A Single Shot MultiBox Detector Based Handwritten Formula Detector

Group 48 Final Project Report

Reid Chen
Department of Computer Sciences, Junior

Jijie Zhang
Department of Economics, Junior

University of Wisconsin-Madison
January 22, 2021

1 Introduction and Background

As a college student, possessing great note-taking skill is the path to success. Nowadays, many students decide to use the iPad or Microsoft Surface to take handwritten notes due to the convenience brought by electronic devices. One of the benefits of taking notes in this manner is that these electronic note-taking applications can recognize handwritten texts so that students can search through their notes and find what they need instantly. Although textual notes can be converted to searchable information successfully, many applications fail to recognize mathematical formulas or expressions, which are pervasive in a note from science or engineering student. Therefore, searchable formulas might be something application users are requiring.

Although many note-taking apps do not have the feature of recognizing math symbols globally, i.e., these applications cannot detect where the formulas are. Some of them have the ability to recognize math formulas given that the user explicitly tells the application where the location of math formulas in the notes. For example, Notability [1] recently announced a math conversion feature that converts a selected region of handwritten math into typewritten math. This shows that a mathematical recognizer exists. In fact, handwritten mathematical formula recognizers like [2] outperforms the state-of-the-art accuracy in the Competition on Recognition of Online Handwritten Mathematical Expressions (CROHME). These recognizers have also been commercialized. For example, a mathematical recognizer called Mathpix [3] provides an accurate mathematical recognizer API. However, these existing recognizers only work well on an image containing a single formula. They cannot be used directly because, in reality, notes contain note only mathematical formulas, but also include other information like texts, figures, and graphs. Therefore, retrieving the exact location mathematical expressions or finding where the mathematical symbols are is an essential step.

Many object detection research have offered methods of solving this problem. For example, Fast R-CNN [4] and Faster R-CNN [5] both solve the detection problem. But the downside is that they require a huge computational power and time. Developing handwritten formula detector using these architectures may not be realistic. One-shot detectors like YOLO [6] and Single Shot MultiBox Detector [7] are more suitable for the purpose

We decided to use ScanSSD [8] as the formula detection architecture. ScanSSD is a typesetting formula detector proposed by Mali et al. It has a precision of 0.848 on typesetting inputs, which is high enough for our purpose. The acronym SSD stands for Single Shot MultiBox Detector, which is an object detector that achieves precise and fast computations. The idea of SSD and ScanSSD is that they apply structures called multi-box detectors. The multi-box detectors can detect formulas at each location of the image with the results stitched together. At last, the pooling process selects the best bounding box as the network's output.

The reason we choose a SSD-based detector is that it consists of multi-box detectors of various sizes, so it can adapt to the shape of formula. Mathematical formulas are various in their aspect ratio. Sliding detectors with different shapes can capture this nature well and output better results.

The precision rate is important because the network's purpose is to provide output to existing state-of-the-art formula recognizers like Mathpix, which will generate corresponding latex code. The false positives that are sent to these recognizers do not result in generating harmful latex code.

2 Method

2.1 Data

To prepare data for training, we collected 39 notes from our friends who take notes on the iPad. These notes are taken in 3 semesters, on 4 subjects, and written by 6 students. These notes are stored in PDF format. In order to annotate the position of mathematical formulas and expressions on the notes, we used a package of Python called `pdf2image` to convert each

page of the notes into a 600 dpi image, which is the same as the dpi of images used in ScanSSD. Although we can convert these images into gray-scale images in order to save some memory usage, it becomes unnecessary since ScanSSD will convert these images into gray-scale within the network.

After converting the notes into images, we used Hasty.ai[9] to annotate the positions of math formulas, as shown in Figure 1. The bounding boxes should be adjoined to the outer-most stroke of math symbols. However, we cannot make sure that all bounding boxes are annotated perfectly due to human error. So we annotated 174 images with 3211 annotations. Then, partition program is designed to randomly assign images into a training set and a validation set. We decided to use 15% of the images as a validation set, and the rest 85% images as a training set.

After annotating all images, annotation results are exported as COCO[10] annotation format, which is a JSON file that saves the size and location of the bounding box corresponding to the image that contains the box. Although COCO is a common format used in the object detection area, ScanSSD did not embrace this format. Instead, it uses GTDB annotation format, which is a CSV file. Therefore, the COCO format is converted into GTDB format using a program. Lastly, we use the script provided by the author of ScanSSD to split annotations on the same page (image) into their own file instead of storing all annotations into one file.

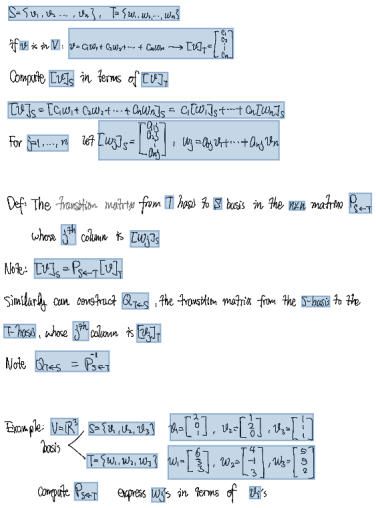

Figure 1: An annotated image with bounding box

2.2 Algorithm and Program

[1] We took advantage of the existing architecture, ScanSSD, which is the network structure we trained on. Although the author published pre-trained weights, their training samples only contain typesetting documents, which belong to a different population then what our model will face. This means that simply using their weights provided by the author will not produce good results. In fact, we tested the weights trained by the authors on our handwriting dataset. The precision is 0.303, which is extremely low. So we decided to retrain the weights with our own handwritten dataset.

Since the original code published by the author works only with PyTorch 1.3.0 or lower, we created a Conda virtual environment with PyTorch 1.3.0 and Python 3.7 on an machine running Ubuntu 20.04. We first used the Intel CPU to train the network. However, due to the CPU's slowness, we switched to an NVIDIA RTX 2070-Super. This graphic card has 8GB of memory, allowing us to use a larger training batch size.

[1] Source code: https://github.com/mrr1vfe/cs539-math-symbol

Model	IOU	Batch size	Precision	Recall	F-score
ScanSSD	0.75	16	0.304	0.292	0.298
ScanSSD (after transfer learning)	0.75	12	0.731	0.553	0.629
ScanSSD (after transfer learning)	0.5	8	0.801	0.660	0.723

Table 1: A comparison between precision and recall on ScanSSD with typesetting weights and weights after transfer learning with different IOU threshold and batch size.

We used the training script provided by the author of ScanSSD. While the original ScanSSD is trained with an IOU threshold of 0.75, we decided to train the network using both the IOU threshold of 0.5 and 0.75. We also differentiated the training batch size, one with a batch size of 12 and one with the batch size of 8. We could not train the network using a batch size of 16, which is the one used by the authors of ScanSSD, since we do not have enough video memory. The other hyper-parameters follow the same as the one used in the original ScanSSD.

2.3 Results and Discussion

Before we decided to apply transfer learning by using our dataset to keep training weights trained with typesetting dataset, we first tested and visualized the performance of vanilla ScanSSD on handwritten texts. A part of the visualization are shown on the left of Figure 2. Clearly, some formulas are not detected and one single formula is split in half. Hence, we decided to keep training ScanSSD. We believe that keep training the weights is a plausible way to improve the accuracy since there are similarities between handwritten formulas and typesetting formulas. Features extracted by the weights of original ScanSSD are still useful.

As mentioned above, we trained the network twice. The first one was trained with a batch size of 12 and an IOU threshold value of 0.75. The other was trained with a batch size of 8 and an IOU threshold of 0.5. During our training, the trend of how the loss is changed via the visualization tool called Visdom[11], which plot the latest loss on the graph. After 150,000 iterations, around 6 epochs, we noticed that the loss was not decreasing, so we used weights at that iteration to test the result. Comparing the evaluation results of the two networks, we found that the one trained with the IOU threshold of 0.5 has a better performance. We got a precision of 0.8 and a recall rate of 0.66 from our validation set. This means that among all formulas we detected, 80% of them are actually formulas. The downside is that we are not able to detect many formulas. This might be due to the fact that our dataset is extremely small comparing to the TFD-ICDAR 2019[12] dataset, used by the original ScanSSD, which contains 805 pages of notes, with 38281 mathematical expressions in total. It can be seen on the right of Figure 2 that our detector can find mathematical formulas and symbols in a relatively complicated environment. More combinations of hyper-parameters were not tested for 2 reasons: 1. training takes a very long time. 2. Modifying other hyper-parameters would not allow us to keep training the weights provided by the authors of ScanSSD.

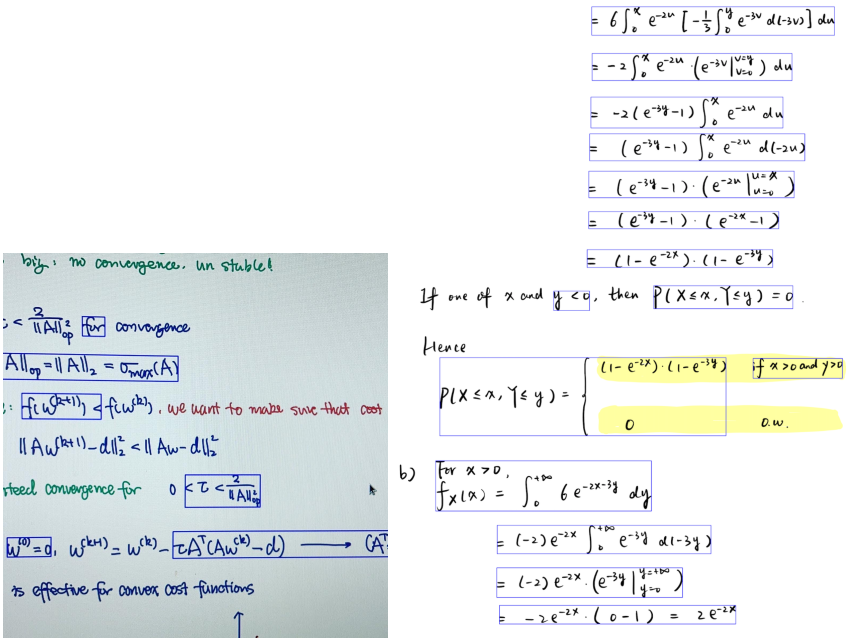

Figure 2: Left: ScanSSD with pre-trained weights (trained using typesetting dataset) miss a lot of formulas. Right: ScanSSD after re-training can find mathematical formulas and symbols in a relative complicated environment

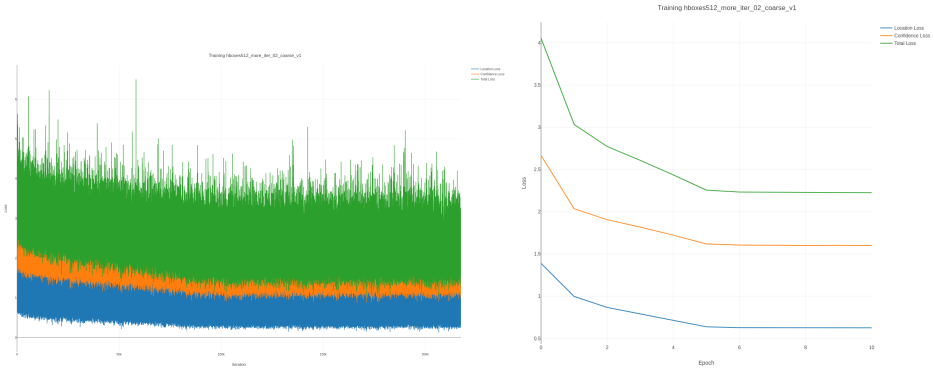

Figure 3: Loss at iterations and epochs

The result showed on the left of Figure 4 is a good detection. The detector successfully separates these two mathematical expressions. Notice that the word "and" between these two mathematical expressions is not as cursive as the word "if" showed on the right, where the detector thinks the word if is also a math symbol. This means that our detector has learned the difference between normal handwriting and math symbols but not cursive writing and math symbols since many math symbols are also written in a cursive manner.

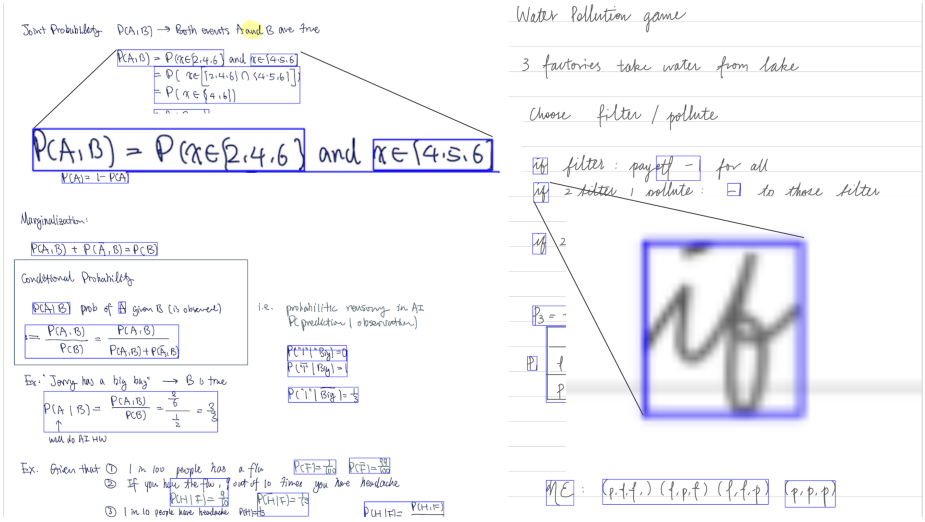

Figure 4: Cursive words are detected as mathematical symbols

Another problem of the detection is it sometimes detects two equations in a single box which is shown on the right of Figure 5. Regardless, results like this are still acceptable since Mathpix can successfully recognize these two lines of equations.

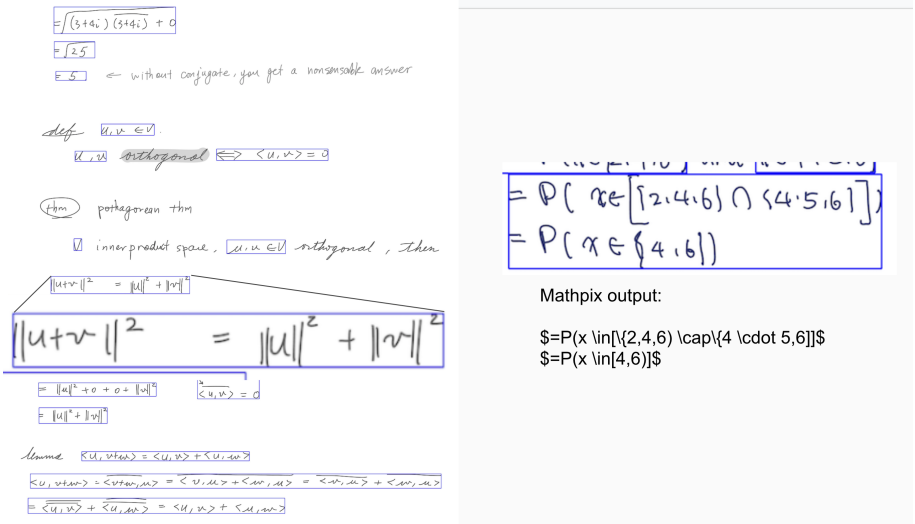

Figure 5: The network considers these two lines of mathematical expression as a single one.

Furthermore, the cursive/math symbol problem can be diminished as increase the size of training data. Also, because our dataset are collected from a few students, the variations on writing styles can potentially result bad recognition in our model. A subsequent improvement will be importing user's own writing scripts or notes, supplementing and enhancing the trained model, which can improve the performance by targeting more on user's own writing style.

Another aspect that hinder us is that obtaining training data is extremely time-consuming. In addition, the manual annotations of every math formula from each note generally takes about five minutes. Applying a faster annotation tool and increasing the training dataset can, therefore, achieve a higher precision rate.

3 Conclusion

In this project, we tested the performance of ScanSSD on handwritten formula dataset. After transfer learning, we showed that ScanSSD is a design that has the ability to work well on handwritten inputs. Note-taking applications can incorporate ScanSSD with weights trained with handwritten dataset to detect the formulas' locations. Then, they can use existing state-of-the-art recognizer to convert mathematical formulas into searchable objects, allowing users to search formulas through their notes. A problem of ScanSSD is that ScanSSD uses VGG[13] architecture as the feature extraction component. However, VGG model may not be computationally tractable to a CPU, which might be the only computational unit on a note-taking device. Therefore, finding a less computational consuming alternative to VGG is a future research direction.

References

[1] Notability. https://www.gingerlabs.com.

[2] Jianshu Zhang, Jun Du, and Lirong Dai. Multi-scale attention with dense encoder for handwritten mathematical expression recognition. In *2018 24th international conference on pattern recognition (ICPR)*, pages 2245–2250. IEEE, 2018.

[3] Mathpix. https://mathpix.com.

[4] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[5] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[6] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[7] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[8] Parag Mali, Puneeth Kukkadapu, Mahshad Mahdavi, and Richard Zanibbi. Scanssd: Scanning single shot detector for mathematical formulas in pdf document images. *arXiv preprint arXiv:2003.08005*, 2020.

[9] Hasty.ai. https://hasty.ai.

[10] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2014.

[11] Visdom. https://ai.facebook.com/tools/visdom.

[12] Parag Mali, Puneeth Kukkadapu, and Mahshad Madhavi. Tfd-icdar 2019: A dataset for typeset math formula detection, 2019. https://github.com/MaliParag/TFD-ICDAR2019.

[13] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.