TABLE III: Benchmarking ScanSSD at the Character Level [4]. Note differences in data sets and evaluation techniques (see main text).

System	Math Symbol		
	Precision	Recall	F-score
ScanSSD†	0.889	0.965	0.925
InftyReader*	0.971	0.946	0.958
ME U-Net*	0.973	0.950	0.961

* Used GTDB dataset
† Used TFD-ICDAR2019v2 dataset

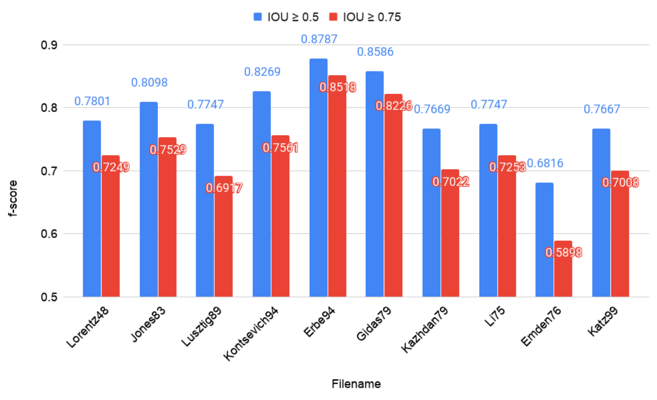

Fig. 7: Document-level results, $IOU \geq 0.5$ and $IOU \geq 0.75$.

Math symbol detection. To measure math detection at the symbol (character) level, we consider all characters located within formula detections as 'math' characters. Our method has 0.9652 recall and 0.889 precision at the character level, resulting in a 0.925 f-score. This benchmarks well against recent results on the GTDB dataset (see Table III). Note that the detection targets (formulas for ScanSSD vs. characters), datasets, and evaluation protocols are different (1000 regions per test page are randomly sampled in Ohayama et al. [4]), and so the measures are not directly comparable. The lower precision for character detection in ScanSSD may be an artifact of predicting formulas rather than individual characters.

The difference betweeen ScanSSD's math symbol detection f-score and formula detection f-score is primarily due to merging and splitting formula regions, which themselves are often valid subexpressions. Merging and splitting valid formula regions often produces regions too large or too small to satisfy the IOU matching criteria, leading to lower scores. Merging occurs in part because formula detections in neighboring text lines may overlap, and splitting may occur because large formulas have features similar to separate formulas within windowed sub-images.

C. *Qualitative results*

Figure 6 provides example ScanSSD detection results. ScanSSD can detect math regions of arbitrary size, from a single character to hundreds of characters. It also detects matrices and correctly rejects equation numbers, page numbers, and other numbers not belonging to formulas. Figure 6 shows some example of detection errors. When there is a large space between characters within a formula (e.g., for variable constraints shown in the third panel of Figure 6), ScanSSD may split the formula and generate multiple detections (shown with pink boxes). Second, when formulas are close to each other, our method may merge them (shown with green boxes in Figure 6). Another error not shown, was wide embedded graphs (visually similar to functions) being detected as math formulas.

On examination, it turns out that most detection 'failures' are because of valid detections merged or split in the manner described, and not spurious detections or false negatives. A small number of these are seen in Figure 6 using red and yellow boxes; note that all but one false negative are isolated symbols.

VII. Conclusion

In this paper we make two contributions: 1) modifying the GTDB datasets to compensate for differences in scale and translation found in the publicly available versions of PDFs in the collection, creating new bounding box annotations for math expressions, and 2) the ScanSSD architecture for detecting math expressions in document images without using page layout, font, or character information. The method is simple but effective, applying a Single-Shot Detector (SSD) using a sliding window, followed by voting-based pooling across windows and scales.

Through our experiments, we observed that 1) carefully selected default boxes improves formula detection, 2) kernels of size 1×5 yield rectangular receptive fields that better-fit wide math expressions with larger aspect ratios, and avoid noise that square-shaped receptive fields introduce.

A key difference between formula detection in typeset documents and object detection in natural scenes is that typeset documents avoid occlusion of content by design. This constraint may help us design a better algorithm for non-maximal suppression, as the original non-maximal suppression algorithm is designed to handle overlapping objects. Also, we would like to use a modified version of the pooling methods based on agglomerative clustering such as the fusion algorithm introduced by Yu et al. [34]. We believe improved pooling will reduce the number of over-merged and split detections, improving both precision and recall.

In our current architecture, we use a fixed pooling method; we plan to design an architecture where we can train the model end-to-end to learn pooling parameters directly from data. ScanSSD allows the use of multiple classes, and we would also like to explore detecting multiple page objects in a single framework.

Acknowledgements. This material is based upon work supported by the Alfred P. Sloan Foundation under Grant