ScanSSD-XYc: Faster Detection for Math Formulas

Abhisek Dey[0000−0001−5553−8279] and Richard Zanibbi ✉[0000−0001−5921−9750]

Rochester Institute of Technology, Rochester NY 14623, USA
{ad4529,rxzvcs}@rit.edu

Abstract. Detecting formulas offset from text and embedded within textlines is a key step for OCR in scientific documents. The Scanning Single Shot Detector (ScanSSD) detects formulas using visual features, applying a convolutional neural network within windows in a large document image (600 dpi). Detections are pooled to produce page-level bounding boxes. The system works well, but is rather slow. In this work we accelerate ScanSSD in multiple ways: (1) input and output routines have been replaced by matrix operations, (2) the detection window stride (offset) can now be adjusted separately for training and testing, with fewer windows used in testing, and (3) merging with non-maximal suppression (NMS) in windows and pages has been replaced by merging overlapping detections using XY-cutting at the page level. Our fastest model processes 3 pages per second on a Linux system with a GTX 1080Ti GPU, Intel i7-7700K CPU, and 32 GB of RAM.

Keywords: math formula detection · Single-Shot Detector (SSD)

1 Introduction

ScanSSD[4] by Mali *et. al* detects math formula regions as bounding boxes around formulas embedded in text or offset using a Convolutional Neural Network (CNN). It was based on the original SSD [2] which is a single pass network with a VGG-16 [8] backbone to locate and classify regions. Unlike SSD which was made to detect objects in the wild, a number of modifications allowed ScanSSD to obtain strong performance for math formula detection.

Figure 1 shows an overview of the detection process in ScanSSD. To mitigate the issue of low recall on large PDF images and non-square aspect ratios, ScanSSD uses a sliding window detection strategy. A 1200×1200 window was used to stride across the PDF images generated at 600dpi to generate sub-images resized to 512×512 as the input to the network. A stride of 10% or 120 pixels across and down was used as the striding factor. Detections from these windows are passed through a second stage, comprised of a page-level pixel-based voting algorithm to determine the page-level predictions. Pixels vote based on the number of detection boxes they intersect, after which pixel scores are binarized, with formula detections defined by connected components in the binarized voting grid. Initial detections are then cropped around connected components in the original page image that they contain or intersect.

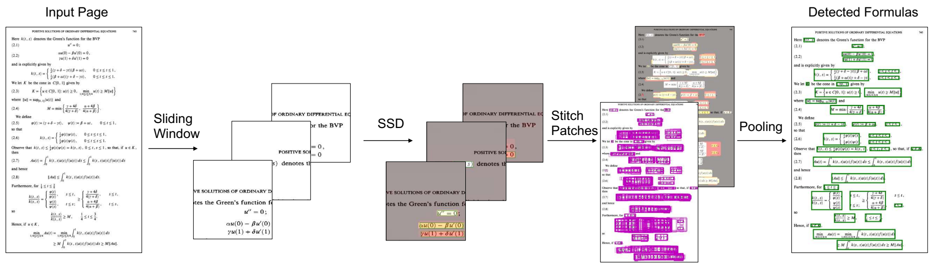

Fig. 1. Sliding window-based formula detection in ScanSSD.

ScanSSD obtains state-of-the-art results with a 79.6 F-score (IoU 0.5) on the TFD-IDCDAR2019v2 dataset, but is quite slow. In this work we have made changes to ScanSSD that allow us to retain comparable accuracy, while decreasing execution times by more than 300 times. Details are provided below.

2 ScanSSD-XYc

ScanSSD-XYc streamlines detection and eliminates redundant operations in ScanSSD, leading to decreases in execution and training times. Substantial changes were made across the pre-processing, windowing and pooling stages, as discussed below.

Pre-Processing. A major bottleneck in ScanSSD was the pre-processing stage, where all windows were generated as separate files. Furthermore, inefficient loading and parallelization resulted in the GPU waiting for batches. To address this, the I/O framework was completely revised. Windows for a page are now generated using a single tensor operation applied to the page image. Ground-truth regions that wholly or partially fall within windows are also identified by another tensor operation. This decreased execution times by 28% over ScanSSD, before incorporating the additional modifications discussed below.

In the original ScanSSD windowing algorithm, many windows used in training contain no ground-truth regions. SSD detects objects at multiple scales, using a grid of initial detection regions at each scale. This leads to thousands of SSD candidate predictions in a single window. Even where target formulas are present in a window, the vast majority of candidate detections are true negatives. To reduce this imbalance for negative examples, in addition to hard negative mining windows without ground-truth regions are ignored in training. Page images are also padded at their edges to be an even multiple of 1200 pixels high and wide, matching the area covered by the fixed-size sliding window.

Windowing. ScanSSD uses a fixed 10% stride for training and inference; we instead use a smaller 5% stride in the training stage to see more examples

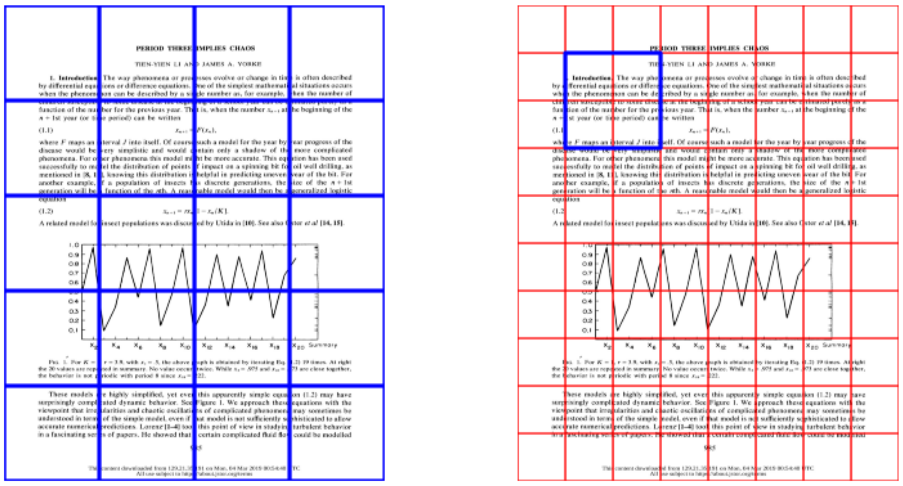

Fig. 2. Left: blue lines show regions for 100% strides of a 1200×1200 window, with each sub-region seen once by the network. **Right:** red lines illustrate regions at left split into four by 50% strides of the same 1200×1200 window (shown in blue). Except for cells around the border, each sub-region indicated by the red lines is seen four times.

of the same target formula alongside other page contents, and then run testing/inference using large strides (e.g., 75%) for faster detection. A stride of 60 pixels (5%) was used for training, and differing strides from 10% to 100% were used for execution. As shown in Figure 2, smaller strides enable the network to see sub-regions multiple times, and improve network convergence. The page edge regions are seen only once, but predominantly consist of white margins.

XY-Cutting to Merge Window-Level Predictions. ScanSSD filters predicted regions twice: once at the window-level and again at the page level. Window-level predictions undergo non-maximal suppression (NMS), and are then stitched together at the page level. Window-level regions vote pixel-wise, producing a vote map over the image which is binarized. Remaining connected components are treated as the final detections. A post-processing step then fits detections tightly around the connected components in the original page image that they intersect. This is an expensive process, illustrated in Figure 1.

As shown in Figure 3, ScanSSD-XYc simply filters window-level detections with less than 50% confidence and then pools detections at page level (Figure 3 left). XY-cutting [1, 5] segments documents into axis-aligned rectangular regions using pixel projections. We use XY-cutting as a partitioning algorithm over detected formula boxes, recursively segmenting the page until regions contain only one set of overlapping boxes. Finally, overlapping boxes are merged to predict the final formula regions (Figure 3 right). Although the worst case time complexity (all overlapping boxes) with n boxes is $O(n^2)$ for both NMS and

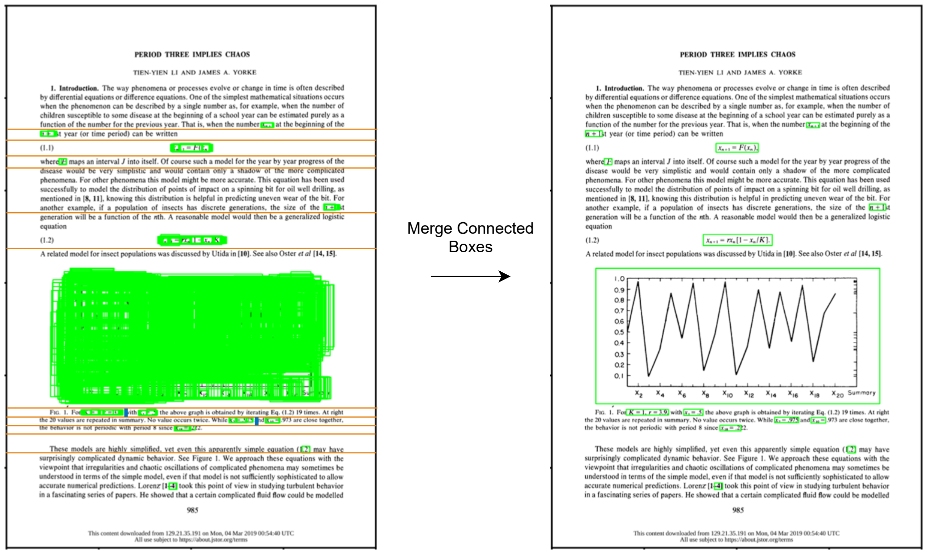

Fig. 3. Left: XY-cutting of Pooled Window-Level Predictions. Orange horizontal lines show splits along the y-axis, and blue vertical lines show splits in the x-direction. Cutting is performed only on detection boxes, and stops when all detection boxes in a region overlap. **Right:** Final page-level predictions after merging overlapping boxes.

XY-Cuts [7], for XY-cut the worst case is rare. XY cuts recursively groups boxes based on spatial positions, and successive splits may be checked independently of one other. In ScanSSD-XYc, cropping is used only for computing evaluation metrics, as blank space at detection borders may be ignored in our applications, but unfairly penalizes detections when using IoU detection metrics.

3 Results

We present preliminary results for ScanSSD-XYc on the TFD-ICDAR2019v2 [3] dataset, which fixes some missing annotations in TFD-ICDAR2019. There are 446 training page images, and 236 testing images taken from 48 PDFs for math papers. Page images are document scans (600 dpi). Experiments were run using a Core i7-7700K CPU with a GTX 1080Ti GPU and 32 GB of RAM.

Table 1 compares ScanSSD-XYc with systems that participated in the ICDAR 2019 CROHME competition [3] and ScanSSD. ScanSSD-XYc performs comparably to the original ScanSSD at 72.4 F-score (IoU of 0.5). RIT 1 was a default implementation of YoloV3, and RIT 2 was an earlier version of SSD [6]. To diagnose the lower recall scores observed for ScanSSD-XYc, we studied the effect of stride size at test-time on accuracy and speed. Table 2 breaks down results by stride size: a smaller stride produces more windows for a page image.

While 100% strides had the smallest number of windows and fastest execution time (3.1 pages/second), a 75% stride gave us the highest accuracy, while still

Table 1. Comparison of Preliminary Results from ScanSSD-XYc (using 75% stride)

Model	IoU 0.5			IoU 0.75		
	Precision	Recall	F-score	Precision	Recall	F-score
TFD-ICDAR2019 (Original)						
ScanSSD	85.1	75.9	80.2	77.4	69.0	73.0
RIT 2	83.1	67.0	75.4	75.3	62.5	68.3
RIT 1	74.4	68.5	71.3	63.2	58.2	60.6
Mitchiking	36.9	27.0	31.2	19.1	13.9	16.1
TFD-ICDAR2019v2						
ScanSSD	84.8	74.9	79.6	78.1	69.0	73.3
ScanSSD-XYc	84.8	63.1	72.4	77.2	57.4	65.9

Table 2. ScanSSD-XYc Results on TFD-ICDAR2019v2 for Different Strides

Stride (%)	IoU 0.5			IoU 0.75			Secs/Page
	Precision	Recall	F-score	Precision	Recall	F-score	
10	41.6	63.1	50.2	36.4	55.3	43.9	28.4
25	79.1	62.6	69.9	71.5	56.6	63.1	4.8
50	83.0	**63.4**	71.9	75.3	**57.6**	65.2	1.3
75	**84.8**	63.1	**72.4**	**77.2**	57.4	**65.9**	0.6
100	80.7	61.4	69.7	71.0	54.0	61.3	**0.3**

processing 1.56 pages/second. ScanSSD-XYc using the smallest stride (10%) is over three times (300%) faster than ScanSSD using the same stride, which takes 90 seconds/page.

As seen in Table2, the 75% stride produces the best balance between speed and accuracy, roughly matching the precision of the original ScanSSD, with some reduction in recall (10.8% for IoU of 0.5, 11.6% for IoU of 0.75). Opportunities to increase recall are described in the next Section. In Table 2 we see little change in recall for different strides, while precision varies. Furthermore, the presence of figures in the evaluation set impacted precision, as these were detected as math (i.e., false positives). The training set contains 9 of 36 documents with at least one figure, while the test set contains 4 of 10 documents with at least one figure. Non-text regions such as figures and tables are often detected wholly or in part by ScanSSD-XYc (see Figure 3), and we plan to address this in future work.

We expected the smallest stride to perform best, but smaller strides produce more partial predictions, which are fit less tightly around targets at page level, and result in more false positives.

4 Conclusion

We have presented an accelerated version of the Scanning Single Shot Detector in this paper. Combining window and page level operations, and eliminating NMS and post-processing led to much shorter execution times with some loss in recall. The entire system was refactored to facilitate faster, scalable tensorized I/O operations. Our long-term goal is to improve the usability of SSD-based detection for use in formula indexing for retrieval applications. We hope to improve detection effectiveness by exploring causes of low recall for small formulas, and issues with over-merging across and between text lines.

We think that setting a relative threshold in the page level XY-cuts based on the sizes of the boxes and detection confidences can mitigate the issue of low recall due to over-merging. We will also attempt to use additional training (beyond two epochs) and different weight initializations and other tuning parameters to better optimize the detector parameters.

Acknowledgements. We thank the anonymous reviewers for their helpful comments, along with Alex Keller, Ayush Shah, and Jian Wu for assistance with the design of ScanSSD-XYc. This material is based upon work supported by grants from the Alfred P. Sloan Foundation (G-2017-9827) and the National Science Foundation (IIS-1717997 (MathSeer) and 2019897 (MMLI)).

References

1. Ha, J., Haralick, R., Phillips, I.: Recursive x-y cut using bounding boxes of connected components. In: Proceedings of 3rd International Conference on Document Analysis and Recognition. vol. 2, pp. 952–955 vol.2 (1995). https://doi.org/10.1109/ICDAR.1995.602059
2. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y., Berg, A.C.: SSD: Single Shot MultiBox Detector. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) Computer Vision – ECCV 2016. pp. 21–37. Springer International Publishing, Cham (2016)
3. Mahdavi, M., Zanibbi, R., Mouchere, H., Viard-Gaudin, C., Garain, U.: ICDAR 2019 CROHME + TFD: Competition on recognition of handwritten mathematical expressions and typeset formula detection. Proceedings of the International Conference on Document Analysis and Recognition, ICDAR pp. 1533–1538 (2019). https://doi.org/10.1109/ICDAR.2019.00247
4. Mali, P., Kukkadapu, P., Mahdavi, M., Zanibbi, R.: ScanSSD: Scanning single shot detector for mathematical formulas in PDF document images. arXiv (2020)
5. Nagy, G., Seth, S.: Hierarchical Representation of Optically Scanned Documents. Proceedings - International Conference on Pattern Recognition pp. 347–349 (1984)
6. Redmon, J., Farhadi, A.: YOLOv3: An incremental improvement (2018), http://arxiv.org/abs/1804.02767
7. Samet, H.: Foundations of Multidimensional and Metric Data Structures. Series on Computer Graphics and Geometric Modeling, Morgan Kaufmann, San Francisco, CA, USA (2005), (XY-cut described in Section 2.1.2)
8. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. In: International Conference on Learning Representations (2015)