentation methods for diagrams and show that they vastly improve object detection results. We also propose a method to augment diagrams with words from an external dataset, to further reduce the confusion between arrow and text symbols.
- We propose a postprocessing pipeline to form a final diagram from object and keypoint candidates.
- We evaluate our system on four datasets from the flowchart and finite automata domain.

Figure 1 provides an overview of our method.

The remainder of the paper is organized as follows. Section 2 surveys related work in diagram recognition and keypoint estimation. Section 3 describes the Arrow R-CNN network and its training procedure. Section 4 proposes augmentation and postprocessing methods designed for diagrams. Section 5 contains experimental results and describes the datasets used for evaluation. Section 6 presents conclusions and potential future work.

2 Related work

Symbol recognition plays an important role in various applications and has a rich literature in graphics recognition [30,31]. Our work focusses on diagram recognition using an arrow keypoint detector. Thus, in the following we separately discuss related work on (1) handwritten diagram recognition and (2) generic keypoint detection with neural networks.

2.1 Handwritten diagram recognition

While our work is focused on offline handwritten diagram recognition, there is an overlap with online recognition with respect to the utilized methods and datasets. From a method perspective, online recognition systems that do not strictly require temporal stroke information can be adjusted for offline recognition. To this end, a stroke reconstruction preprocessing step can extract a set of strokes from a raster image. From a dataset perspective, online datasets can also be used to evaluate offline systems by plotting the strokes as an image. This is common practice, due to the lack of public datasets that are offline by nature. In the following, we discuss related offline work and also include online works that introduce a new dataset or contain an offline extension.

In 2011, Awal et al. [1] published the FC_A online handwritten flowchart dataset. The dataset is publicly available, and its size has been increased to 419 flowcharts after the publication date. Following the release, several methods for online flowchart recognition were proposed [2–6,9,18,36,37,40]. Wu et al. [38] is the first work that uses FC_A for offline recognition. [38] use a three-stage recognition pipeline, including a shapeness estimation algorithm to figure out if a stroke grouping has a regular appearance. The pipeline is evaluated using the ground truth strokes without temporal information and achieves 83.2% symbol recognition accuracy. Since the evaluation does not attribute for stroke reconstruction errors, the result is not comparable with an image-based recognition system.

Bresler, Průša, and Hlaváč published a series of works on online diagram recognition [2–6], and also introduced two online diagram datasets in the domains of finite automata (FA) and flowcharts (FC_B). The latest online system in [6] is a pipeline with text/non-text separation, symbol segmentation, symbol classification and structural analysis as its core parts. An offline extension to this online system is proposed in [7] and uses a stroke reconstruction preprocessing step. For the evaluation, two offline flowchart datasets based on the existing FC_B dataset were introduced. One of those datasets contains scans of a printed thicker stroke visualization containing real noise, which we refer to as $\texttt{FC\_B}_{\texttt{scan}}$. The adapted recognizer was also tested on the unordered FC_A strokes and achieved 84.2% symbol recognition recall.

As the proposed approach, Julca-Aguilar and Hirata [15] train a Faster R-CNN [26] object detector to recognize symbols in the FC_A dataset. To this end, they do transfer learning from models pre-trained on MS COCO, a large dataset of natural images depicting everyday scenes [20]. The detector

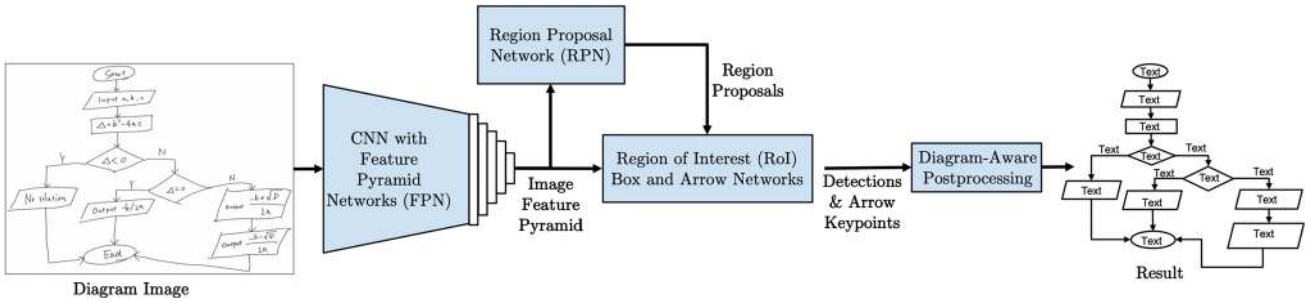


Fig. 1 Overview of our method: the CNN backbone network computes the image feature pyramid from an input image. The region proposal network (RPN) uses the feature pyramid to compute a set of region proposals. The region of interest (RoI) box network (Fig. 2) classifies each proposal and predicts a refined bounding box. For detected arrows, our RoI arrow network additionally predicts arrow head and tail keypoints. The postprocessing component (Fig. 7 ff.) uses the detected symbols and arrow keypoints to construct the final diagram

achieves a mAP@0.5 of 97.7%, where mAP@0.5 corresponds to mean average precision (mAP) at a ground truth bounding box overlap of at least 50%. Due to differing evaluation metrics, this result is not comparable to [7].

Gervais et al. published the online handwritten flowchart dataset DIDI in 2020 [11]. It 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 handwritten diagram has been collected by showing a flowchart image to the user who was then asked to draw over it. The flowchart images were rendered using GraphViz based on randomly-generated dot files. Unlike other online datasets, the handwritten diagrams are not annotated on stroke level. However, the provided dot files contain information about the rendered diagram, such as the position and size of each node.

The present paper builds upon an earlier workshop paper [32]. In [32], we proposed a deep learning system based on Faster R-CNN for recognizing the symbols and structure of flowcharts. We identified data augmentation methods for flowchart recognition and showed that they vastly improve object detection results on the FC_A dataset. We also demonstrated how a Faster R-CNN object detector can be extended with a lightweight arrow keypoint predictor for flowchart structure recognition. For the present paper, we have consolidated the overall technique and provide a more broadly applicable system for recognizing arrow-connected diagrams. Specifically, we improve on the data augmentation and arrow proposal sampling methods used during network training. Moreover, we propose a postprocessing pipeline to form a final diagram from object and keypoint candidates. We evaluate our system on four datasets, adding a finite automata, a flowchart dataset of scanned images, and a large-scale flowchart dataset, and provide profound insights into accurate arrow and text phrase detection, which we consider the main challenge for recognizing arrow-connected diagrams.

2.2 Keypoint detection

Keypoint detection methods are typically used for tasks such as pose estimation in natural images, i.e. to estimate facial or human body keypoints. For human pose estimation (HPE), where the task is to predict human body keypoint locations such as elbow and wrist, there are two mainstream methods: directly regressing the position of keypoints [34,35], and estimating keypoint heatmaps [23,39].

In the heatmap approach, the keypoints are chosen as the locations with the highest heat values. Over the last years, the heatmap approach has been used in all state-of-the-art systems evaluated on the MS COCO dataset keypoint detection task [20]. The HPE systems typically tackle multi-person keypoint estimation in a top-down process. In the first stage, individual 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 feed into a dedicated single-person pose estimation network. While methods used in HPE can serve as inspiration for an arrow keypoint detector, the two tasks differ substantially. In the COCO dataset, persons instances 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 [23] encourage the learned person features to be reevaluated in a larger global context. They achieve this with CNN architectures that involve successive steps of pooling and upsampling, with additional skip connections. In arrow keypoint estimation, most arrow heads and tails are clearly visible. Further, arrow head and tail keypoints are often the outermost arrow pixel in one spatial direction and thus define one border of the bounding box. Therefore, there is a task overlap between arrow keypoint detection and bounding box detection. Thus, in a multi-task learning setup, it might be beneficial to learn those tasks together with shared features.

In conclusion, although HPE is dominated by heatmap methods, we opt for a keypoint regression method that shares its features with the object detector network. We find that this approach is effective in the diagram domain, where most datasets are small and keypoints are typically located at the bounding box border.

3 Arrow R-CNN

Arrow R-CNN is based on the Faster R-CNN [26] object detector. Faster R-CNN is the successor of R-CNN [12], which has popularized a two-stage approach in object detection. The first stage generates a set of class-agnostic region proposals or region of interests (RoI), where each RoI is defined by a bounding box location and an objectness score. The second stage then classifies each RoI and predicts a refined bounding box location. From a high-level perspective, Faster R-CNN consists of three sub-networks:

1. CNN backbone network
2. Region proposal network (RPN)
3. RoI box network

The CNN backbone network is used to extract a featurized representation of the entire image. This feature map has a lower spatial resolution $w \times h$, but a much higher number of channels c than the original image. The RPN uses the feature map to compute a large set of RoIs. The RoI box network classifies each RoI as one of the object classes or as background, and refines its bounding box location. It uses RoI pooling, a pooling mechanism to extract a fixed-sized $7 \times 7 \times 512$ feature map for each RoI proposal. RoI pooling uses the proposal bounding box to extract the relevant spatial part of the backbone feature map and then applies pooling to reduce it to a fixed-size representation. The box network processes each RoI feature map with intermediate fully connected layers, before it classifies each RoI and predicts its refined bounding box.

One of the limitations of Faster R-CNN is that it has difficulties with datasets where objects have a large-scale variance. [19] addresses this issue by incorporating feature pyramid networks (FPNs) into Faster R-CNN. In this extension, the backbone network generates a pyramid of feature maps 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. During RoI pooling, an RoI is assigned to a feature pyramid level based on its bounding box dimension. This has the advantage that if the RoI's scale becomes smaller, it can be a mapped into a finer resolution level. Our initial experiments showed that we get consistently better results with the FPN extension. Thus, we use it in all our experiments.

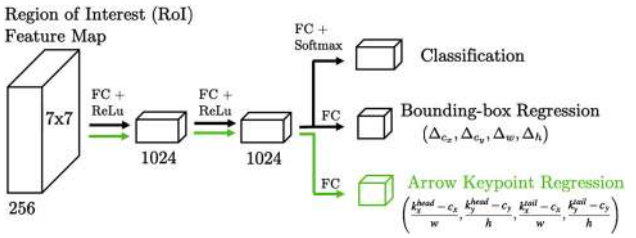


Fig. 2 *Arrow R-CNN RoI networks* The box and arrow modules process each RoI feature map through fully connected layers (FC) with ReLu activation functions. The box network predicts a class and refined bounding box for all proposals. The arrow network predicts a 4-d arrow keypoint vector for each arrow proposal (training) or arrow detection (inference)

In the next sections, we outline the Arrow R-CNN network architecture (Sect. 3.1), explain how we train the entire network (Sect. 3.2), and detail how Arrow R-CNN computes the detected symbols and keypoints from a diagram image during inference time (Sect. 3.3).

3.1 Network architecture

For predicting the keypoints at the arrows head and tail, we add a parallel RoI arrow network to the existing RoI box network. Figure 2 shows both Arrow R-CNN RoI networks. The arrow network reuses the fully connected feature extraction layers from the box network and regresses both 2-d arrow keypoints as a 4-d vector $(\mathbf{k}^{\text{head}\top}, \mathbf{k}^{\text{tail}\top})^\top$ from the extracted 1024-d arrow feature representation. Figure 2 illustrates this series of computation with a green path from the RoI feature map to the arrow keypoint regressor. In theory, we could directly use the absolute arrow keypoint pixel coordinates as regression targets. However, this would require the RoI feature map 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 proposal bounding box. Suppose we have a proposal 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 $\mathbf{k} = (k_x, k_y)^\top$ assigned to a proposal with bounding box $\mathbf{b}$, we define bounding box normalized keypoints as

$$\bar{\mathbf{k}} = (\bar{k}_x, \bar{k}_y)^\top = \left(\frac{k_x - c_x}{w}, \frac{k_y - c_y}{h} \right)^\top . \quad (1)$$

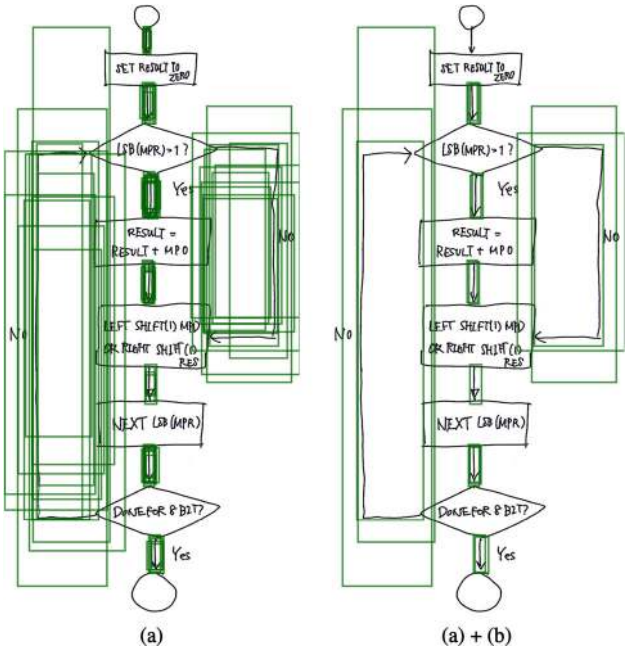


Fig. 3 Arrow proposal example: the left image shows the 72 proposals with at least 70% IoU to a ground truth arrow, the right image shows the 19 proposals that additionally have both arrow keypoints located within their proposal box

Thus, $\bar{k}_x$ and $\bar{k}_y$ are within the range $[-1, 1]$, for keypoints $\mathbf{k}$ contained in bounding box $\mathbf{b}$. 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 relative 2-d coordinates of the two keypoints per arrow.

3.2 Training

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 a bounding box overlap of at least 70%, NMS iteratively removes the proposal with the lower objectness score. The bounding box overlap is commonly referred to as intersection over union (IoU). For training the RoI box network, Faster R-CNN considers the top 2000 proposals ranked by their objectness score [19]. For training our arrow network, we use a subset of the 2000 proposals. Concretely, we define an arrow proposal as a proposal that fulfills two criteria:

(a) at least 70% IoU to a ground truth arrow
(b) both arrow keypoints are located within the proposal box

Figure 3 shows exemplary arrow proposals that fulfill either criteria (a), or both (a) and (b). For each arrow bounding box proposal $\mathbf{b}$, our arrow network predicts two keypoints $\bar{\mathbf{k}}^{\text{head}}$ and $\bar{\mathbf{k}}^{\text{tail}}$ which are encoded against the proposal bounding box.

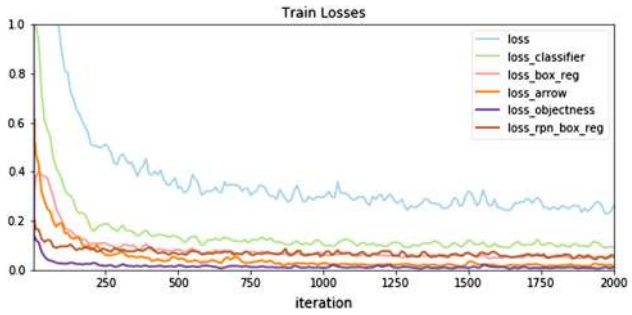


Fig. 4 $\texttt{FC\_B}_{\texttt{scan}}$ training losses during the first 2000 iterations ($\lambda = 1$)

Arrow loss L_{arw}: In the following, we discuss how we combine the individual arrow keypoint prediction into an overall arrow loss L_{arw}. Given our set of N arrow proposal pairs with the 4-d 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 (1) for arrows $i = 1 \ldots N$ and corresponding predictions $\hat{\mathbf{t}}_i \in \hat{T}$. The arrow loss is computed as the mean squared error over all predictions and dimensions:

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

Finally, we extend the Faster R-CNN box network multi-task loss L_{roi} by adding the arrow regression loss term

$$L_{\text{roi}} = L_{\text{cls}} + L_{\text{loc}} + \lambda L_{\text{arw}}, \tag{3}$$

where L_{cls} is the classification loss and L_{loc} the localization loss for bounding box refinement regression. 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 4 shows the individual loss terms throughout the first 2000 iterations on the $\texttt{FC\_B}_{\texttt{scan}}$ database.

3.3 Inference

During inference, Faster R-CNN generates a set of detections per image, where each detection has a bounding box, a predicted class, and a classification score that corresponds to the maximum softmax score over all classes. During arrow network training, the RoI align operation takes the arrow proposal bounding box and the image feature pyramid as input and computes an arrow feature map as output. During inference, this procedure differs: here, we use the final (refined) arrow bounding box from the RoI box network as input. This is due to the fact that the refined detection bounding box is more accurate, which makes it easier for the arrow network to identify the keypoint locations. The arrow network uses the arrow feature maps to compute the encoded 4-d arrow

head and tail keypoint vectors. The absolute arrow keypoint locations are then computed by applying the inverse encoding operation. Section 4.2 explains how we use the detected objects and arrow keypoints to recognize the entire diagram.

4 Integrating diagram domain knowledge

Our Arrow R-CNN network generates a set of detected symbols, where detected arrows additionally have predicted head and tail keypoints. This section addresses two key questions in using our deep learning method for diagram recognition:

1. How can we synthetically increase the size for the small diagram datasets?
2. How can we form the final diagram from a set of detections and keypoints?

For addressing the first question, we use a pipeline of augmentation methods guided by our domain knowledge about diagrams, which we outline in Sect. 4.1. Regarding the latter question, we present our diagram-aware postprocessing method in Sect. 4.2.

4.1 Augmentation

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

1. *LongestMaxSize* Resize longest image size to 1333 and preserve 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 [21], where $5 \leq w \leq 300$ and $12 \leq h \leq 150$ ($p = 1.0$)
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* Randomly rotate image by 90 degrees zero or more times ($p = 0.3$)
5. *Flip* Randomly flip image horizontally, vertically, or both ($p = 0.3$)

This pipeline is applied as a sequence, and each step is applied with probability p. We use the Albumentations library [8] for all augmentations except *IAMWordAugmentation*.

IAM word augmentation Due to the limited size of most datasets and the varying shapes and forms of arrow and text symbols, we noticed that the model frequently confuses arrows with texts and vice versa. As an example, we noticed several cases where the detector falsely predicted an arrow within a text phrase, e.g. a handwritten "l" within the term "false". To increase the robustness of arrow and text detection, during training we augment the diagrams with handwritten words from the IAM-database [21]. The corpus consists of 1066 forms written in English and produced by ≈400 different writers, resulting in more than 80k word instances out of a vocabulary of ≈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 handwritten diagrams, the forms have been scanned and contain document noise. To assure our detector does not learn to classify those text phrases solely due to their document noise, we preprocess the IAM words to increase the visual similarity to the diagram text phrases. 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 Sect. 5.1 to create words with uniform 3 pixel wide smoothed strokes. Figure 5 shows an exemplary IAM word and different preprocessing stages.

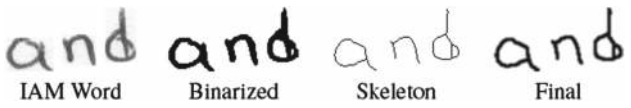


Fig. 5 IAM word preprocessing example

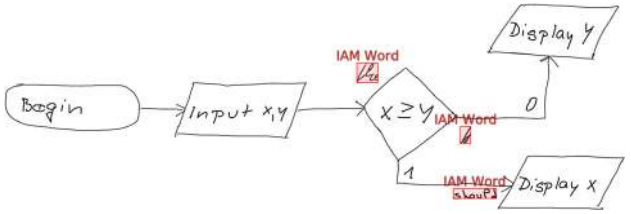


Fig. 6 Exemplary FC_B$_{\text{scan}}$ flowchart augmented with three IAM words

During training, we augment each diagram by inserting up to three random IAM words into background regions. In the diagram datasets, text phrases 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 6 shows an exemplary flowchart augmented with IAM words.

4.2 Diagram-aware postprocessing

The standard Faster R-CNN postprocessing method has a major downside for recognizing symbols in diagrams: it does not consider any domain knowledge about the global structure of diagrams. In this work, we design a postprocessing

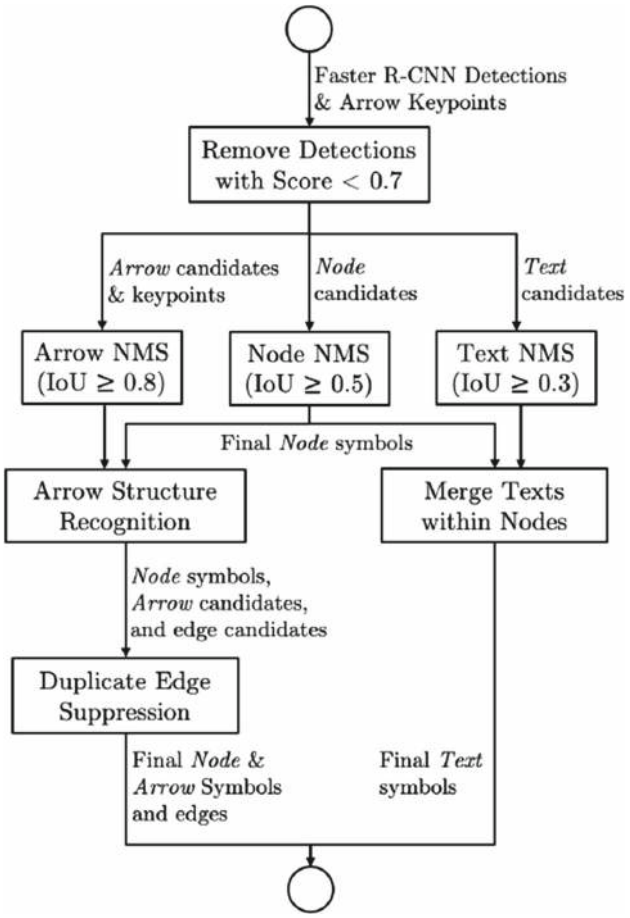


Fig. 7 *Diagram-aware postprocessing* We remove detections with a score below threshold. Then, we perform NMS over different groups of classes using dedicated IoU thresholds. Third, we generate candidate edges between nodes based on arrow keypoint to node symbol distances. Then, we ensure that there is at most one edge per direction between two nodes. We also merge text phrases located within a node

method that takes into account the following spatial and structural observations about handwritten arrow-connected diagrams:

1. Nodes in diagrams are typically drawn in a way that the bounding boxes of any two nodes have little overlap.
2. There is at most one text phrase within a node that labels this node.
3. Most graphical languages, including flowcharts and finite automata, allow at most one edge per direction between two nodes.
4. The bounding boxes of arrows can have a large overlap, especially for opposite arrows that join the same nodes.

Based on these observations, Fig. 7 shows our diagram-aware postprocessing method. We opt for a rule-based sequential method that starts by filtering symbol candidates based on a classification score threshold. We then perform

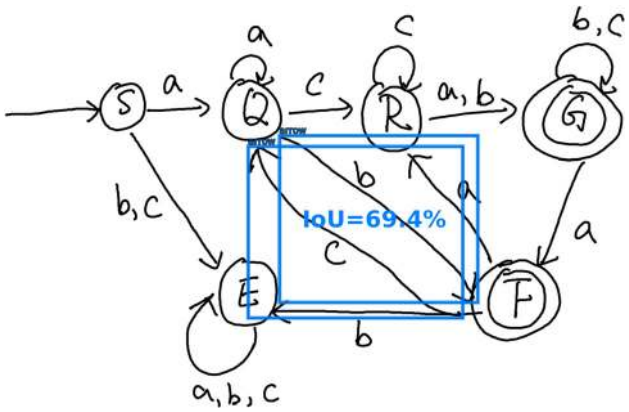


Fig. 8 *Arrows with large bounding box overlap* The two highlighted arrows have a high IoU, which makes it impossible for any Faster R-CNN detector with standard NMS to accurately detect both at once

NMS with a diagram-specific strategy. As mentioned, we observe that arrows can have large bounding box overlap. Figure 8 shows an exemplary diagram with two arrows that have close to 70% IoU. Even with a perfect model, the NMS postprocessing in standard Faster R-CNN with a 50% IoU threshold would eliminate one of both arrows (the one with the lower score). Therefore, we increase the arrow IoU threshold to 80%. Increasing the threshold generates a lot more arrow candidate detections, and often multiple candidates per ground truth arrow. To filter those duplicates, we employ a duplicate edge suppression step later in the pipeline. For Text NMS, we reduce the IoU threshold to 30%, since the axis-aligned bounding boxes typically enclose the written text very well, and it is uncommon to have text that largely overlaps. Our node NMS procedure is based on the first observation that node bounding boxes typically have little overlap. More concretely, we want to ensure that our model does not generate two detections for different node classes with almost identical bounding boxes. While allowing multiple detections for the same node 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 node classes jointly.

The arrow structure recognition step computes the distance of each arrow keypoint to its closest node and creates a candidate edge between the two respective closest nodes. For the FA dataset, where initial arrows have no predecessor node, we use a heuristic and only connect an arrow to a predecessor node if the spatial distance between the arrow tail keypoint and the node bounding box is lower than 50. The duplicate edge suppression step eliminates duplicate candidate arrows that join the same two nodes in the same direction. Duplicates are resolved by choosing the arrow with the highest classification score.

Table 1 Handwritten diagram dataset statistics

Dataset	Split	Writers (split/total)	Templates (split/total)	Diagrams	Symbols
FC_A	Train	31/35	14/28	248	5540
	Test	15/35	14/28	171	3791
FC_B	Train	10/24	28/28	280	6195
	Validation	7/24	28/28	196	4342
	Test	7/24	28/28	196	4343
FA	Train	11/25	12/12	132	3631
	Validation	7/25	12/12	84	2307
	Test	7/25	12/12	84	2323
$\text{DIDI}_{\text{no\_text}}$	Train	?/364*	940/940	27,278	193,939
	Validation	?/364*	916/940	4545	34,464
	Test	?/364*	919/940	4545	34,139
$\text{DIDI}_{\text{text}}$	Train	?/364*	5300/5629	16,717	173,070
	Validation	?/364*	2131/5629	2785	30,468
	Test	?/364*	2090/5629	2785	34,052

* The DIDI dataset is split by writer (364 total). However, the writer distribution is unknown since the writer identifiers are not public

In case the model detects multiple text phrases within a node bounding box, we merge those into a unified text phrase during postprocessing. The unified text phrase is created with a union bounding box and maximum classification score over all text phrases in question.

5 Experiments

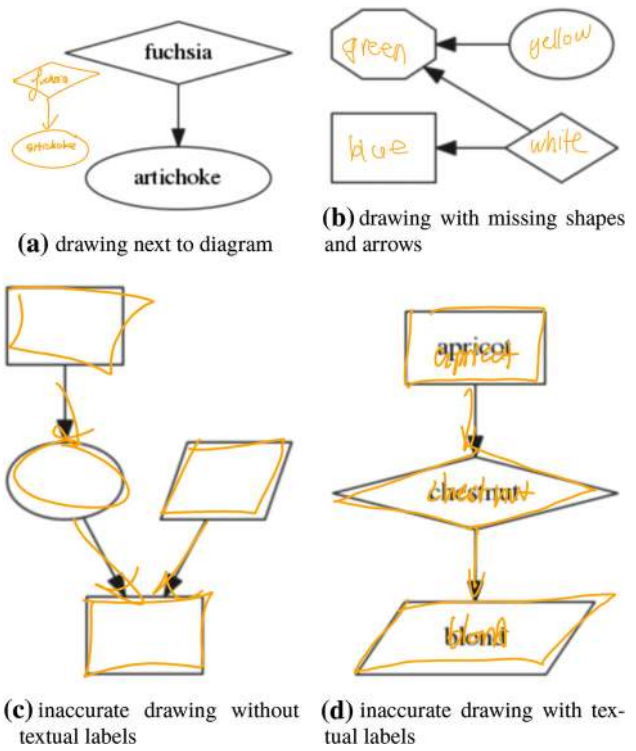
In this section, we describe the datasets (Sect. 5.1) and metrics (Sect. 5.2) used to evaluate our method, before we discuss implementation details (Sect. 5.3) and the experimental results in Sect. 5.4. We complete the experiments with an error analysis in Sect. 5.5, where we also outline how future work could address common sources of error.

5.1 Datasets

We evaluate our method on four handwritten diagram datasets, three depicting flowcharts (FC_A [1], FC_B [6], and DIDI [11]), and one finite automata dataset (FA [2]). Table 1 shows basic statistics for all datasets. As mentioned in Sect. 2.1, the DIDI dataset consists of two parts: one that contains diagrams with textual labels ($\text{DIDI}_{\text{text}}$) and one without textual labels ($\text{DIDI}_{\text{no\_text}}$). Throughout the experiments, we train on the entire DIDI dataset, but report the results for both parts separately. For all datasets, the splits were either created based on writers (FC_B, FA, DIDI) or based on templates (FC_A), such that the sets of writers or templates in the respective training, validation, and test parts 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 validation set. Obviously, it would be possible to take a subset of the train dataset as validation set. Since the majority of related works does not conduct a further split of the training set, we opt for the same approach. 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.

Online-to-offline conversion All four diagram datasets are online datasets, where each diagram has been captured as a sequence of strokes. For the FC_B dataset, we use the offline $\text{FC\_B}_{\text{scan}}$ dataset introduced in [7], which contains scans of printed FC_B diagrams. For the FC_A and FA datasets, we render the strokes as anti-aliased polylines with a stroke width of 3. To ensure border-touching strokes are fully visible, we pad each image by 10 pixels on each side. For the DIDI dataset, we create an image with the dimension of the drawing area that was shown to the user and plot the strokes at their corresponding positions. During data collection, 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 generated flowchart.

Bounding boxes For the offline $\text{FC\_B}_{\text{scan}}$ dataset, we use the provided bounding box annotations. In the following, we outline how we generate the ground truth bounding boxes for the other online datasets. For FC_A and FA, we define the symbol bounding box as the union bounding box of all its strokes. As mentioned in Sect. 2.1, the DIDI dataset is not annotated on stroke level. Instead, we use the symbol bounding boxes of the corresponding GraphViz diagram. Since the participants do not draw precisely over the diagrams, the



(a) drawing next to diagram (b) drawing with missing shapes and arrows (c) inaccurate drawing without textual labels (d) inaccurate drawing with textual labels

Fig. 9 *Exemplary* `DIDI` *diagrams with overlaid drawings* the examples illustrate the inconsistencies between GraphViz and handwritten diagrams

extracted bounding boxes do not perfectly fit the handwritten shapes. To quantify this difference, we manually annotate 100 handwritten diagrams (50 without and 50 with textual labels). Figure 9 shows some drawings from this sample and illustrates two major drawing issues we identified: diagrams where a user did not draw over the flowchart as instructed (9a) and diagrams where a user forgot to draw some or all of the shapes (9b). Since the evaluation metrics described in Sect. 5.2 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 within the 100 diagram sample, but misses one diagram where the user forgot to draw an arrow. Table 2 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.

Further, to account for inaccurate drawings such as Fig. 9c and d , we use an IoU threshold of 50% instead of 80% for the evaluation metrics described in Sect. 5.2. Within the sample of 92 diagrams without drawing mistake, 0/325 annotated nodes, 19/271 (7%) arrows, and 40/188 (21%) text phrases have less than 50% IoU between handwritten and GraphViz bounding box. Two of those text phrases and two arrows can be found in Fig. 9c and d. Overall, this means that a

Table 2 *Excluded* `DIDI` *diagrams with drawing errors*: a drawing is excluded if at least one diagram bounding box contains only white pixels

Split	$\texttt{DIDI}_{\texttt{no\_text}}$	$\texttt{DIDI}_{\texttt{text}}$
Train	404/27278 (1.48%)	1303/16717 (7.79%)
Validation	12/4545 (0.26%)	50/2785 (1.80%)
Test	12/4545 (0.26%)	867/2785 (31.13%)
Total	428/36368 (1.18%)	2220/22287 (9.96%)

For such a bounding box, the user either drew the symbol at another location, or not at all

bounding box that perfectly encloses a handwritten 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.

Arrow keypoints For training our arrow keypoint regressor, we need to specify ground truth arrow head and tail points. The head and tail points are explicitly annotated in the `FA` and $\texttt{FC\_B}_{\texttt{scan}}$ dataset. For the `DIDI` dataset, we extract the head and tail keypoints from the arrow spline 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 set the arrow head and tail keypoints as the corner points closest to the next and previous node, respectively. We 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.

5.2 Evaluation metrics

We evaluate our method using recognition metrics on symbol and diagram level. Regarding symbol recognition, Bresler, Průša, and Hlaváč [7] compute the symbol recognition recall at an IoU threshold of 80%. Additionally, arrows are required to be connected to the correct node symbols. 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 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

Table 3 *Diagram recognition rate*: Comparison of our method with other online and offline methods

	FC_A	FC_B	FA	$\text{DIDI}_{\text{no\_text}}$	$\text{DIDI}_{\text{text}}$
Online methods					
Wang et al. [36]	5.8	–	–	–	–
Julca-Aguilar et al. [14]	34.0	–	–	–	–
Bresler et al. [6]	59.1	67.9	79.8	–	–
Offline methods					
Bresler et al. [7]	–	37.7	–	–	–
Arrow R-CNN	**68.4**	**78.6**	**83.3**	**83.9**	**85.1**

Bold indicates best result per column
For FC_B, the offline results are based on $\text{FC\_B}_{\text{scan}}$

Table 4 *Augmentation ablation study*: the augmentation methods increase the diagram recognition rate for the small datasets substantially, but lower the rate on the large DIDI dataset

	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
+ RandomRotate90 & Flip	57.3	**77.0**	79.8	80.9	**83.7**
+ IAMWordAugmentation	**66.7**	76.0	**81.0**	80.8	83.5

Bold indicates best result per column
All results use standard faster R-CNN postprocessing and are based on the test set

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 pattern that stem from the scanning process, might still lead to false-positive symbols, which affect the precision of the system. In [7] 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 node NMS postprocessing, 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 of arrow predecessor and successor nodes has been correctly identified.

5.3 Implementation

Our Arrow R-CNN implementation is based on the maskrcnn-benchmark [22] R-CNN framework and uses PyTorch [25]. 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 [13]. 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 Sect. 4.2, we use an arrow distance threshold to account for arrows without a predecessor node in the FA dataset. For the DIDI dataset, we do not use augmentation methods since the dataset is very large.

5.4 Results

Diagram recognition Table 3 shows that Arrow R-CNN improves state-of-the-art in offline recognition. Even though our method uses no stroke information, the diagram recognition rates are also higher than state-of-the-art online systems.

Further, we conduct two ablation studies to quantify the effect of each augmentation and postprocessing method proposed in Sect. 4. Table 4 shows that the augmentation methods substantially improve the diagram recognition rate for small datasets, especially 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.

Table 5 *Postprocessing ablation study*: the diagram-aware postprocessing methods increase the diagram recognition rate on all test sets

	FC_A	FC_B$_{scan}$	FA	DIDI$_{no_text}$	DIDI$_{text}$
Standard NMS (IoU $\leq$ 0.5)	66.7	76.0	81.0	82.5	83.7
+ Node NMS	66.7	**78.6**	82.1	83.5	83.8
+ Arrow NMS & edge suppr.	67.8	**78.6**	**83.3**	**83.9**	**85.1**
+ Text NMS & merge node texts	**68.4**	**78.6**	**83.3**	**83.9**	**85.1**

Bold indicates best result per column

Table 6 FC_A symbol recognition at IoU 80% on test set

Class	Arrow R-CNN		Wu [38]
	Precision	Recall	Recall$_{IoU50}$
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 arrow has been correctly matched to nodes

Table 7 FC_B$_{scan}$ symbol recognition at IoU 80% on test set

Class	Arrow R-CNN		Bresler [7]
	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 arrow has been correctly matched to nodes

However, the combination of 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 shows the results of the postprocessing ablation study and reveals that node 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 FC_B$_{scan}$, where the rate stays the same. Also, merging texts within a node is a straightforward method to improve the results on FC_A.

Symbol recognition Tables 6, 7, 8 and 9 show symbol recognition results for the evaluated datasets. Overall, Arrow R-CNN achieves perfect recognition results for several node shapes, which can be explained by the fact that the shape and scale of nodes has a much lower variance than arrow and texts.

Table 8 FA symbol recognition at IoU 80% 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 arrow has been correctly matched to nodes

On the FC_A dataset (Table 6), where related works report 83.2% [38] and 84.2% [7] symbol recognition recall, Arrow R-CNN has a much higher recall (98.3%). The largest source of error for Arrow R-CNN 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 node symbols. On the FC_A training set, we noticed that our model fails to recognize diagrams of template 5 and 7. In these layouts, nodes are sometimes connected through a sequence of two arrows. Yet, our current postprocessing logic assumes that an arrow always points to a node and thus connects the arrow to the node closest to its head keypoint. We leave the development of appropriate methods for the arrow-after-arrow scenario to future work.

Table 7 shows that Arrow R-CNN can accurately recognize symbols in scanned flowcharts. In the FC_B$_{scan}$ test set, all arrows that have been detected correctly are also connected to their ground truth nodes. This demonstrates the effectiveness of the Arrow R-CNN arrow keypoint mechanism and postprocessing. Arrow R-CNN is also applicable to diagrams other than flowcharts. As Table 8 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 9 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 Sect. 5.1, this has to do with the discrepancy between the ground truth bounding boxes extracted from the diagram and the actual

Table 9 DIDI symbol recognition at IoU 50% on test set

Class	$\text{DIDI}_{\text{no\_text}}$		$\text{DIDI}_{\text{text}}$	
	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 arrow has been correctly matched to nodes

Table 10 $\text{FC\_B}_{\text{scan}}$ runtime per image: Arrow R-CNN timings are taken using a Tesla V100 GPU, with the image already resized and loaded to memory

Method	Runtime [ms]			
	Min	Mean	Std.	Max
Arrow R-CNN	59	91	15	119
Bresler et al. [7]	2623	10,970	–	37,972

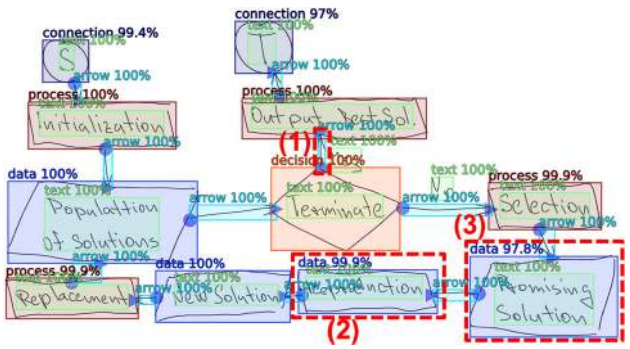


Fig. 10 $\text{FC\_B}_{\text{scan}}$ flowchart with most missing symbols in validation set: highlighted in red are (1) an arrow with only 74% IoU, (2) a process symbol confused as data, and (3) a data symbol confused as process

drawings. In Sect. 5.5, we further discuss the implications of this discrepancy and show some error cases.

Finally, Table 10 shows our system recognizes a diagram in less than 100ms on average and is two orders of magnitude faster than related work.

5.5 Error analysis and future work

Figure 10 shows the predicted symbols and arrow keypoints of a $\text{FC\_B}_{\text{scan}}$ flowchart. For texts 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 whether an 80% IoU threshold all symbols types is too strict. From an end-user perspective, it might

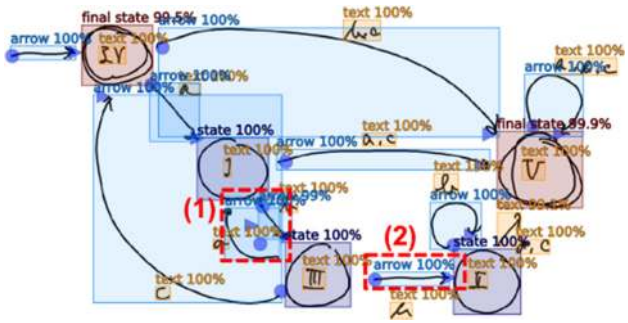
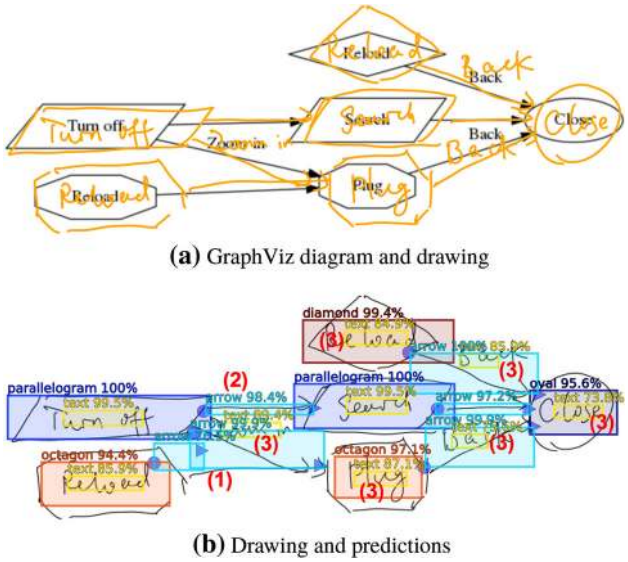


Fig. 11 FA diagram with arrow-match errors: highlighted in red are two arrows confused as initial arrows. The arrows have not been matched correctly to their source shape since the distance between predicted tail keypoint and shape bounding box exceeds the threshold

only matter that the arrow has been correctly identified as an edge between two nodes. 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 questions draws very uneven lines. These uneven lines are typically caused by uncontrolled oscillations of the hand muscles, dampened by inertia [33]. In handwriting recognition, elastic distortion augmentation is a way to simulate these oscillations [17,33]. 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 11 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 limita-



(a) GraphViz diagram and drawing

(b) Drawing and predictions

Fig. 12 DIDI$_{\text{text}}$ diagram with arrow and text errors: 12a shows the extent to which the drawn shapes and text differ from their corresponding GraphViz symbols. The errors highlighted in red in 12b 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

tions 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 [41]. These rotated bounding boxes would capture arrows in a more compact way and lead to less overlapping bounding boxes. As another approach, diagram structure recognition could be framed as visual relationship detection [16]. Instead of detecting arrow instances, a classifier could directly predict if two given nodes are connected through an arrow. Alternatively, a classifier could predict which arrow head and tail keypoint belong together.

For the DIDI dataset, Fig. 12 illustrates that the model is not only required to recognize the handwritten diagram, but also to predict the GraphViz diagram it originates from. As can be seen, the model correctly predicts node 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 differs too much, this task becomes nearly impossible. This is illustrated by the numerous text localization errors in the example, where, e.g. the hand-drawn arrow labels "Back" have very little intersection with the corresponding GraphViz texts. The example also demonstrates why the IAM word augmentation method is not very effective on DIDI, since the augmented word bounding box is defined by the handwritten strokes instead of the corresponding diagram label. Future work could propose evaluation methods that better disentangle handwriting and GraphViz diagram recognition performance for DIDI.

6 Conclusion

We propose Arrow R-CNN, the first deep learning system for offline handwritten diagram recognition. Our system is able to correctly recognize more than 68% of all diagrams in four public 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 guided by human domain knowledge. Since standard Faster R-CNN postprocessing is not well suited for diagram symbol recognition, we propose a postprocessing method that takes into the account the global structure of diagrams. Our system recognizes arrow-connected diagrams in less than 100ms on average, which allows it to be used in environments that require quick response times.

Our results bring us a step closer to the ability of assisting users in the translation of hand-drafted diagrams into an electronic version that can be managed, validated and further processed by diagram modeling software. The missing piece at this point is the integration with a handwritten text recognition system for recognizing the spotted text phrases. In future work, we plan to investigate such a combination. Further, we outline several ways on how to further improve the recognition rate, putting special emphasis on recognizing arrows and the global diagram structure.

Funding Open Access funding enabled and organized by Projekt DEAL.

Compliance with ethical standards

Conflict of interest The authors declare that they have no conflict of interest.

References

1. Awal, A.M., Feng, G., Mouchère, H., Viard-Gaudin, C.: First experiments on a new online handwritten flowchart database. In: Document Recognition and Retrieval XVIII, p. 78740A (2011)
2. Bresler, M., Phan, T.V., Prusa, D., Nakagawa, M., Hlavác, V.: Recognition System for On-Line Sketched Diagrams. In: 2014 14th International Conference on Frontiers in Handwriting Recognition, pp. 563–568 (2014)
3. Bresler, M., Průša, D., Hlaváč, V.: Modeling flowchart structure recognition as a max-sum problem. In: 2013 12th International Conference on Document Analysis and Recognition, pp. 1215–1219 (2013)
4. Bresler, M., Průša, D., Hlaváč, V.: Simultaneous segmentation and recognition of graphical symbols using a composite descriptor. In: 18th Computer Vision Winter Workshop, vol. 13, pp. 16–23 (2013)
5. Bresler, M., Průša, D., Hlaváč, V.: Detection of arrows in on-line sketched diagrams using relative stroke positioning. In: 2015 IEEE Winter Conference on Applications of Computer Vision, pp. 610–617 (2015)
6. Bresler, M., Průša, D., Hlaváč, V.: Online recognition of sketched arrow-connected diagrams. Int. J. Doc. Anal. Recognit. (IJDAR) **19**(3), 253–267 (2016)
7. Bresler, M., Průša, D., Hlaváč, V.: Recognizing off-line flowcharts by reconstructing strokes and using on-line recognition techniques. In: 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 48–53 (2016)
8. Buslaev, A., Parinov, A., Khvedchenya, E., Iglovikov, V.I., Kalinin, A.A.: Albumentations: Fast and flexible image augmentations. arXiv:1809.06839 [cs] (2018)
9. Carton, C., Lemaitre, A., Coüasnon, B.: Fusion of statistical and structural information for flowchart recognition. In: 2013 12th International Conference on Document Analysis and Recognition, pp. 1210–1214 (2013)
10. Cherubini, M., Venolia, G., DeLine, R., Ko, A.J.: 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, pp. 557–566 (2007)
11. Gervais, P., Deselaers, T., Aksan, E., Hilliges, O.: The DIDI dataset: digital ink diagram data. arXiv:2002.09303 [cs] (2020)
12. Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587 (2014)
13. Goyal, P., Dollár, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, large minibatch SGD: training ImageNet in 1 Hour. arXiv:1706.02677 [cs] (2017)
14. Julca-Aguilar, F., Mouchère, H., Viard-Gaudin, C., Hirata, N.S.T.: A general framework for the recognition of online handwritten graphics. Int. J. Doc. Anal. Recognit. (IJDAR) **23**, 143–160 (2020). https://doi.org/10.1007/s10032-019-00349-6
15. Julca-Aguilar, F.D., Hirata, N.S.T.: Symbol detection in online handwritten graphics using faster R-CNN. In: 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), pp. 151–156 (2018)
16. Krishna, R., Zhu, Y., Groth, O., Johnson, J., Hata, K., Kravitz, J., Chen, S., Kalantidis, Y., Li, L.J., Shamma, D.A., Bernstein, M.S., Fei-Fei, L.: Visual genome: connecting language and vision using crowdsourced dense image annotations. Int. J. Comput. Vis. **123**(1), 32–73 (2017)
17. Krishnan, P., Jawahar, C.V.: HWNet v2: an efficient word image representation for handwritten documents. Int. J. Doc. Anal. Recognit. (IJDAR) **22**(4), 387–405 (2019)
18. Lemaitre, A., Mouchère, H., Camillerapp, J., Coüasnon, B.: Interest of syntactic knowledge for on-line flowchart recognition. In: Graphics Recognition. New Trends and Challenges, Lecture Notes in Computer Science, pp. 89–98 (2013)
19. Lin, T.Y., Dollar, P., Girshick, R., He, K., Hariharan, B., Belongie, S.: Feature pyramid networks for object detection. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2117–2125 (2017)
20. Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft COCO: common objects in context. In: Computer Vision—ECCV 2014, Lecture Notes in Computer Science, pp. 740–755 (2014)
21. Marti, U.V., Bunke, H.: The IAM-database: an english sentence database for offline handwriting recognition. Int. J. Doc. Anal. Recognit. **5**(1), 39–46 (2002)
22. Massa, F., Girshick, R.: Maskrcnn-benchmark: fast, modular reference implementation of instance segmentation and object detection algorithms in PyTorch. https://github.com/facebookresearch/maskrcnn-benchmark (2018)
23. Newell, A., Yang, K., Deng, J.: Stacked hourglass networks for human pose estimation. In: Computer Vision—ECCV 2016, Lecture Notes in Computer Science, pp. 483–499 (2016)
24. Notowidigdo, M., Miller, R.C.: Off-line sketch interpretation. In: AAAI Fall Symposium, pp. 120–126. Arlington, VA (2004)
25. Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., Chintala, S.: PyTorch: an imperative style, high-performance deep learning library. Adv. Neural Inf. Process. Syst. **32**, 8024–8035 (2019)
26. Ren, S., He, K., Girshick, R., Sun, J.: Faster R-CNN: towards real-time object detection with region proposal networks. Adv. Neural Inf. Process. Syst. **28**, 91–99 (2015)
27. Santosh, K., Wendling, L., Antani, S., Thoma, G.R.: Overlaid arrow detection for labeling regions of interest in biomedical images. IEEE Intell. Syst. **31**(3), 66–75 (2016)
28. Santosh, K.C.: Document Image Analysis: Current Trends and Challenges in Graphics Recognition. Springer, Berlin (2018)
29. Santosh, K.C., Alam, N., Roy, P.P., Wendling, L., Antani, S., Thoma, G.R.: A simple and efficient arrowhead detection technique in biomedical images. Int. J. Pattern Recognit. Artif. Intell. **30**(05), 1657002 (2016)
30. Santosh, K.C., Lamiroy, B., Wendling, L.: Symbol recognition using spatial relations. Pattern Recognit. Lett. **33**(3), 331–341 (2012)
31. Santosh, K.C., Lamiroy, B., Wendling, L.: Integrating vocabulary clustering with spatial relations for symbol recognition. Int. J. Doc. Anal. Recognit. (IJDAR) **17**(1), 61–78 (2014)
32. Schäfer, B., Stuckenschmidt, H.: Arrow R-CNN for flowchart recognition. In: 2019 International Conference on Document Analysis and Recognition Workshops (ICDARW), p. 7 (2019)
33. Simard, P.Y., Steinkraus, D., Platt, J.C.: Best practices for convolutional neural networks applied to visual document analysis. In: Proceedings of the 7th International Conference on Document Analysis and Recognition - Volume 2, ICDAR '03, p. 958 (2003)
34. Sun, X., Xiao, B., Wei, F., Liang, S., Wei, Y.: Integral human pose regression. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 529–545 (2018)
35. Toshev, A., Szegedy, C.: DeepPose: Human pose estimation via deep neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1653–1660 (2014)
36. Wang, C., Mouchère, H., Lemaitre, A., Viard-Gaudin, C.: Online flowchart understanding by combining max-margin Markov random field with grammatical analysis. Int. J. Doc. Anal. Recognit. (IJDAR) **20**(2), 123–136 (2017)
37. Wang, C., Mouchère, H., Viard-Gaudin, C., Jin, L.: Combined segmentation and recognition of online handwritten diagrams with high order markov random field. In: 2016 15th International Con-

ference on Frontiers in Handwriting Recognition (ICFHR), pp. 252–257 (2016)
38. Wu, J., Wang, C., Zhang, L., Rui, Y.: Offline sketch parsing via shapeness estimation. In: Twenty-Fourth International Joint Conference on Artificial Intelligence (2015)
39. Xiao, B., Wu, H., Wei, Y.: Simple baselines for human pose estimation and tracking. In: Proceedings of the European Conference on Computer Vision (ECCV), pp. 466–481 (2018)
40. Yun, X.L., Zhang, Y.M., Ye, J.Y., Liu, C.L.: Online handwritten diagram recognition with graph attention networks. In: Image and Graphics, Lecture Notes in Computer Science, pp. 232–244 (2019)
41. Zhong, Z., Sun, L., Huo, Q.: An anchor-free region proposal network for faster R-CNN-based text detection approaches. Int. J. Doc. Anal. Recognit. (IJDAR) **22**(3), 315–327 (2019)