Arrow R-CNN for Flowchart Recognition

Bernhard Schäfer*†, Heiner Stuckenschmidt†
*SAP SE, Walldorf, Germany
bernhard.schaefer@sap.com
†Data and Web Science Group, University of Mannheim, Mannheim, Germany
{bernhard, heiner}@informatik.uni-mannheim.de

***Abstract*—We propose Arrow R-CNN for recognizing the symbols and structure of offline handwritten flowcharts. Arrow R-CNN extends the Faster R-CNN object detection system with an arrow keypoint predictor. This keypoint predictor is used to reconstruct the flowchart structure. We propose a network architecture and data augmentation methods that allow us to train a very deep model on a small publicly available flowchart dataset. Evaluation results show that Arrow R-CNN outperforms existing offline systems by a wide margin. For comparison with existing online flowchart recognizers, we propose an extension for mapping strokes to recognized symbols. Results show that Arrow R-CNN also achieves state of the art in online recognition, even though it does not explicitly leverage stroke information. An ablation study reveals that data augmentation guided by domain knowledge is key to high accuracy on such a small dataset.**

***Keywords*-offline recognition, flowcharts, online recognition, Faster R-CNN, object detection, sketch recognition**

I. Introduction

Research in text and graphics recognition has a long history [1]. In the area of graphics recognition, one research area deals with the recognition of diagrams such as flowcharts. Flowchart recognition can be divided into two basic approaches: online and offline recognition [2]. In online recognition, the diagrams are drawn with an ink input device such as a tablet. This input device captures the drawing as a sequence of strokes. In offline recognition, the input is a raster image and thus less structured. In the past, online flowchart recognition consistently reached higher precision than its offline counterpart. Moreover, online recognition has received more attention in research. However, offline recognition is the only viable option in many environments where individual strokes cannot be captured, e.g. when trying to recognize flowcharts drawn on whiteboards.

For recognizing objects within images, object detectors based on convolutional neural networks (CNN) are very popular. While they can be applied to detect the individual symbols of a flowchart, an off-the-shelf object detector cannot be used to detect the relationships between elements of a graphic. For example, the bounding box of the largest arrow in Fig. 1 is not sufficient to identify the symbols the arrow is connected to. Besides the challenge of recognizing symbol interconnections, training deep neural networks generally requires very large datasets. However, in the area of flowcharts, the publicly available datasets are comparatively small.

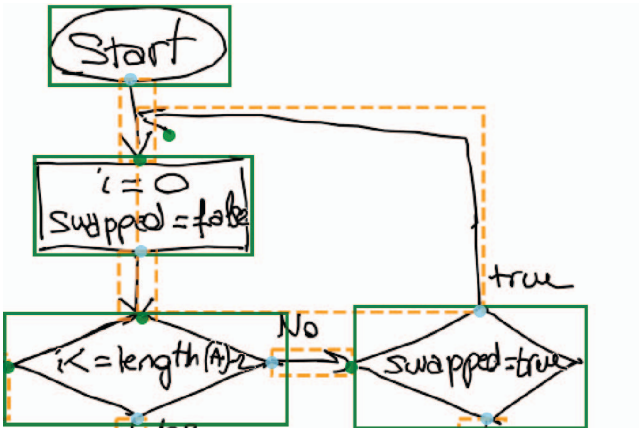

Figure 1. Flowchart picture detail: dashed bounding boxes localize each arrow, the flowchart structure can be reconstructed by connecting symbols based on their distance to predicted head and tail arrow keypoints

This work presents Arrow R-CNN, the first deep learning system for flowchart structure recognition to our knowledge. The main contributions of this work are:

- We outline how a state of the art object detector can be extended with an additional arrow keypoint predictor for flowchart structure recognition,
- we show how this keypoint predictor can be jointly trained with the other parts of the network,
- we demonstrate suitable data augmentation methods for flowchart recognition,
- we show the superior performance of Arrow R-CNN on a public flowchart dataset and
- we show that Arrow R-CNN also outperforms existing online flowchart recognition systems when using a simple offline-to-online extension.

The paper is organized as follows. Section II briefly surveys related work in flowchart recognition and object detection. Section III describes our offline recognition system and its online extension. Section IV contains experimental results and describes the data augmentation methods. Section V presents conclusions and potential future work.

978-1-7281-5054-3/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDARW.2019.00007

II. Related Work

A. Flowchart Recognition

In the area of handwritten flowchart recognition, the majority of research took place after the publication of the Online Handwritten Flowchart Dataset (OHFCD) in 2011 by Awal et al. [3], [4]. Lemaitre et al. [5] use DMOS, a grammatical method for structured document analysis, to analyze the structure of the flowcharts. Carton et al. [6] improve this method by incorporating statistical information into the grammatical description. Wu et al. [7] propose shapeness estimation to figure out if a stroke grouping has a regular appearance. In their pipeline, they use this step to generate symbol candidates as a set of strokes. They leverage the fact that symbols of one class, e.g. a decision symbol, look very similar throughout the dataset. Bresler et al. [8] propose a pipeline where they first extract symbol candidates and then use a max-sum model to solve the optimization task of finding the best set of symbol candidates. Symbol candidates are generated by grouping temporally and spatially close strokes. In follow-up works they improve their pipeline, e.g. by adding a dedicated text/non-text classifier [9]. The authors also propose an offline extension that uses a stroke reconstruction preprocessing step [2]. Wang et al. [10] train a max-margin markov random field on extracted stroke features to perform segmentation and recognition simultaneously. In [11] they extend their work by adding a grammatical description that combines the labeled isolated strokes while ensuring global consistence of the recognition. Lastly, Julca-Aguilar and Hirata [12] is the existing research most similar to our approach and besides [2] the only work on offline flowchart recognition. The authors train a Faster R-CNN object detector for symbol recognition and evaluate its performance on the OHFCD using mean average precision (mAP). However, they do not recognize the structure of the flowcharts by specifying the nodes each arrow interconnects.

B. Object Detection with R-CNN

In the area of object detection, recent work is mostly based on CNNs. Within this family, the highest accuracy object detectors use a two-stage approach popularized by R-CNN and its successor Faster R-CNN [13], [14]. The first stage proposes a set of 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. On a high-level, Faster R-CNN consists of the following sub-networks for both stages:

1) CNN Backbone Network (Stage I & II)
2) Region Proposal Network (RPN) (Stage I)
3) Head network for object detection (Stage II)

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. For example, a 1000×800 RGB image might be compressed to a $20 \times 16 \times 512$ feature map, where each entry represents a high-level feature detected within the corresponding part of the image. The *RPN* uses the feature map to propose a set of RoIs. The *head 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 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 head network uses intermediate fully connected layers before it classifies each RoI and predicts its refined bounding box using a softmax classification and a linear regression head.

There have been two notable improvements over Faster R-CNN. [15] incorporates *Feature Pyramid Networks (FPN)* into Faster R-CNN. In this extension, the backbone network generates a pyramid of feature maps at different scales. 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. The latest successor in the R-CNN family is *Mask R-CNN* [16]. Mask R-CNN replaces RoI Pooling with a *RoI Align* operation. RoI Align uses interpolation to produce a feature map more consistent with the spatial location of the RoI. Mask R-CNN is a framework for object instance segmentation, which extends object detection by predicting the set of pixels that form an object. Unlike object detection, instance segmentation has the major downside that during training it requires images to be annotated on pixel level. While it is feasible to train an instance segmentation system on an online flowchart dataset, in this work we focus on offline recognition with minimum labeling requirements. This enables us to train our system on datasets that are offline by nature and lack annotations on pixel level.

DeepPose is the first CNN-based system for *human pose estimation (HPE)*, where the task is to predict human body keypoint locations such as elbow and wrist [17]. DeepPose treats keypoint estimation as a regression problem. In contrast, Mask R-CNN has a HPE extension that predicts a binary 56×56 mask per person and keypoint. The mask is generated by a keypoint head network consisting of eight convolutional layers followed by deconvolution and upscaling layers. It is then projected over the predicted person bounding box. Our proposed Arrow R-CNN is inspired by HPE and predicts arrow instead of person keypoints. The Mask R-CNN keypoint head has many layers with a lot of trainable parameters suitable for training on a large dataset. In contrast, we want our flowchart recognizer to be trainable on a small number of images. Therefore we opt for an architecture with fewer parameters, which is closer to older HPE systems such as DeepPose.

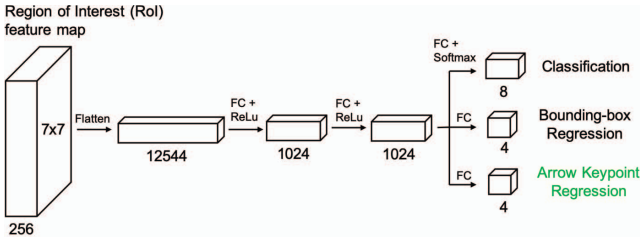

Figure 2. The Arrow R-CNN head network processes each RoI feature map through fully connected layers (FC) with rectified linear unit activation functions (ReLu). It then predicts a class, refined bounding box and arrow keypoints from a 1024-d feature vector.

III. Arrow R-CNN

A. Network Architecture

Our proposed Arrow R-CNN is based on Faster R-CNN with the FPN and RoI Align additions. We reuse the CNN backbone and RPN architecture and extend the head network to predict arrow keypoints. Fig. 2 shows the Arrow R-CNN head network. As described in Section II-B, it takes a $7\times7\times256$ RoI feature map as input. We append an arrow keypoint regressor which predicts the arrow head and tail keypoints as a 4-d vector. Similar to bounding box regression, we encode the arrow keypoint targets relative to the proposal bounding box. Suppose we have a proposal bounding box $b = (c, w, h)$ with the center point $c = (c_x, c_y)$, the width w and the height h. w and h are defined such that $4wh$ measures the size of the bounding box. For a ground-truth arrow keypoint $k = (k_x, k_y)$ assigned to a proposal with bounding box b, we define the keypoint regression target t as

$$t = (t_x, t_y) = (\frac{k_x - c_x}{w}, \frac{k_y - c_y}{h}). \tag{1}$$

This means that t_x and t_y are within the range $[-1, 1]$, unless the proposal bounding box does not fully contain the ground-truth arrow bounding box.

B. Training

During training Faster R-CNN uses the smooth L_1 loss

$$\text{smooth}_{L_1}(y, \hat{y}) = \begin{cases} 0.5(y - \hat{y})^2 & \text{if } |y - \hat{y}| < 1 \\ |y - \hat{y}| - 0.5 & \text{otherwise} \end{cases} \tag{2}$$

for bounding box regression to prevent exploding gradients [14]. For arrow keypoint regression we also use smooth L_1 loss. A flowchart with n arrows has $2n$ keypoints, one head and tail keypoint per arrow. Given the set of target keypoints over all arrows in the image $T = \{t^{(1)}, \ldots, t^{(2n)}\}$ and corresponding predictions $\tilde{T} = \{\tilde{t}^{(1)}, \ldots, \tilde{t}^{(2n)}\}$, we sum up the loss over all keypoints as the flowchart arrow loss

$$L_{arw}(\tilde{T}, T) = \sum_{i=1}^{2n} \text{smooth}_{L_1}(\tilde{t}_x^{(i)}, t_x^{(i)}) + \text{smooth}_{L_1}(\tilde{t}_y^{(i)}, t_y^{(i)}). \tag{3}$$

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

$$L_{head} = L_{cls} + L_{loc} + \lambda L_{arw}, \tag{4}$$

where L_{cls} is the classification loss and L_{loc} the localization loss for bounding box refinement regression. The hyperparameter λ controls the loss reduction over multiple arrows and the balance between arrow and other task losses. In our experiments we found that $\lambda = \frac{1}{2n}$ gives sufficient balance.

As discussed in Section II-B, there are other possible architectures for arrow keypoint prediction. For example, we could opt for a dedicated arrow keypoint branch next to the existing head that also takes as input the RoI feature map and regresses the keypoints after several layers. Our intuition is that this large increase in head parameters would lead to overfitting on small datasets. We therefore use a joint head for the three tasks which imposes a strong regularization effect. We leave a more thorough comparison of different arrow keypoint architectures for a follow-up work.

C. Inference

During inference Arrow R-CNN generates a set of detections per image, where each detection has a score corresponding to the highest softmax score. To reduce the false-positive ratio, we eliminate all detections with a score smaller than a threshold of 0.7. We found that this default framework score threshold gives a good precision-recall balance.

For each predicted arrow we specify the symbols it connects as the symbols closest to each arrow keypoint. We define the *closeness* between a keypoint and a symbol as the distance between the keypoint and the symbol bounding box. Since the bounding boxes are axis-aligned we can compute this distance as the minimum distance to each side of the rectangle.

D. Online Extension

The majority of existing work focuses on online flowchart recognition. To allow a comparison to existing online recognizers, we propose a simple stroke-mapping online extension that assigns each flowchart stroke to a detected object. We create this mapping by going through all detections and create a match for all non-matched strokes that are fully contained in the detection's bounding box. To be more robust to tiny detection bounding box errors, we expand each detection bounding box by 3 pixels during the matching process. This matching strategy only works if we iterate the detections in a meaningful order. As an example, if we first consider a process symbol in a scenario where the process contains a text phrase, we would incorrectly assign all strokes of the text phrase to the process symbol. We observe that for the most part, text phrases have tight bounding boxes and are either located within a symbol or close to an arrow. In contrast, the bounding boxes of direction-changing

arrows can become very large and contain bounding boxes of other symbols. Therefore, we group the detections into three groups of classes:

1) *Text phrases*
2) *Symbols*: Connection, Data, Decision, Process, and Terminator
3) *Arrows*

We go through each group in the illustrated order and within each group, we iterate the detections in descending detection score order.

E. Implementation

We implemented Arrow R-CNN on top of an open-source R-CNN framework implemented in PyTorch [18]. As backbone we use ResNet-101 with FPN (ResNet-101-FPN). We adopt the default framework training parameters for Faster R-CNN with ResNet-101-FPN. We use SGD with a weight decay of 0.0001 and momentum of 0.9. The model is trained for 90k mini-batches, while reducing the learning rate after 60k and 80k iterations by a factor of 10. We use a single Tesla V100 GPU with 16GB memory for training, which fits a batch size of 8. We use the learning rate 0.01 recommended for a batch size of 8. We did not notice a consistent improvement when experimenting with other learning rate schedules. Although the 90k iterations are intended for a much larger dataset, we did not notice overfitting when using multiple augmentation methods. On our GPU training takes about 20 hours. Inference on a single image takes ≈100ms on mentioned GPU and ≈3s on a MacBook Pro CPU.

[12] found that a model pre-trained on natural images from the COCO dataset converges much faster than a model with randomly initialized weights. Our initial experiments confirm those findings, we therefore use pretrained COCO models throughout all reported experiments. In the default setting of the framework, the low-level feature extraction layers of the backbone network are frozen to prevent overfitting. We found that training the entire backbone in combination with data augmentation greatly improves accuracy while still preventing overfitting. We attribute this to the fact that images from everyday scenes are very different from documents.

IV. Experiments

We evaluate Arrow R-CNN on the Online Handwritten Flowchart Dataset (OHFCD) [3], [4]. The dataset contains 419 handwritten flowcharts from 35 writers, divided in 248 train and 171 test set flowcharts. Fig. 3 shows a rendered flowchart from the test set along with Arrow R-CNN predictions.

A. Dataset Generation

Since the OHFCD is an online dataset, we need to render the strokes of a flowchart as a raster image. Throughout all

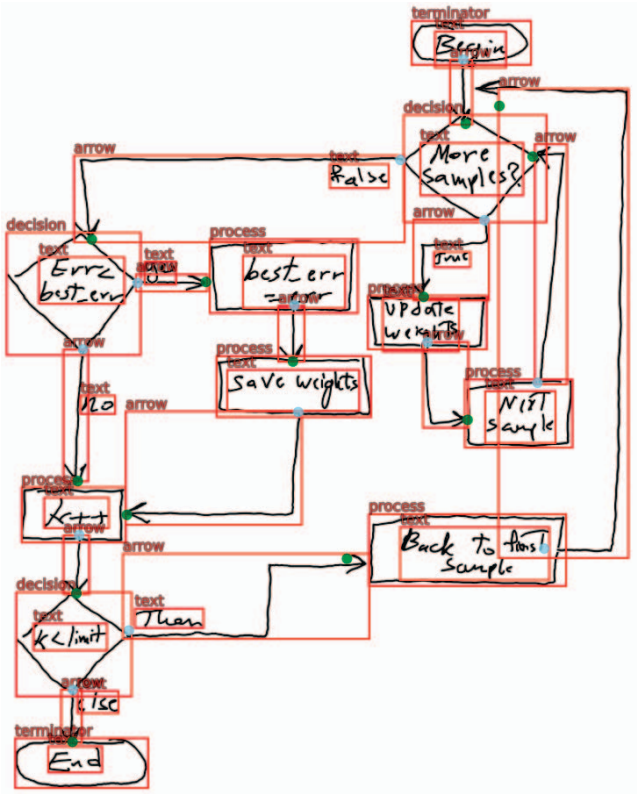

Figure 3. Arrow R-CNN test flowchart result: Arrow R-CNN correctly recognizes complex flowchart layouts with only minor bounding box and arrow keypoint deviations

flowcharts, the stroke traces are in the range $[0, 2000]$. To generate a raster image that looks as natural as possible, we use the following rendering process: The strokes are plotted as anti-aliased polylines with a stroke width of 3. We use a 2010×2010 image and shift the stroke range to $[5, 2005]$ to ensure that border-touching strokes are fully contained within the image. Afterwards the images are resized to 800×800 using inter-area interpolation. Our approach differs from [12] where flowcharts are visualized with one pixel wide strokes.

For regressing the arrow keypoints we need to specify the target head and tail coordinates for each arrow. These coordinates are not explicitly provided in the OHFCD, which only specifies the strokes that make up each arrow. As mentioned in Section III-C, during postprocessing we connect each arrow to the symbol with the closest bounding box distance to the respective keypoint. To exploit this closeness property, we follow a similar approach and approximate each arrow keypoint as the arrow pixel with minimum distance to the pixels of its connected node. Fig. 1 illustrates the shortcoming of arrow keypoints when a sequence of arrows is used to interconnect two symbols. However, it also shows that our keypoint approximation mitigates this problem. For

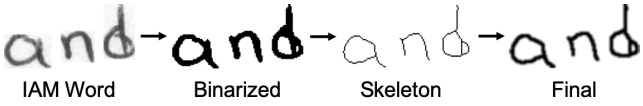

Figure 4. IAM Word Preprocessing Example

the largest arrow, the arrowhead tip is closer to the incorrect terminator symbol, whereas the approximated keypoint is closer to the true process symbol.

Future research could take this keypoint concept further and directly predict the bounding box center of the connected symbols of an arrow. This would eliminate the need for arrow keypoint labels. However, this would likely require tweaks such as increasing the receptive field of the RoI feature map.

B. Augmentation

Since our training set contains only 248 images, we use augmentation during training to improve generalization.

1) Train-time image transformations: We apply the following random image transformations:

- Random rotation and flipping: images are randomly flipped horizontally and rotated by $0°$, $90°$, $180°$, or $270°$
- Random resizing: images are randomly resized to scales of $[640, 800]$ pixels, with a step size of 32.

The flowchart symbols can still be recognized when rotating them by a multiple of $90°$. We did not use other rotation angles such as $45°$ to avoid confusion between symbols, e.g. a decision symbol rotated by $45°$ looks like a process symbol.

2) Train-time IAM word augmentation: The most challenging classes are arrow and text due to their varying shapes and forms. 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 improve the arrow and text phrase detection, we opted to augment the flowcharts during training with handwritten words from the IAM-database [19]. The corpus consists of 1,066 forms written in English and produced by ≈400 different writers, resulting in more than 80k word instances out of a vocabulary of ≈11k words. Unlike the online handwritten flowcharts, 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 OHFCD 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 using [20]. Afterwards, we use a procedure similar to the flowchart strokes rendering process to create words with uniform 3 pixel wide smoothed strokes. An exemplary IAM word and different preprocessing stages are shown in Fig. 4.

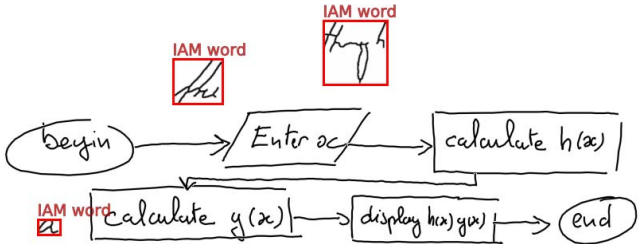

Figure 5. Exemplary OHFCD flowchart augmented with three IAM words

During training, we augment each flowchart by inserting three random IAM words into background flowchart regions. In the flowchart dataset, 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 flowchart element 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 $[3, 50]$. Fig. 5 shows an exemplary flowchart augmented with IAM words.

C. Evaluation Metrics

The two existing offline systems [12] and [2] use different evaluation metrics. For comparison we report both metrics. Julca-Aguilar and Hirata [12] use mean average precision (mAP) at an intersection over union (IoU) threshold of 50%, a standard metric in object detection. Bresler et al. [2] report the correct rate of symbol segmentation and recognition (SR), a standard metric in flowchart recognition. SR requires matching labels and at least 80% IoU between the ground truth and predicted bounding box. Arrows are additionally required to be connected to the correct symbols, which we refer to as arrow-match criteria AM. We use the notation SR_{80+AM} to explicitly denote the IoU threshold and whether the AM criteria is used. For high-level results on graph-level, we report the offline flowchart recognition (FR) rate as the proportion of flowcharts without a single recognition error. Here, we use the same subscript notation for comparative results.

For online recognition, we report the stroke and symbol recognition rates. In this setting, a symbol is recognized if all its strokes and its class have been correctly identified.

D. Results

1) Offline Symbol and Graph Recognition: Julca-Aguilar and Hirata [12] report 97.7% mAP on the test set. Arrow R-CNN achieves a higher mAP without any and with augmentation (99.3%/99.8%). This gap might be partially due to FPN, which is better at detecting small objects such as single letter text phrases. Regarding the SR metric, Table I reveals that Arrow R-CNN achieves more than 99% precision and recall on all classes except text and arrow. Table II shows the overall offline symbol recognition rate in comparison to

Table I
OFFLINE SYMBOL RECOGNITION PER CLASS ON TEST FLOWCHARTS

Class	SR_{80+AM} Precision (%)	SR_{80+AM} Recall (%)
Connection	99.2	100.0
Data	99.7	99.7
Decision	100.0	99.5
Process	99.3	100.0
Terminator	99.5	100.0
Text	98.5	98.3
Arrow	95.8	96.4

Table II
OFFLINE SYMBOL RECOGNITION RATE ON TEST FLOWCHARTS

	SR_{80+AM} (%)
Bresler et al. [2]	84.2
Arrow R-CNN	87.4
+ resize	89.8
+ flipping/rotation	96.8
+ 3 IAM words	**97.9**

other systems. As can be seen, Arrow R-CNN outperforms existing offline methods by a wide margin. The table also shows the benefits of different augmentation methods as an ablation study.

Table III reports the flowchart recognition rate. As can be seen, the largest improvement is gained from flipping and rotating the flowchart images. Using augmentation with IAM words further increases FR_{80+AM} from 62.6% to 68.4%. The test set flowcharts greatly differ in terms of recognition difficulty. 25 out of 171 flowcharts do not have any text phrases, which makes recognition far less complex. On the other hand, there are flowcharts with many symbols, text phrases and long reaching arrows. Given this spectrum of difficulty, the results show the importance of non-standard augmentation methods for recognizing complex flowchart layouts.

2) Online Symbol Recognition: In Table IV, we compare Arrow R-CNN to existing systems that report an online symbol recognition rate higher than 80%. Arrow R-CNN outperforms existing online recognition systems, without having knowledge of the temporal order of strokes and without using other explicit stroke features.

Table III
OFFLINE FLOWCHART RECOGNITION RATE ON TEST FLOWCHARTS

method	FR_{50}	FR_{50+AM}	FR_{80}	FR_{80+AM}
Arrow R-CNN	42.1	26.9	29.8	21.1
+ resize	46.2	32.7	36.8	26.9
+ flipping/rotation	79.5	73.1	66.7	62.6
+ 3 IAM words	**90.1**	**77.2**	**77.2**	**68.4**

Table IV
ONLINE STROKE AND SYMBOL RECOGNITION RATE ON TEST FLOWCHARTS

	Stroke Rec. (%)	Symbol Rec. (%)
Wu et al. (2015) [7]	94.9	83.2
Wang et al. (2016) [10]	95.8	84.3
Wang et al. (2017) [11]	96.2	80.7
Bresler et al. (2016) [9]	96.3	84.2
Arrow R-CNN	**97.7**	**93.5**

E. Error Analysis

The vast majority of errors relate to the text and arrow classes. For text phrases, there are cases where multiple phrases are detected within a symbol. These errors could be avoided by merging detected text phrases inside symbols. Another source of error are large arrows with multiple changes of direction. For those arrows, sometimes the detector fails to accurately identify the bounding box or even predicts two arrows. A structural analysis component might help in reducing these error types. During error analysis we also found ground truth labeling errors that we corrected and plan to submit with this work.

V. CONCLUSION

We propose Arrow R-CNN, an offline handwritten flowchart recognizer that is able to correctly recognize the majority of flowcharts in a public dataset. Our work shows that we can train a highly accurate deep R-CNN model on a small dataset when using data augmentation methods guided by human domain knowledge. Without any data augmentation methods, our system outperforms prior work using Faster R-CNN [12]. We attribute this partially to Feature Pyramid Networks (FPN) [15], which help to detect small objects. FPN might also improve other existing Faster R-CNN systems in the context of 2D handwritten languages, such as the music object detector by Pacha et al. [21].

Our experiments show that text phrases and arrows are the biggest challenge in flowchart recognition. We believe that our system could benefit from structural and grammatical analysis methods used in related work [9], [11]. By detecting if a subset of the symbol candidates forms a valid diagram, the system could become more robust to false positive candidates.

As illustrated in Section I, bounding box object detection is not sufficient to recognize flowcharts interconnections. Our work shows that we can overcome this limitation with an arrow keypoint regressor. Other 2D handwritten languages might also benefit from a keypoint component in order to detect objects at a higher granularity. For example, a music note could be detected as one object with several component keypoints, such as a note head keypoint. Future work could investigate the effectiveness of the outlined keypoint architectures to flowchart recognition and other 2D languages.

REFERENCES

[1] J. Lladós, "Two Decades of GREC Workshop Series. Conclusions of GREC2017," in *Graphics Recognition. Current Trends and Evolutions*, ser. Lecture Notes in Computer Science, 2018, pp. 163–168.

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

[3] H. Mouchère, "Online Handwritten Flowchart Dataset (OHFCD)," http://tc11.cvc.uab.es/datasets/OHFCD_1.

[4] A.-M. Awal, G. Feng, H. Mouchère, and C. Viard-Gaudin, "First Experiments on a new Online Handwritten Flowchart Database," in *Document Recognition and Retrieval XVIII*, Jan. 2011, p. 78740A.

[5] A. Lemaitre, H. Mouchère, J. Camillerapp, and B. Coüasnon, "Interest of Syntactic Knowledge for On-Line Flowchart Recognition," in *Graphics Recognition. New Trends and Challenges*, ser. Lecture Notes in Computer Science, 2013, pp. 89–98.

[6] C. Carton, A. Lemaitre, and B. Coüasnon, "Fusion of Statistical and Structural Information for Flowchart Recognition," in *2013 12th International Conference on Document Analysis and Recognition*, Aug. 2013, pp. 1210–1214.

[7] J. Wu, C. Wang, L. Zhang, and Y. Rui, "Offline Sketch Parsing via Shapeness Estimation," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, Jun. 2015.

[8] M. Bresler, D. Prùa, and V. Hlavác, "Modeling Flowchart Structure Recognition as a Max-Sum Problem," in *2013 12th International Conference on Document Analysis and Recognition*, Aug. 2013, pp. 1215–1219.

[9] M. Bresler, D. Průša, and V. Hlaváč, "Online recognition of sketched arrow-connected diagrams," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 19, no. 3, pp. 253–267, Sep. 2016.

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

[11] C. Wang, H. Mouchère, A. Lemaitre, and C. Viard-Gaudin, "Online flowchart understanding by combining max-margin Markov random field with grammatical analysis," *International Journal on Document Analysis and Recognition (IJDAR)*, vol. 20, no. 2, pp. 123–136, Jun. 2017.

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

[13] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587.

[14] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," in *Advances in Neural Information Processing Systems 28*, 2015, pp. 91–99.

[15] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature Pyramid Networks for Object Detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 2117–2125.

[16] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 2961–2969.

[17] A. Toshev and C. Szegedy, "DeepPose: Human Pose Estimation via Deep Neural Networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1653–1660.

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

[19] U.-V. Marti and H. Bunke, "The IAM-database: An English sentence database for offline handwriting recognition," *International Journal on Document Analysis and Recognition*, vol. 5, no. 1, pp. 39–46, Nov. 2002.

[20] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "Scikit-image: Image processing in Python," *PeerJ*, vol. 2, p. e453, Jun. 2014.

[21] A. Pacha, K. Choi, B. Coüasnon, Y. Ricquebourg, R. Zanibbi, and H. Eidenberger, "Handwritten Music Object Detection: Open Issues and Baseline Results," in *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*, Apr. 2018, pp. 163–168.