An Integrated Approach of Deep Learning and Symbolic Analysis for Digital PDF Table Extraction

Mengshi Zhang*
University of Texas at Austin
Austin, TX, USA
mengshi.zhang@utexas.edu

Daniel Perelman, Vu Le, Sumit Gulwani
Microsoft
Redmond, WA, USA
{danpere, levu, sumitg}@microsoft.com

Abstract—**Deep learning has shown great success at interpreting unstructured data such as object recognition in images. Symbolic/logical-reasoning techniques have shown great success in interpreting structured data such as table extraction in webpages, custom text files, spreadsheets. The tables in PDF documents are often generated from such structured sources (text-based Word/LaTeX documents, spreadsheets, webpages) but end up being unstructured. We thus explore novel combinations of deep learning and symbolic reasoning techniques to build an effective solution for PDF table extraction. We evaluate effectiveness without granting partial credit for matching part of a table (which may cause silent errors in downstream data processing). Our method achieves a 0.725 F_1 score (vs. 0.339 for the state-of-the-art) on detecting correct table bounds—a much stricter metric than the common one of detecting characters within tables—in a well known public benchmark (ICDAR 2013) and a 0.404 F_1 score (vs. 0.144 for the state-of-the-art) on our private benchmark with more widely varied table structures.**

I. Introduction

PDF is a widely used file format due to its compatibility across various platforms. This is achieved via a complete description of a fixed-layout flat document (including text, fonts, vector graphics, etc.) which, unfortunately, makes it difficult to extract any higher-level structure. There is a huge demand to extract tabular data from PDFs for any further processing/analysis (PDF table extraction was one of top voted requests in Microsoft Power Query [1]). This problem is especially complicated because a PDF document can be generated from a Microsoft Word or LaTeX document, an Excel spreadsheet, an HTML web page, or an Adobe InDesign document, where each one has its own specific table styles.

Deep Learning (DL) and in particular Convolutional Neural Networks (CNNs) are a very effective technique for object localization and recognition in images. With minimal human effort, given a training set, myriad object types can be detected in photographs. This success has led to using similar techniques to detect objects in structured images, including multiple papers on detecting tables, which report high (95%+) accuracy as measured with the standard IOU (Intersection Over Union) metric for object detection in images [2], [3], [4], [5], [6]. (The ICDAR 2013 contest rules [7] score on IOU over characters, not pixels, but the distinction is minor.)

The IOU metric allows for some fuzziness on the detected object as in a photograph pixel-precise bounds are unimportant and is also differentiable which is important for DL algorithms. For example, in Fig. 4, the red predicted table bounding box cuts through the bottom row; it is confused about whether the row should be included. A table detection algorithm that consistently omits the bottom row of 100 row tables would score as 99% accurate despite being useless for downstream processing. In fact, extracting all but that last row of a table with high confidence is often *worse* than not detecting a table at all, so we propose instead using the binary metric of whether the entire table is correct for evaluating table detection algorithms. This choice was encouraged by the Microsoft Power Query team who said they could not ship a feature that would silently omit part of a customer's data. The state-of-the-art table detection algorithm DeepDeSRT [6] which is reported to have an F_1 score of 0.9677 on the ICDAR 2013 table competition using the IOU metric, instead under the binary metric gets an F_1 score of 0.339. While the IOU metric may have made us think table detection was a solved problem, this stricter metric shows there is a lot of room for improvement.

On the other hand, symbolic/logical-reasoning techniques have been very effectively used to extract tables from a variety of semi-structured documents including webpages [8], custom text/log files [9], and spreadsheets [10]. These techniques synthesize parsing programs from few user-provided examples or completely automatically [11] by inferring a repetitive pattern in the underlying input document. The PDF domain poses new challenges for such techniques due to lack of any explicit structure (in glyph-based input) and due to inherent noise (in pixel-based positions). One of our key contributions is a symbolic algorithm that decomposes the problem of table identification into identifying a series of hierarchical substructures, each of which can be efficiently inferred. However, such inference rules become increasingly difficult to fine-tune when we attempt to precisely define what constitutes a table, which happens automatically in a DL-based approach.

These conflicting concerns motivate a combined methodology that can provide the benefit of precision that symbolic techniques yield by hierarchical identification of sub-structures and the benefit of automated fine tuning that DL-based techniques yield via their grasp of real-world data and noise. Initially, we built a shallow combination: given that DL is

*Mengshi Zhang performed this work as part of his internship with the PROSE team at Microsoft. Now, he is a research scientist at Facebook.

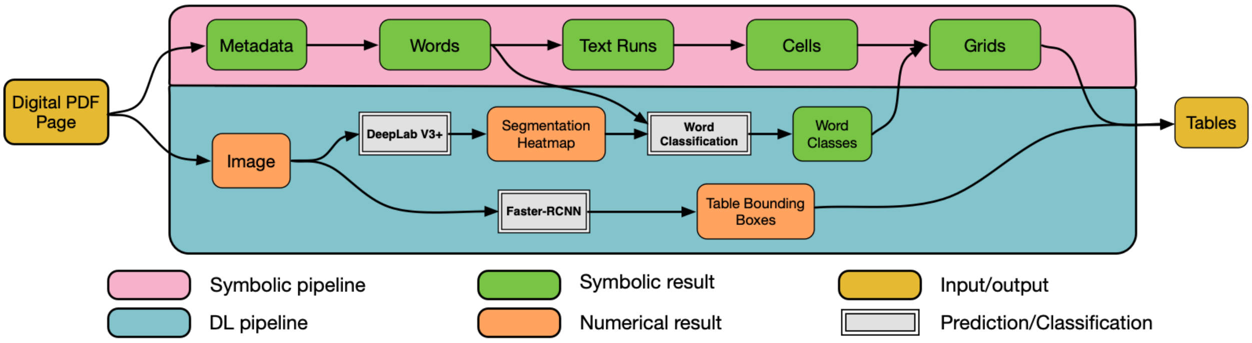


Fig. 1: Workflow of TableExtractor

good at detection but bad at precise localization while the symbolic algorithm is good at precise localization but bad at determining what is and is not a table, we simply filter the results from the symbolic algorithm to those table proposals overlapping a result from the DL algorithm. To achieve further improvements, we looked into deeper integration where the symbolic and the DL pipelines can provide feedback to each other at any stage. For instance, one common failure mode of the symbolic algorithm is to over-expand the detected table to include nearby text which is aligned by coincidence; DL algorithms rarely make this category of mistake, so we added a step where the symbolic algorithm, before building tables out of words, first queries a DL algorithm for whether each word is definitely not part of a table. Specifically, we first use a state-of-the-art semantics segmentation model to predict the heatmap (i.e. whether a pixel belongs to a table word) and classify words by majority of the pixel classes within their bounding boxes. Moreover, we build a novel artificial dataset, TableBank++ with semantic masks to support model training. Our key idea to facilitate deep integration of symbolic and DL algorithms is to design our symbolic algorithm as a pipeline algorithm consisting of a series of steps, where each step computes some set of rectangles—a data structure that can be consumed or refined by any module from the DL pipeline.

Our experimental results show that our approach achieves a 0.725 F_1 score on the ICDAR 2013 dataset and a 0.404 F_1 score on our private dataset with more varied table structures—greatly exceeding the scores of 0.339 and 0.144, respectively, of the state-of-the art DL algorithm DeepDeSRT [6].

In addition, the symbolic algorithm is very lightweight and can run on devices with limited resources. It has been released as the PDF Data Connector in Power Query, a popular analytics offering from Microsoft [12] and is the fastest growing new connector introduced in the past two years; in part due to the greater computational demands of DL inference, the combined approach has not yet been released commercially.

Our paper makes the following contributions:

- We present a novel symbolic algorithm for PDF table extraction that uses a series of inference rules to hierarchically extract tables from glyphs.
- We present a novel DL-pipeline, parameterized by DL-models trained with a new dataset TableBank++.
- We present a novel combination methodology that facilitates a tight integration of the symbolic and DL pipelines.
- We present an extensive evaluation showcasing that the combined approach outperforms both.

II. Background

A. Dataset

We used these datasets to develop & evaluate our approach:

TableBank++ is built on top of TableBank [13], a benchmark that contains 417k labeled tables. TableBank first collects thousands of Word and LaTeX documents from the Internet as, unlike PDF, those documents explicitly provide the object information (e.g. tables and cells) and can be converted to PDF format. TableBank then instruments the documents to generate two versions of the PDFs with highlighted table boundaries in different colors, and uses basic computer vision techniques to extract the table boxes as labels. Unfortunately, TableBank does not contain enough information to train our semantic segmentation networks. In particular, we also want the label of each pixel as (1) `TabText` (text box in a table), (2) `NonTabText` (text box not in a table), or (3) `NonText` (outside of any text box). To address this issue, for each PDF page, we use the symbolic pipeline of TableExtractor (Section III-A) to extract the text bounding boxes (e.g. Fig. 3b). We use TableBank++ as the training dataset to build DL models.

ICDAR 2013 [7] is a competition dataset of natively-digital PDF documents for table detection and structure recognition. ICDAR 2013 includes a total of 150 tables (i.e. labeled with table and cell bounding boxes): 75 tables in 27 excerpts from the EU and 75 tables in 40 excerpts from the US government. We use ICDAR 2013 as a test dataset to evaluate our approach.

Camelot is an open-source Python library[1] for digital PDF table extraction. We refer to their test suite as the Camelot dataset, which contains 52 PDFs (62 pages) and 70 tables. This dataset is only used to evaluate our approach.

ComplexTab is a private dataset which includes a collection of PDFs we received from customers as part of the requirements gathering stage of developing the symbolic pipeline of TableExtractor. The reason for constructing ComplexTab to

[1] https://github.com/atlanhq/camelot

Glyph g	:=	$\{$string, font$\}$	text metadata from PDF parser
Word W	:=	$\{[g]$, font$\}$	maximal list of "nearby" *Glyphs* that share a font
Text Run R	:=	$[W]$	maximal list of "nearby" *Words* that may have different fonts
Cell C	:=	$[R]$	maximal list of "aligned" *Text Runs*
Alignment A	:=	$\{[C]$, axis$\}$	maximal list of *Cells* that have the "same" position along some axis
Contiguous List L	:=	$\{A, [C]\}$	a maximal uninterrupted subsequence of multiple *Cells* within an *Alignment* (the rectangle defined by $[C]$ may not overlap any other *Cells*)
Grid G	:=	$[[L]]$	maximal list of uninterrupted *Contiguous Lists* in each axis must satisfy that a *Cell* C in the rectangle of G iff $\forall_{a \in G} \exists L \in a\ C \in a$
Table T	:=	$\{[G], [C]\}$	maximal list of overlapping *Grids* plus an optional list of "related" *Cells*

Fig. 2: Grammar forming hierarchical structure on document

evaluate our approach is two-fold. First, ComplexTab provides more tables than ICDAR 2013 dataset: 2737 PDF pages across 64 documents for a total of 4358 carefully labeled tables. Moreover, while tables in ICDAR 2013 are very regular, containing explicit lines and few multi-row or multi-column cells, the tables in ComplexTab are more diverse and complicated.[2] For instance, many tables do not have borders and the columns may have big gaps among them. Thus, ComplexTab is more effective in evaluating the generalizability of our approach.

B. Metric

Our task is identifying the bounding boxes of all of the tables in a given page. As we target PDFs with explicit text information, this is equivalent to selecting all the text in each table, so for the DL parts that output bounding boxes, we snap them to the word bounding boxes for our comparisons. Words here are defined by the glyph bounding boxes directly from the PDF grouped by a heuristic tuned to conservatively combine glyphs close enough to definitely be part of the same word.

Note that while a complete system would include interpreting the table structure, we do not evaluate that in this paper. In practice, our symbolic technique builds a predicted table structure while identifying tables. As DL is not used for that part of our algorithm, it is out of scope for this paper.

III. Approach

Fig. 1 illustrates the workflow of our approach. The system takes a digital PDF page as input and utilizes two interleaved approaches to extract tables. In this section, we first define the key concepts and introduce the workflow of the symbolic approach. Next, we present the details of the deep-learning-based approach. Finally, we introduce the combination of them for accurate table extraction.

A. Symbolic Algorithm

The symbolic algorithm is a **bottom-up** approach that builds a hierarchy (Fig. 2) starting with *Glyphs* all the way up to *Tables*. The key observation is that a table primarily is a grid of cells that are aligned both horizontally and vertically, and that such spacing and alignment is unlikely to be observed in text blocks that are not tables. Leveraging this observation is not straightforward because defining "cell" and "aligned" symbolically is difficult and tables often have more complicated structures involving merged cells, hierarchical headers, and other formatting that is not a straightforward grid of cells.

The objects in the grammar in Fig. 2 (Fig. 3 shows visualizations for them on an example) share the following properties:

Property 1: They are all rectangles with additional metadata. This allows for straight-forward communication with a DL algorithm which can also report its results as rectangles or consume rectangles as inputs.

Property 2: The number of instances of each object is tightly bounded (mostly linear; follows from the maximality constraints in the definitions), which means the number of objects the algorithm has to deal with never gets too large.

Words, *Text Runs*, *Cells*, and *Tables* are all partitionings of the next smaller object (e.g. each *Glyph* is in exactly one *Word*). *Alignments* can be thought of 6 (non-singleton) partitionings of *Cells*, corresponding to the 6 coordinates: the two edges and the center along each axis

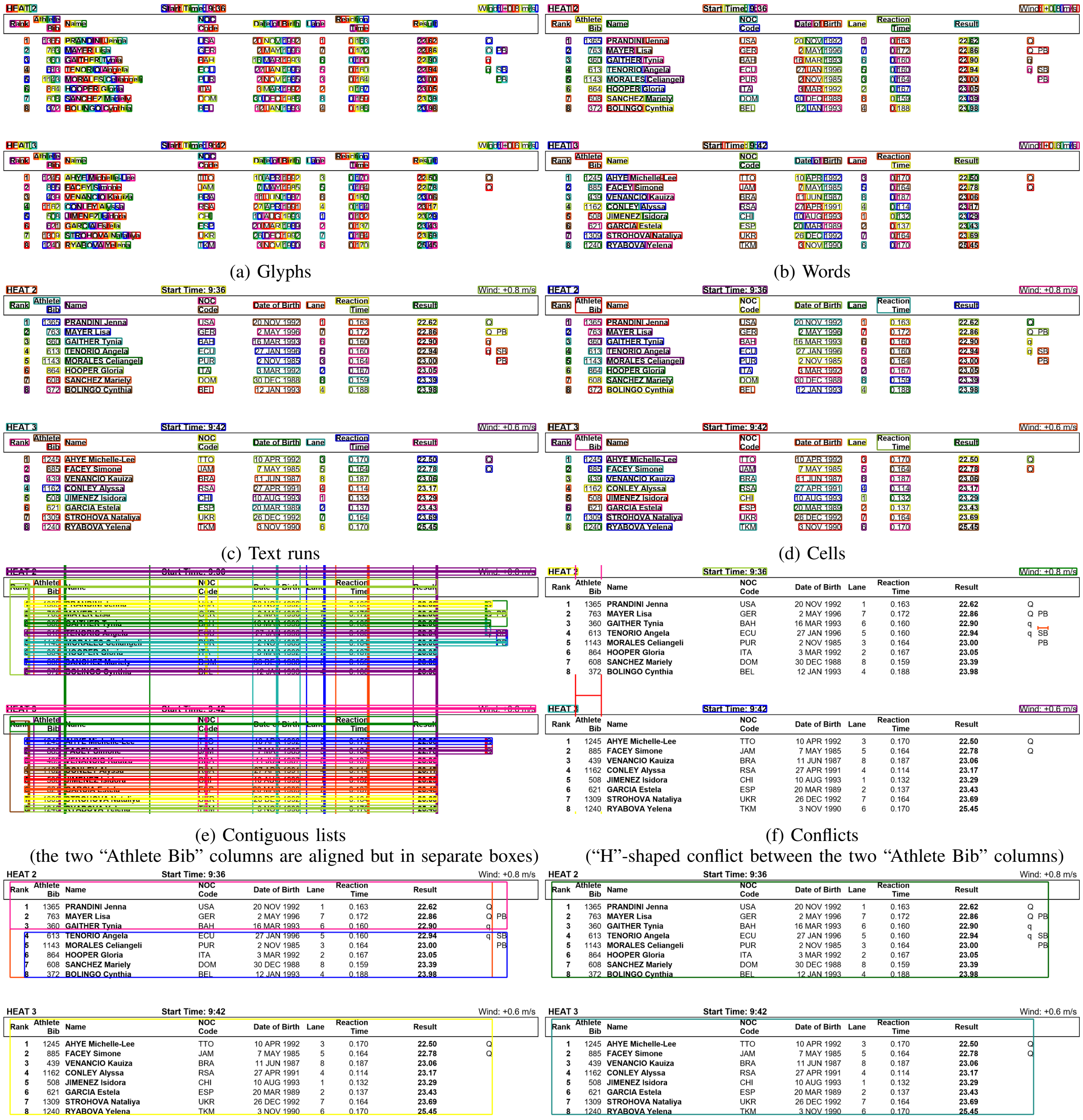
Contiguous Lists and *Grids* are both bounded by the square of the number of *Cells*; we omit the proofs for space reasons.

The grammar has some undefined terms in quotes like "nearby" and "same" where our manually authored heuristics come in. While we had to choose that, for example, two *Cells* have the "same" (left, center, etc.) position for deciding if they they belong in the same *Alignment* if their positions differ by no more than half the width of a character, we do not think there are any deep insights hidden in these heuristic values; we just happened to learn them through hand-tuning. There is ongoing work on learning such heuristics automatically [14].

Additionally we gloss over many engineering details: e.g. two *Glyphs* are not "nearby" to form a *Word* if there is a border line between them. But that border line could be specified in many different ways involving PDF paths or images that our algorithm normalizes into a single concept of a line.

While not required by this formulation of the problem, our algorithm is deterministic and selects a single answer for, say, identifying the *Words*, before grouping them into *Text Runs*, although one could imagine a probabilistic algorithm that instead proposes multiple possibilities that were carried through the rest of the algorithm. We made our choice both for simplicity of implementation and for speed. Alternatively, any of the symbolic phases can be replaced or augmented with

[2] https://library.olympic.org/Default/doc/SYRACUSE/165312 for example



(a) Glyphs

(b) Words

(c) Text runs

(d) Cells

(e) Contiguous lists
(the two “Athlete Bib” columns are aligned but in separate boxes)

(f) Conflicts
(“H”-shaped conflict between the two “Athlete Bib” columns)

(g) Grids

(h) Tables

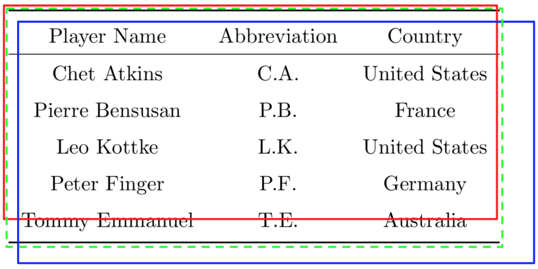
Fig. 3: Visualizations of selected steps of the symbolic algorithm

a different algorithm, perhaps powered by DL, given that our process is about passing rectangles thorugh a series of phases.

The actual algorithm consists of a series of passes that takes the objects from the previous phase, does a pass over them in order top-to-bottom and/or left-to-right, and while doing so generates the objects for the next phase. The more interesting steps are those to generate *Cells*, *Grids*, and *Tables*.

For *Cells*, when not clearly defined by border lines, multiple interpretations are attempted by making multiple hypotheses about the line spacing between *Text Runs* and checking if any of those hypotheses lead to a consistent identification of rows. If both heuristics fail, then each *Cell* is just a single *Text Run*.

As an optimization, there is an intermediate object between *Contiguous Lists* and *Grids*, called “*Conflicts*” which are rect-



Player Name	Abbreviation	Country
Chet Atkins	C.A.	United States
Pierre Bensusan	P.B.	France
Leo Kottke	L.K.	United States
Peter Finger	P.F.	Germany
Tommy Emmanuel	T.E.	Australia

Fig. 4: Example of Table Detection

angles that define the maximal edges of *Grids*. Any *Cell* that belongs to no *Contiguous List* is a *Conflict* as the rectangle of a *Grid* may never overlap the rectangle of such a *Cell*. Also, the space between any two aligned *Contiguous Lists* is a *Conflict*: if a *Grid* overlaps that entire space along the axis of those *Contiguous Lists*, then it overlaps two *Contiguous Lists* that were intentionally separated due to having some "interruption" between them. Given *Conflicts*, computing *Grids* requires only sweeping across the page top-to-bottom keeping track of the rectangles that can be drawn without violating any *Conflicts*.

Finally, for *Tables*, first note that the choice of overlapping *Grids* instead of a single *Grid* allows the algorithm to capture more complicated table structures without having to explicitly support them. As long as enough cells are aligned, the rectangle around all of the overlapping *Grids* will capture the table. For headers that may not be aligned with the data, we have heuristics for "related" cells based on following border lines that go through or nearby the rectangle defined by *Grids*.

B. DL-Based Detection and Classification

The DL-based detection and classification is a **top-down** approach that observes a page globally and narrows the scope down to inference. We use two DL models to predict table bounding boxes and classify *Words* defined in Section III-A.

Table Detection: We use the state-of-the-art object detection framework, Faster-RCNN, to predict the table bounding boxes. Faster-RCNN is a two-stage object detector, which includes two distinct modules: (1) Region Proposal Networks (RPN) generate proposals of Region of Interest (RoI) for covering objects and (2) Region-based CNNs (RCNN) take image features and proposals as input to classify the content in proposals and to adjust the bounding box shapes. Two stage predictor generally has a better prediction accuracy (e.g. higher mean Average Precision) but suffers low speed. On the other hand, one-stage object detectors (e.g. YOLO [15]) have higher inference speed but their mAP is relatively low. We choose Faster-RCNN because we think the prediction accuracy is more important for our scenario. The quality of table detection is very sensitive to the accuracy of predicted bounding boxes, where missing or extraneous text (e.g. missing the last row) will greatly hurt the reliability of the extracted tables. Moreover, unlike common objects, the aspect ratio range of tables is very large (e.g. narrow and long vs. wide and short tables), which is challenging for detectors.

Country	Abbr.
China	CN
Vietnam	VN
Japan	JP
Thailand	TH

(a) Asian Countries

Country	Abbr.
United Kingdom	UK
Germany	DE
France	FR
Spain	ES

(b) European Countries

Fig. 5: Example of Side-by-side Tables

Word Classification: Bounding boxes play a key role in locating tables, however, they are predicted on the pixel-level and such results may not be suitable for the table extraction task. Fig. 4 shows a toy example of table detection using Faster-RCNN. In the figure, the green dashed box is the ground truth and the blue and red boxes are two predicted candidates with confidence scores `0.85` and `0.9`, respectively. As the overlap of the blue and red boxes is greater than the Non-Maximum Suppression [16] threshold and the red box's confidence is higher, Faster-RCNN ignores the blue one and only outputs the coordinates of the red box. In addition, this prediction also has a better IoU (Intersection Over Union) score since the red-green and blue-green intersection areas are similar and the blue-green union area is larger. However, from the figure, we can observe that the bottom line of the red box cuts through the last row and is confused as to whether these words should be included in table. On the other hand, the blue box includes all the words and it is more practical for our task.

To address the above conflict, we introduce a new DL model for finer-grained prediction. We use the extra model to predict whether the words belong to tables or not, a task the symbolic algorithm can have difficulty with in the presence of columns or otherwise coincidentally lined up text in paragraphs. As showed in the blue block of Fig. 1, DeepLab V3 [17] takes an image as input and outputs a segmentation heatmap. The heatmap contains three channels: (1) `TabText`, (2) `NonTabText`, and (3) `NonText`, which are defined in Section II-A. Next, a word classification module investigates the heatmap pixels in each word bounding box and takes the majority as the class, and the results of both table detection and word classification are incorporated into the symbolic pipeline to help improve the extraction accuracy.

Word classification provides finer-grained results, and it is intuitive that we can detect tables by just grouping adjacent `TabText` words. We may be tempted to skip the table detection model to simplify the workflow, but that turns out to only work if there is only one table on the page; the intuitive approach fails on pages with multiple tables. The example in Fig. 5 showing side-by-side tables demonstrates this. The trivial word grouping technique cannot separate them, however, the table detection network can predict multiple objects in the image, which boosts the extraction accuracy.

C. Integrated Approach

The key insight of TableExtractor (Fig. 1) is the symbolic technique is great at finding all parts of the page that are well-structured and aligned but has to rely on the DL-based

TABLE I: Faster-RCNN architecture

Module	Component	Hyperparameter
Backbone	Stem	$C(7,64), stride2 + MP(3), stride2$
	Res2	$[C(1,64), C(3,64), C(1,256)] \times 3$
	Res3	$[C(1,128), C(3,128), C(1,512)] \times 4$
	Res4	$[C(1,256), C(3,256), C(1,1024)] \times 23$
	Res5	$[C(1,512), C(3,512), C(1,2048)] \times 3$
FPN	p2 - p5	$C(1,256), C(3,256)$
	p6	$MP(2)$
RPN	head	$C(3,256)$
	objectness	$C(1,6)$
	anchor offset	$C(1,24)$
ROI	head	$FC(,1024) \times 2$
	classification	$FC(,2)$
	bbox regress.	$FC(,4)$

approach for the more vague concept of what parts of the page appear to be part of a table. The integration of the two follows.

First, the *Words* identified by the symbolic technique are classified by the DL-based approach, and those identified as NonTabText are included in the *Conflicts*, thereby omitting them from the *Grids*. This helps avoid over-expanding tables when they contain cells that happen to line up with nearby paragraphs or other similar issues. This two-way communication between the symbolic and DL parts of the approach is the key novelty of our algorithm.

Second, the *Tables* are filtered by whether they overlap with a table found by the DL-based table detection step. This helps avoid including text that is coincidentally lined up as tables while not being impacted by the imprecision of the bounding boxes generated by DL.

IV. Experimental Setup

A. Model Architecture

Faster-RCNN: Table I shows the model architecture with the following definitions: (1) $C(m, N)$ is the 2D convolution with N $m \times m$ kernels, (2) $MP(m)$ is max pooling with a $m \times m$ kernel, (3) $[m_a, m_b, m_c] \times N$ means modules m_a, m_b, m_c are grouped and repeated N times, and (4) $FC(, m)$ is a fully connected layer with output dimension m.

The Backbone net is ResNet-101 which is used for generating features. FPN indicates Feature Pyramid Network which is used for fusing Backbone features at different scales. Note that FPN contains 5 modules (i.e. p2 to p6) and the architecture of p2 to p5 are identical. We present their architectures in one row for simplicity. RPN indicates Region Proposal Network which is developed for generating proposals for covering objects and ROI indicates Region of Interest Network which aims to classify object and regress bounding boxes.

Deeplab V3: The model architecture of Deeplab v3 is presented in Table II. The backbone is ResNet-101, which is identical to the backbone of Faster-RCNN. ASPP indicates Atrous Spatial Pyramid Pooling, which is developed to encode multi-scale contextual information using Atrous Convolution. Specifically, $C_d(m, d, N)$ means the Atrous Convolution with N $m \times m$ kernels with dilation size of d. Decoder is a simple Convolutional Network which fuses features & outputs masks.

TABLE II: Deeplab v3 architecture

Module	Component	Hyperparameter
Backbone	ResNet	-
ASPP	Conv	$C(1,256),\ C_d(1,12,256),$ $C_d(1,24,256),\ C_d(1,36,256)$
	Pooling	$adaptive\ avg\ pooling(1)$
Decoder		$C(1,256)$

B. Implementation

We implemented the symbolic algorithm in C# and the DL models in PyTorchand ran the experiment on an Ubuntu 16.04 Azure virtual machine with an Nvidia K40 GPU.

V. Experimental Results

Table III shows experimental results of evaluating TableExtractor on three datasets. In the `Number of Pages` block, the columns `W/ all T.P.` and `W/o F.P.` denote the numbers of the detected pages *with all true positive tables* and *without any false positive tables*, respectively. In the `Number of Tables` block, `Rele.`, `Sele.`, `Overl.`, and `T.P.` denote *Relevant* (ground truth), *Selected* (detected), *Overlapping* (ground truth overlapping any detected) and *True Positive* (ground truth matching detected) tables, respectively. Note `Overl.` may exceed `Sele.` in the case where multiple nearby tables are misidentified as a single table.

We show precision, recall, and F_1 for two metrics: `Localization` is the one we discussed above that requires matching exactly the words in the table (so a bounding box that cuts through a row/column on the edge of a table may be expanded using word boundaries and considered correct); and `Detection` is a very forgiving metric which considers any pixels overlapping the ground truth to be a correct answer. While we do not grade algorithms on the `Detection` metric, it is informative both because it helps identify how the algorithm is failing and because TableExtractor combines table detection results based on any overlap at all. Because `Detection` recall numbers are nearly perfect in every category, only the precision column distinguishes algorithms.

Our story starts with the DL approaches: `TableBank` [13] (the baseline), `DL` (our DL), and `DeepDeSRT` (the state-of-the-art DL). All do poorly on the `Localization` metric: none of the precision, recall, or F_1 scores exceed 0.4 and most of them are below 0.2. They do best on the `ICDAR` dataset, with DeepDeSRT in a clear lead with an F_1 score of 0.339 compared to 0.251 for TableBank and 0.209 for DL. The comparison is tighter but in a different order for `ComplexTab` where DL is in the lead with an F_1 score of 0.143 (resulting from our training using TableBank++), followed by DeepDeSRT at 0.134 and TableBank at 0.123.

Next we look at `Symbolic`, our algorithm that involves no DL. Its F_1 scores are somewhat higher at 0.359 overall and 0.509 for the `ICDAR` dataset, primarily driven by significantly higher recall of 0.418 (0.747 for `ICDAR`) compared to 0.200 (0.371) for the best recall among the DL-based techniques, but looking at the `Detection` precision column compared

TABLE III: Comparison results of the integrated, ML-based and symbolic approaches

	Dataset	Number of Pages			Number of Tables				Detection			Localization		
		Total	W/ all T.P.	W/o F.P.	Rele.	Sele.	Overl.	T.P.	Precision	Recall	$\mathbf{F_1}$	Precision	Recall	$\mathbf{F_1}$
Tablebank	All	3054	1761	557	4598	7445	4522	780	0.493	0.983	0.657	0.105	0.170	0.130
	ComplexTab	2737	1585	448	4358	7045	4283	704	0.481	0.983	0.646	0.100	0.162	0.123
	Camelot	62	14	10	70	84	70	15	0.952	1.000	0.976	0.179	0.214	0.195
	ICDAR	255	162	99	170	316	169	61	0.649	0.994	0.785	0.193	0.359	0.251
Integrated	All	3054	2317	2053	4598	3910	4465	1794	0.796	0.971	0.875	0.459	0.390	0.422
	ComplexTab	2737	2055	1802	4358	3659	4226	1618	0.787	0.970	0.869	0.442	0.371	0.404
	Camelot	62	44	43	70	68	70	48	0.971	1.000	0.985	0.706	0.686	0.696
	ICDAR	255	218	208	170	183	169	128	0.913	0.994	0.952	0.699	0.753	0.725
DL	All	3054	1788	483	4598	7959	4494	919	0.539	0.977	0.695	0.115	0.200	0.146
	ComplexTab	2737	1611	417	4358	7469	4255	843	0.535	0.976	0.691	0.113	0.193	0.143
	Camelot	62	15	9	70	96	70	17	0.938	1.000	0.968	0.177	0.243	0.205
	ICDAR	255	162	57	170	394	169	59	0.515	0.994	0.679	0.150	0.347	0.209
Symbolic	All	3054	2392	1314	4598	6109	4583	1923	0.527	0.997	0.689	0.315	0.418	0.359
	ComplexTab	2737	2131	1136	4358	5695	4343	1748	0.524	0.997	0.687	0.307	0.401	0.348
	Camelot	62	44	36	70	85	70	48	0.776	1.000	0.874	0.565	0.686	0.619
	ICDAR	255	217	142	170	329	170	127	0.514	1.000	0.679	0.386	0.747	0.509
Sym.+TD	All	3054	2333	1802	4598	4542	4479	1830	0.686	0.974	0.805	0.403	0.398	0.400
	ComplexTab	2737	2073	1577	4358	4255	4240	1656	0.677	0.973	0.799	0.389	0.380	0.385
	Camelot	62	44	42	70	74	70	48	0.892	1.000	0.943	0.649	0.686	0.667
	ICDAR	255	216	183	170	213	169	126	0.789	0.994	0.880	0.592	0.741	0.658
Sym.+WC	All	3054	2372	1974	4598	4179	4564	1883	0.769	0.993	0.867	0.451	0.410	0.429
	ComplexTab	2737	2109	1727	4358	3915	4324	1706	0.761	0.992	0.862	0.436	0.391	0.412
	Camelot	62	44	42	70	71	70	48	0.930	1.000	0.964	0.676	0.686	0.681
	ICDAR	255	219	205	170	193	170	129	0.870	1.000	0.931	0.668	0.759	0.711
DeepDeSRT	All	3054	1708	1534	4598	3105	4366	554	0.893	0.950	0.920	0.178	0.120	0.144
	ComplexTab	2737	1530	1382	4358	2818	4126	479	0.893	0.947	0.919	0.170	0.110	0.134
	Camelot	62	13	10	70	85	70	12	0.953	1.000	0.976	0.141	0.171	0.155
	ICDAR	255	165	142	170	202	170	63	0.861	1.000	0.926	0.312	0.371	0.339
$\cdots$+Sym.	All	3054	2352	2344	4598	3507	4439	1840	0.871	0.965	0.916	0.525	0.400	0.454
	ComplexTab	2737	2091	2092	4358	3258	4199	1665	0.866	0.964	0.912	0.511	0.382	0.437
	Camelot	62	44	43	70	70	70	48	0.943	1.000	0.971	0.686	0.686	0.686
	ICDAR	255	217	209	170	179	170	127	0.944	1.000	0.971	0.709	0.747	0.728
$\cdots$+WC	All	3054	2342	2339	4598	3494	4429	1814	0.880	0.963	0.920	0.519	0.395	0.448
	ComplexTab	2737	2079	2083	4358	3248	4189	1637	0.875	0.961	0.916	0.504	0.376	0.430
	Camelot	62	44	43	70	70	70	48	0.943	1.000	0.971	0.686	0.686	0.686
	ICDAR	255	219	213	170	176	170	129	0.955	1.000	0.977	0.733	0.759	0.746

to the DL approaches, it detects tables where there are not any between marginally more often and a lot more often, which leads us to the first version of the integrated approach.

Sym.+TD ($\cdots$+Sym.) uses DL (DeepDeSRT [6]) to filter tables generated by the symbolic algorithm to those that overlap some output from the DL algorithm. The result is a significant improvement in the Detection precision at a small cost in recall: 0.686 (0.871) vs. 0.527. There is a similar improvement in the Localization precision of 0.403 (0.525) vs. 0.315.

Since that approach made the recall worse, we tried a different way to integrate DL into our pipeline, by building a model for word classification used as an intermediate step. We evaluate this algorithm as Sym.+WC. The Localization recall is only slightly worse at 0.410 vs. 0.418 but the precision is better at 0.451 vs. 0.351. Note the recall is actually slightly better on the ICDAR dataset, the word classification information actually allows the combined algorithm to get the correct bounds for two more tables.

Adding both table detection and word classification to the symbolic algorithm gives us Integrated (or $\cdots$+WC when using DeepDeSRT as the table detection filter). We can tell the techniques are complementary as the precision of Integrated exceeds that of Sym.+TD or Sym.WC. Its F_1 is the best for the simpler tables in the Camelot and ICDAR datasets, but for the more complicated tables in the ComplexTab dataset, it is slightly better to use one of the simpler combinations (Sym.+WC or ...+Sym.).

VI. Related work

Several works have been published on the topic of table detection for multiple media: e.g., HTML [18], [19], [20], images [21], [22], [23], [24], [25], spreadsheets [26], text/log files [11], [9] and PDF documents [27], [28]. Generally, these techniques aim to extract tables from surrounding text using logical structure analysis or traditional image processing techniques. Liu et al. [28] proposed a heuristic rule-based approach to detect table boxes in PDFs. Tran et al. [29] use vertical alignments to identify tables. Kim et al. [30] identify grids, using a similar intuition to our grids of combining horizontal and vertical alignments. These prior rule-based approaches assume a regular table structure, lacking equivalents to our use of overlapping grids and related cells. Le et al. [9] proposed an interactive approach (where users provide examples) that leverages inductive synthesis to generate programs for table extraction. Raza et al. [11] proposed a predictive program synthesis technique for text and web table extraction. Due to the great success of DL in computer vision, recent research focuses more on table detection using Convolutional Neural Networks (CNNs). Kavasidis et al. [2] proposed a saliency-based CNN performing multi-scale reasoning on visual cues followed by a fully-connected conditional random field (CRF) to localize tables and charts in digital documents. Schreiber et al. proposed DeepDeSRT[6], a Faster-RCNN based approach for table detection and structure recognition. Dong et al. proposed TableSense [26], which utilizes CNNs and precision bounding box regression to detect tables in spreadsheets.

Kalyan et al. [31] combined deep learning & symbolic methods for program synthesis, wherein deep learning was used to replace some heuristic components in the overall symbolic method. Iyer et al. [32] cycled the learning result between machine learning and program synthesis to produce robust programs with few explicit examples. We showcase a more intricate combination, where the two methods feed into each other, and for a different data extraction application.

VII. Conclusion

PDF is a widely used file format due to its cross-application support. However, PDF does not provide structure information for objects like tables and paragraphs, which limits downstream processing. In this paper, we focus on the problem of digital PDF table extraction and propose an integrated approach combining DL and symbolic techniques to improve the accuracy of table extraction. Also, we propose that, as table detection without accurate localization is not useful for downstream processing, we should evaluate the latter. The experimental results indicate the combined algorithm out-performs both purely symbolic and purely DL-based approaches. We believe that such novel combinations of symbolic/logical-reasoning techniques (which can precisely model the semantics of the underlying domain) with machine learning based methods (which can deal with noise and model the bias in the real world data) will enable new applications.

References

[1] PowerBI, "Microsoft PowerBI forum: "Tables in PDF files"," https://ideas.powerbi.com/forums/265200-power-bi-ideas/suggestions/6973371-tables-in-pdf-files, 2020.

[2] I. Kavasidis, S. Palazzo, C. Spampinato, C. Pino, D. Giordano, D. Giuffrida, and P. Messina, "A saliency-based convolutional neural network for table and chart detection in digitized documents," *arXiv preprint arXiv:1804.06236*, 2018.

[3] D. Augusto Borges Oliveira and M. Palhares Viana, "Fast cnn-based document layout analysis," in *ICCV*, 2017.

[4] X. Yang, E. Yumer, P. Asente, M. Kraley, D. Kifer, and C. Lee Giles, "Learning to extract semantic structure from documents using multimodal fully convolutional neural networks," in *CVPR*, 2017.

[5] Y. Li, Y. Zou, and J. Ma, "Deeplayout: A semantic segmentation approach to page layout analysis," in *International Conference on Intelligent Computing*. Springer, 2018.

[6] S. Schreiber, S. Agne, I. Wolf, A. Dengel, and S. Ahmed, "Deepdesrt: Deep learning for detection and structure recognition of tables in document images," in *ICDAR*, vol. 1. IEEE, 2017.

[7] M. Göbel, T. Hassan, E. Oro, and G. Orsi, "Icdar 2013 table competition," in *ICDAR*. IEEE, 2013.

[8] M. Raza and S. Gulwani, "Web data extraction using hybrid program synthesis: A combination of top-down and bottom-up inference," in *SIGMOD*. ACM, 2020.

[9] V. Le and S. Gulwani, "Flashextract: a framework for data extraction by examples," in *ACM SIGPLAN Notices*, vol. 49, no. 6. ACM, 2014.

[10] D. W. Barowy, S. Gulwani, T. Hart, and B. G. Zorn, "FlashRelate: extracting relational data from semi-structured spreadsheets using examples," in *PLDI*, 2015.

[11] M. Raza and S. Gulwani, "Automated data extraction using predictive program synthesis," in *AAAI*, 2017.

[12] PowerBI, "Connect to PDF files in power BI desktop," https://docs.microsoft.com/en-us/power-bi/connect-data/desktop-connect-pdf, https://aka.ms/prose-pdf, 2020.

[13] M. Li, L. Cui, S. Huang, F. Wei, M. Zhou, and Z. Li, "Tablebank: Table benchmark for image-based table detection and recognition," *arXiv preprint arXiv:1903.01949*, 2019.

[14] N. Natarajan, A. Karthikeyan, P. Jain, I. Radicek, S. Rajamani, S. Gulwani, and J. Gehrke, "Programming by rewards," 2020.

[15] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *CVPR*, 2016.

[16] A. Neubeck and L. Van Gool, "Efficient non-maximum suppression," in *ICPR*, vol. 3. IEEE, 2006.

[17] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv preprint arXiv:1706.05587*, 2017.

[18] J. Wang, H. Wang, Z. Wang, and K. Q. Zhu, "Understanding tables on the web," in *International Conference on Conceptual Modeling*, 2012.

[19] Y. Zhai and B. Liu, "Web data extraction based on partial tree alignment," in *WWW*. ACM, 2005.

[20] Y. Wang and J. Hu, "A machine learning based approach for table detection on the web," in *WWW*. ACM, 2002.

[21] B. Gatos, D. Danatsas, I. Pratikakis, and S. J. Perantonis, "Automatic table detection in document images," in *PRIA*, 2005.

[22] K. Zuyev, "Table image segmentation," in *ICDAR*, vol. 2. IEEE, 1997.

[23] J. Hu, R. S. Kashi, D. P. Lopresti, and G. Wilfong, "Medium-independent table detection," in *Document Recognition and Retrieval VII*, vol. 3967. International Society for Optics and Photonics, 1999.

[24] Y. Liu, P. Mitra, and C. L. Giles, "Identifying table boundaries in digital documents via sparse line detection," in *CIKM*. ACM, 2008.

[25] F. Shafait and R. Smith, "Table detection in heterogeneous documents," in *IAPR International Workshop on Document Analysis Systems*, 2010.

[26] H. Dong, S. Liu, S. Han, Z. Fu, and D. Zhang, "Tablesense: Spreadsheet table detection with convolutional neural networks," in *AAAI*, 2019.

[27] J. Fang, P. Mitra, Z. Tang, and C. L. Giles, "Table header detection and classification," in *AAAI*, 2012.

[28] Y. Liu, K. Bai, P. Mitra, and C. L. Giles, "Tableseer: automatic table metadata extraction and searching in digital libraries," in *JCDL*, 2007.

[29] D. N. Tran, T. A. Tran, A. Oh, S. H. Kim, and I. S. Na, "Table detection from document image using vertical arrangement of text blocks," *International Journal of Contents*, 2015.

[30] J. Kim and H. Hwang, "A rule-based method for table detection in website images," *IEEE Access*, 2020.

[31] A. Kalyan, A. Mohta, O. Polozov, D. Batra, P. Jain, and S. Gulwani, "Neural-guided deductive search for real-time program synthesis from examples," in *ICLR*, 2018.

[32] A. Iyer, M. Jonnalagedda, S. Parthasarathy, A. Radhakrishna, and S. K. Rajamani, "Synthesis and machine learning for heterogeneous extraction," in *PLDI*, 2019.