CS771 Project Final Report: Offline Hand-Drawn Diagram Conversion

Moniek Smink
smink2@wisc.edu

Venkata Saikiranpatnaik
vbalivada@wisc.edu

Sourav Suresh
sourav.suresh@wisc.edu

1. Team Member Contributions

Name	Contribution
Moniek Smink *smink2@wisc.edu*	The rest: Models, Custom Data, etc.
Venkata Saikiranpatnaik *vbalivada@wisc.edu*	Image Data Setup Similarity Testing
Sourav Suresh *sourav.suresh@wisc.edu*	IAM Text Data Setup ChatGPT Plu$

2. Introduction

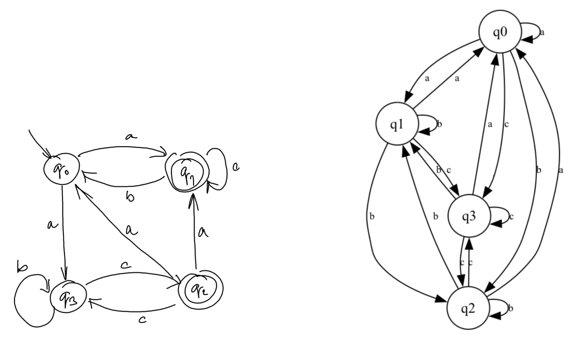
In many fields, text, node, and arrow diagrams are widely used to represent things such as procedures, decision trees, and organizational plans. Visual representations of complex ideas can often be more intuitive and easier to understand for a wider audience. Thus, these diagrams are regularly used within publications, presentations, and discussions. Often, these diagrams are made quickly on the fly in a handwritten format on devices such as tablets and mobile phones. However, when these diagrams are presented, they must be converted into a polished digital format. We wanted to automate the process of diagram conversion from a hand-drawn to a digital format. This would allow authors and presenters to spend less time in formatting software and more time researching, thinking, and presenting.

We have created a system of models that can perform offline hand-drawn diagram conversion. This system takes as input static hand-drawn diagram images, detects text, nodes (shapes), and arrows, determines the relationship between these objects, reads any text, and creates a digital image version as the output. The code is open sourced at https://github.com/bvskp-projects/YoloGraph/.

3. Related Work

3.1. Diagram Recognition

There are two kinds of diagram recognition: online and offline. Online diagram data contains a series of brushstrokes with corresponding timestamps that make up the final diagram. Offline diagram data has only the final product without the individual stroke data. A lot of work has been done in online diagram recognition [2, 12, 13]. However, online diagram recognition is limited when brushstroke information is not available. Further work has also been done in offline diagram recognition [3, 8, 11]. However, these approaches often involve more time consuming models such as Faster-RCNN [9] and rarely directly read the text involved in the diagram. We wanted to build an efficient offline diagram recognition system with text recognition capabilities.



(a) Finite Automaton Diagram (b) Output from ChatGPT

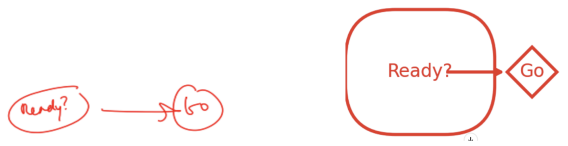
Figure 1. Finite Automaton digital diagram generated by ChatGPT+

3.2. Generative AI: Chat-GPT

In this subsection, we illustrate the main challenges with handdrawn diagram recognition using ChatGPT+ as baseline tool for solving this problem.

3.2.1 Relationships

Let's take a look at a simple hand-drawn finite automaton diagram as seen in figure 1a. The best output we could generate using ChatGPT+ is shown in the figure on the right 1b. Clearly, ChatGPT+ is struggling a lot to identify the right relationships between the nodes. Hence, we claim that the current state-of-the-art multi-modal language models have difficulty identifying graph-structured latents. Hence, we introduce heavy inductive bias assuming that the underlying structure of the input images is a graph with directed edges to get better predictive performance.



(a) Shape Recognition Input (b) Shape Recognition Output

Figure 2. Shape recognition using ChatGPT+

3.2.2 Shapes

The next challenge is shape recognition. We run a simple input graph 2a through ChatGPT+ to identify the right shapes and generate the image. We use a single-step, non-iterative approach but experiment with a few prompts to extract shape information. The best image generated 2b is far from accurate even on a simple diagram. We attribute this to distributional challenges and human preferences. ChatGPT+ generates the figure on the right because it "thinks" that a human would prefer the image on the right. Aesthetic aspects shouldn't trump the accuracy of shapes. Our approach exhibits a much better accuracy on shapes. That said, we fail to generalize to new shapes. This is also discussed in the results section 5.2.

4. Methods and Evaluating Individual Models

Our offline diagram conversion system functions in several steps. First, a YOLOv5s object detection model [5] recognizes nodes, text, and arrows within the input image. Then, any text objects are read using a TPS-ResNet-BiLSTM-Attn text recognition model [1] and arrows are extrapolated from the arrow bounding boxes. Finally, the bounding boxes, arrow extrapolates, and text readings are used to form an output image.

4.1. Object Detection Model

A pre-trained YOLOv5s object detection model [5] was fine-tuned on the FCA and FCB datasets [7, 11] for 100 epochs using SGD with compound loss, learning rate of 0.01, weight decay of 0.0005, three warmup epochs, and momentum of 0.937. The datasets contained a total of 805 and 90 images in the train and validation sets, respectively. (In our proposal, we stated the intention of using the DIDI dataset [4] for training. However, this dataset turned out to not contain the type of object detection bounding box data we needed.) The model is currently limited to 6 classes: circle, rectangle, parallelogram, diamond, arrow, and text. The final version of the model had an mAP@0.5 of 0.991 and mAP@0.5:0.95 of 0.873, likely due to the simplicity of the detection task and limited data. For a depiction of the mAPs, losses, precisions, and recalls during training please see Figure 3. For a depiction of the precision-recall curve of the final model please see Figure 4. For a depiction of sample outputs please see Figure 5.

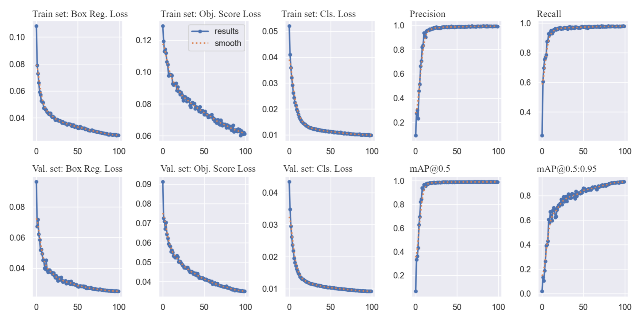


Figure 3. The training/validation losses, mAPs, precisions, and recalls during the training of the YOLOv5s diagram node detection model.

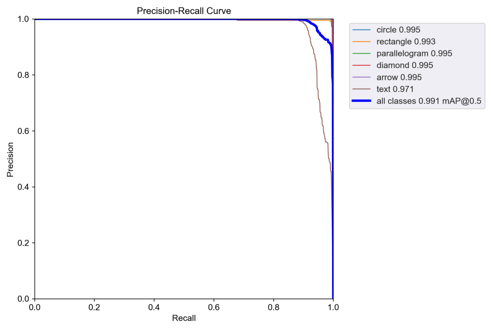


Figure 4. The precision-recall curve of the final YOLOv5s diagram node detection model.

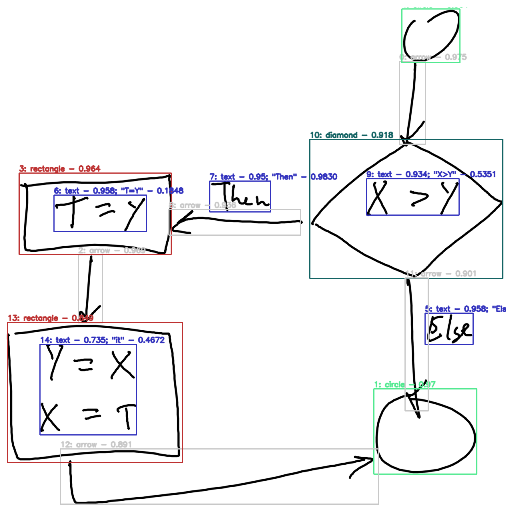


Figure 5. Sample results of the YOLOv5s model.

4.2. Text Recognition Model

A pre-trained case-sensitive TRBA scene text recognition model was fine-tuned on a subset of the IAM handwritten sentence database [6] and some hand-labeled texts cropped out of the FCA and FCB datasets [7, 11] for 300 epochs using an AdaDelta optimizer [14] with a decay rate of 0.95, batch size of 192, and gradient clipping magnitude of 5. This model consists of a thin-plate spine (TPS) input image normalizer to straighten out text, a ResNet feature extractor, a bidirectional LSTM sequence modeler, and an attention-based sequence predictor (Attn) as described in [1]. The dataset contained a total of 35,120 and 3,922 images in the train and validation sets respectively. 738 of these images were hand-labeled images cropped out of the FCA and FCB datasets; the rest of the images were from the IAM database. The final version of the model had an overall accuracy of 85.186% on the validation set. For a depiction of sample outputs please see Figure 6.

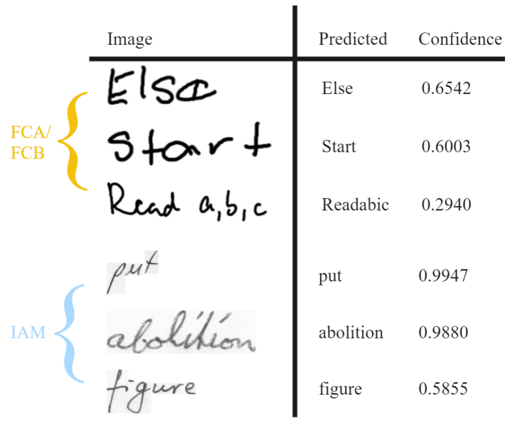


	Image	Predicted	Confidence
FCA/ FCB	Elsa	Else	0.6542
	Start	Start	0.6003
	Read a,b,c	Readabic	0.2940
IAM	put	put	0.9947
	abolition	abolition	0.9880
	figure	figure	0.5855

Figure 6. Sample results of the TRBA model.

4.3. Arrow Extrapolation

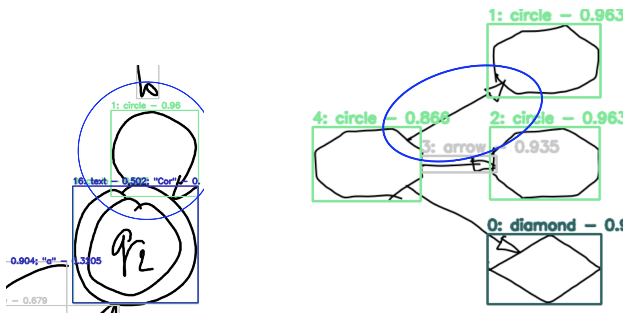
After the arrows within a graph are detected, the arrow end points and direction must be determined. If we had more time, we would have preferred to train a key point detection model to do this. Until then, we extrapolate the arrows from a bounding box in post-processing. First, we transpose the bounding box so the longest side is the width. Next, we cut the bounding in half and determine the half with more drawn in it. Next, we cut the half into three parts making a top, middle, and bottom section. The section with the most drawn in it is deemed the end point of the arrow. Then, the other half of the original cut is split into three parts. The top, middle, or bottom section with the most drawn in it is deemed the start point of the arrow. Finally, the arrow start point and end point are transposed as needed and returned. The limitations of this approach are discussed in Section 5.3.1.

5. Evaluating the Whole System

In this section, we first describe the datasets we use for our benchmark. Next, we introduce a **S**imple **S**imilarity **S**core measure that we refer to as S3 for evaluation. Finally, we report our results and provide relevant insights.

Dataset	FA	FCA	FCB	FCB_SCAN	DIDI
Correct	4	155	186	192	42
Total	18	171	196	196	130
Score (%)	5	91	95	98	32

Table 1. Similarity Score for different datasets



(a) Self loop misclassified as a circle (b) Arrow unrecognized for style reasons

Figure 7. Shape Generalization Problems

5.1. Experimental Setup

We preprocess the diagrams provided in [10] and [4]. We then use the test images for our benchmark. You can see the total number of test images for each dataset in Table 1. `FA` represents the finite automaton dataset, `FC` prefixed datasets are flowchart images, and `DIDI` is an early dataset for hand-drawn diagrams.

5.2. Results

We compute the similarity score by comparing the graph representation. We do not compute similarity directly on the image since images have a lot of redundancy such as color, etc. We classify a test image as correctly transformed when (a) the shapes cross a specific object score threshold (b) the number of shapes for each instance matches the ground truth and (c) the number of arrows match. We do not match text or direction of the arrow for reasons mentioned in the limitations section 5.3. We do this for each image and compute the fraction of correctly transformed images. We report this score as percentage in Table 1.

We perform really well on flowchart-based datasets (prefixed with FC). This is expected since we train our models on FCA and FCB. However, we do poorly on finite automaton and somewhat less poorly on the DIDI dataset. We try to understand the underlying reasons for this behavior in Figure 7. Our shapes do not generalize well because of the limited dataset we trained on. For example, the self loop in Figure 7a is misclassified as a circle since our model is not trained on automata. Similarly, we are unable to recognize the arrows in the figure on the right (7b) because it is out-of-distribution for our model.

5.3. Limitations & Future Improvements

5.3.1 Arrow End Points & Direction

Our system has trouble with drawing the final arrows. Detecting the location of the arrows is no problem as shown by the mAP of 0.995 for the arrow class in the object detection model. However, it is difficult to extrapolate the exact arrow end points from a bounding box, especially when arrows come in many different forms. For examples of arrow bounding box, output arrow pairs please see Figure 8. As you can see, simple arrows such as left-right or top-down arrows are easily extrapolated. However, for arrows that intercept shapes or arrows that are drawn with a curve or corner, our system is not able to easily determine where exactly the arrows start and end and also which direction the arrowhead goes in. If we had more time, our team would train a key point detection model that would take the arrow bounding boxes and find the start and end point of the arrow. We leave this part unfinished.

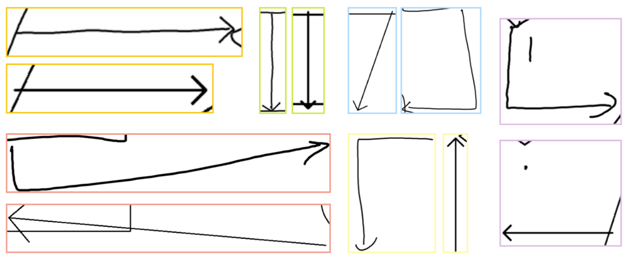


Figure 8. Sample ground truth extrapolate arrow pairs.

5.3.2 Text Recognition: Formulas & Multi-line text

The IAM text dataset we chose to train on was based on handwritten text that had been written on paper and then scanned into a digital format. This meant that this data was not very representative of what is commonly seen in diagrams written in a digital format. Firstly, there were no formulas in the IAM database, meaning any math formulas seen in the validation sets would not be read very well. Secondly, the subset of the IAM database we trained on never contained any multi-line text, meaning the text recognition model was never trained on text that was written in two or more lines. To try to mitigate some of these effects, we added some hand-labeled cropped out regions of text from the diagram datasets we had (738 in total). This slightly improved the performance of the text model. Simpler formulas could now be recognized, while more complicated formulas were still mostly incorrect (see Figure 9). Multi-line text was still mostly incorrect with the new training data (see Figure 10). In the future, a more representative text dataset should be chosen with plenty of formulas and more multi-line text.

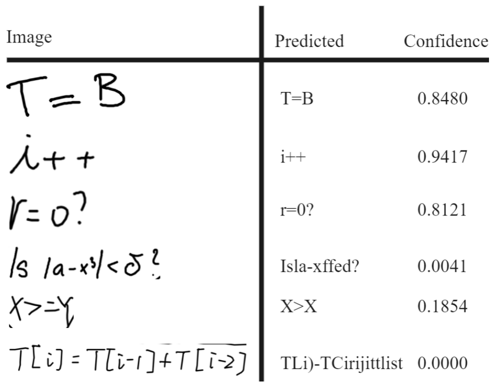


Image	Predicted	Confidence
T=B	T=B	0.8480
i++	i++	0.9417
r=0?	r=0?	0.8121
Is |a-x³|<δ?	Isla-xffed?	0.0041
X>=Y	X>X	0.1854
T[i]=T[i-1]+T[i-2]	TLi)-TCirijittlist	0.0000

Figure 9. Sample results of the TRBA model on formulas.

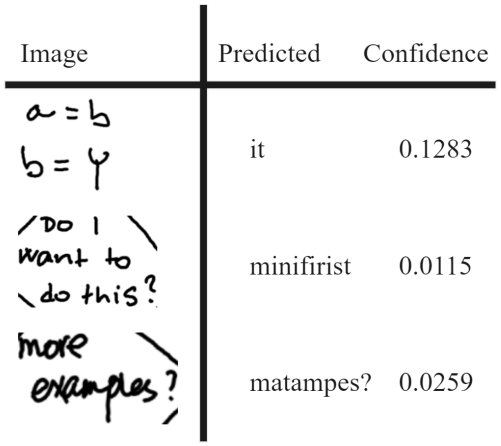


Image	Predicted	Confidence
a=b b=Y	it	0.1283
/Do I want to do this?\	minifirist	0.0115
more examples?	matampes?	0.0259

Figure 10. Sample results of the TRBA model on multi-line text.

6. Conclusion

Our system has the potential to accurately convert offline hand-written diagrams to a digital format. There are several places where we could improve our system including having more representative text data and leveraging a key point detection model for the arrow extrapolation. An offline hand-written diagram converter would allow users to quickly convert their diagram rough drafts to final polished formats ready for presentation, saving them time in their day-to-day processes.

References

[1] Jeonghun Baek, Geewook Kim, Junyeop Lee, Sungrae Park, Dongyoon Han, Sangdoo Yun, Seong Joon Oh, and Hwalsuk Lee. What is wrong with scene text recognition model comparisons? dataset and model analysis. *CoRR*, abs/1904.01906, 2019. 2, 3

[2] Martin Bresler, Truyen Van Phan, Daniel Prusa, Masaki Nakagawa, and Václav Hlavác. Recognition system for on-line sketched diagrams. In *2014 14th International Conference*

on Frontiers in Handwriting Recognition, pages 563–568, 2014. 1

[3] Martin Bresler, Daniel Průša, and Václav Hlaváč. Recognizing off-line flowcharts by reconstructing strokes and using on-line recognition techniques. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 48–53, 2016. 1

[4] Philippe Gervais, Thomas Deselaers, Emre Aksan, and Otmar Hilliges. The didi dataset: Digital ink diagram data, 2020. 2, 3

[5] Glenn Jocher. YOLOv5 by Ultralytics, 2020. 2

[6] U-V Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5: 39–46, 2002. 2

[7] Mouchère. Online handwritten flowchart dataset (ohfcd). 2

[8] Matthew J. Notowidigdo and Robert C. Miller. Off-line sketch interpretation. In *AAAI Technical Report*, 2004. 1

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2016. 1

[10] Bernhard Schäfer. Handwritten diagram datasets. https://github.com/bernhardschaefer/handwritten-diagram-datasets, 2022. 3

[11] Bernhard Schäfer, Margret Keuper, and Heiner Stuckenschmidt. Arrow r-cnn for handwritten diagram recognition. *International Journal on Document Analysis and Recognition (IJDAR)*, 24(1):3–17, 2021. 1, 2

[12] Chengcheng Wang, Harold Mouchère, Christian Viard-Gaudin, and Lianwen Jin. Combined segmentation and recognition of online handwritten diagrams with high order markov random field. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 252–257, 2016. 1

[13] Xiao-Long Yun, Yan-Ming Zhang, Jun-Yu Ye, and Cheng-Lin Liu. Online handwritten diagram recognition with graph attention networks. In *Image and Graphics*, pages 232–244, Cham, 2019. Springer International Publishing. 1

[14] Matthew D. Zeiler. Adadelta: An adaptive learning rate method, 2012. 3