TableParser: Automatic Table Parsing with Weak Supervision from Spreadsheets

Susie Xi Rao[12*], **Johannes Rausch**[1*], **Peter Egger**[2], **Ce Zhang**[1]

[1] Department of Computer Science, ETH Zurich
[2] Department of Management, Technology, and Economics, ETH Zurich
{srao,jrausch,pegger,cezhan}@ethz.ch

Abstract

Tables have been an ever-existing structure to store data. There exist now different approaches to store tabular data physically. PDFs, images, spreadsheets, and CSVs are leading examples. Being able to parse table structures and extract content bounded by these structures is of high importance in many applications. In this paper, we devise TableParser, a system capable of parsing tables in both native PDFs and scanned images with high precision. We have conducted extensive experiments to show the efficacy of domain adaptation in developing such a tool. Moreover, we create TableAnnotator and ExcelAnnotator, which constitute a spreadsheet-based weak supervision mechanism and a pipeline to enable table parsing. We share these resources with the research community to facilitate further research in this interesting direction.

1 Introduction

Automated processing of electronic documents is a common task in industry and research. However, the lack of structures in formats such as native PDF files or scanned documents remains a major obstacle, even for state-of-the-art OCR systems. In practice, extensive engineering and ad-hoc code are required to recover the document structures, e.g., for headings, tables, or nested figures. Sometimes this is required even for text, e.g., in case of PDFs built on the basis of scans, especially, low-quality scans. These structures are hierarchically organized, which many existing systems often fail to recognize.

With the advance of machine learning (ML) and deep learning (DL) techniques, parsing documents can be done more efficiently than ever. As the first end-to-end system for parsing renderings into hierarchical document structures, DocParser (Rausch et al., 2021) was recently introduced. It presents a robust way to parse complete document structures from rendered PDFs. Such learning-based systems require large amounts of labeled training data. This problem is alleviated through a novel weak supervision approach that automatically generates training data from structured LaTeX source files in readily available scientific articles. DocParser

demonstrates a significant reduction of the labeling complexity through this weak supervision in their experiments.

As a special document type, tables are one of the most natural ways to organize structured contents. Tabular data are ubiquitous and come in different formats, e.g., CSV (plain and unformatted) and Microsoft Excel (annotated and formatted), depending on the file creation. Many data processing tasks require tables to be represented in a structured format. However, structured information is not always available in rendered file formats such as PDF. Especially when PDFs are generated from image scans, such information is missing. Luckily, the existing matrix-type organization of spreadsheets can assist to automatically generate document annotations to PDFs. **With spreadsheets as weak supervision, this paper proposes a pipeline to provide an automated process of reading tables from PDFs and utilize them as a weak supervision source for DL systems.**

There are three different types of tasks discussed in the literature about table processing in PDFs, namely, table detection, table structure parsing/recognition (Zhong, ShafieiBavani, and Jimeno Yepes, 2020).[1] Table detection is a popular task with a large body of literature, table structure parsing and table recognition were revisited[2] after the pioneering work of Schreiber et al. (2017) using state-of-the-art deep neural networks. Before DL started to gain success in object detection, table structure parsing was done by bottom-up approaches, using heuristics or ML-based methods like Pinto et al. (2003); Farrukh et al. (2017). See Pivk et al. (2007); Wang, Phillips, and Haralick (2004) for comprehensive reviews on ML methods. The purposes of table structure de-

*These authors contributed equally to this work.

---

[1]Table detection is a task to draw the bounding boxes of tables in documents; table structure recognition/parsing refers to the (additional) identification of the structural (row and column layout) information of tables. We distinguish between bottom-up and top-down approaches in table structure detection. Bottom-up typically refers to structure detection by recognizing formatting cues such as text, lines, and spacing, while top-down entails table cell detection (see Kieninger and Dengel (1998); Pivk et al. (2007); Zhong, ShafieiBavani, and Jimeno Yepes (2020)).

[2]Some recent works on Cascade R-CNN Fernandes et al. (2022); Prasad et al. (2020) manage to push the frontier of table detection. See Rausch et al. (2021) for a general review on table detection and Zhong, ShafieiBavani, and Jimeno Yepes (2020) for a general review on table recognition.

tection are either layout detection (Kieninger and Dengel, 1998) or information retrieval (Pivk et al., 2007) from tabular structures, usually with the former as a preprocessing step for the latter.

The DL-based methods in Schreiber et al. (2017); Qasim, Mahmood, and Shafait (2019) are among the first to apply neural networks designed for object detection to table parsing. Typically, taking pretrained object detection models e.g., Faster RCNN (Long, Shelhamer, and Darrell, 2015; Ren et al., 2015) on benchmarking datasets like ImageNet (Russakovsky et al., 2015), Pascal VOC (Everingham et al., 2010), and Microsoft COCO (Lin et al., 2015), they finetune the pretrained models with in-domain images for table detection and table structure parsing (domain adaption and transfer learning). In some best performing frameworks (Raja, Mondal, and Jawahar, 2020; Zheng et al., 2021; Jiang et al., 2021), they all jointly optimize the structure detection and entity relations in the structure, as in DocParser.

However, a key problem in training DL-based systems is the labeling complexity of generating high-quality indomain annotations. More generally, an essential limiting factor is the lack of large amounts of training data. Efforts have been put into generating datasets to enable tasks with weak supervision. TableBank (Li et al., 2019) is built upon a data set of Word and LaTeX files and extracts annotations directly from the sources. They use 4-gram BLEU score to evaluate the cell content alignments. However, the table layout structure is not of particular focus in TableBank. PubTabNet (Zhong, ShafieiBavani, and Jimeno Yepes, 2020) enables table detection and table cell content detection. arXivdocs-target and arXivdocs-weak by DocParser (Rausch et al., 2021) enables an end-to-end document parsing system of the hierarchical document structure.

In this paper, we devise TableParser with inspiration from DocParser, due to its flexibility in processing both tables and more general documents. We demonstrate that **TableParser** is an effective tool for recognizing table structures and content. The application of TableParser to a new target domain requires newly generated training data. Depending on the target domain, we specify two TableParsers: **ModernTableParser** fine-tuned with native PDFs and **HistoricalTableParser** fine-tuned with scan images. TableParser works in conjunction with **TableAnnotator** (Figure 1) which efficiently assists developers in visualizing the output, as well as help users to generate high-quality human annotations.[3] To generate training instances, we develop **ExcelAnnotator** to interact with spreadsheets and produce annotations for weak supervision.

With **ExcelAnnotator**, we have compiled a spreadsheet dataset ZHYearbooks-Excel, which is processed via a Python library on Excel (PyWin32[4]) to leverage the structured information stored in the spreadsheets. TableParser is

---

[3]For a live demo of table annotations using our annotation tool, refer to the video under `https://github.com/DS3Lab/TableParser/blob/main/demo/2021-06-15%2002-05-58.gif`.

[4]`https://pypi.org/project/pywin32/` (last accessed: Sep. 30, 2021).
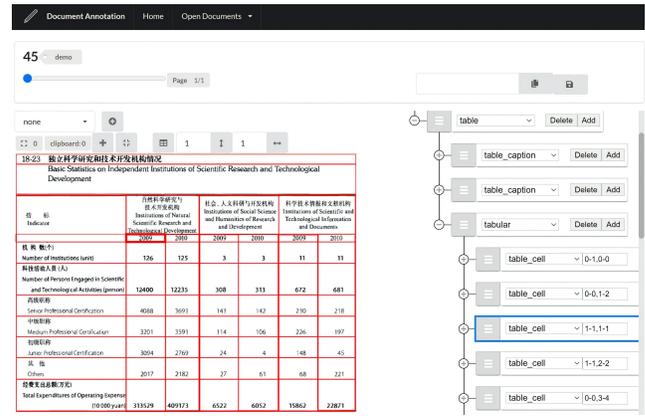

Figure 1: TableAnnotator.

trained with 16'041 Excel-rendered tables using detectron2 (He et al. (2017); Wu et al. (2019)) and fine-tuned with 17 high-quality manual annotations in each domain. We have conducted extensive experiments of domain adaptation. Finally, we evaluate different TableParsers in two domains and make the following observations:

1. In general, domain adaptation works well with fine-tuning the pretrained model (M_{WS} in Figure 2) with high-quality in-domain data.

2. On the test set of 20 tables rendered by Excel, with ModernTableParser we are able to achieve an average precision score (IoU ≥ 0.5) of 83.53% and 73.28% on table rows and columns, respectively.

3. We have tested our HistoricalTableParser on scanned tables in both historical (medium-quality, scan-based) and modern tables. Overall, HistoricalTableParser works better than ModernTableParser on tables stored in image scans.

4. Interestingly, we find that ModernTableParser built on top of DocParser (Rausch et al., 2021) is very robust in adapting to new domains, such as scanned historical tables.

We are willing to open source the ZHYearbook-Excel dataset, TableAnnotator, TableParser system, and its pipeline to the research communities.[5] Moreover, we welcome future contributions to the project to further increase the usability of TableParser in various domains.

To summarize, our key contributions in this paper are:

1. We present **TableParser** which is a robust tool for parsing modern and historical tables stored in native PDFs or image scans.

2. We conduct experiments to show the efficacy of domain adaptation in TableParser.

3. We contribute a new pipeline (using **ExcelAnnotator** as the main component) to automatically generate weakly labeled data for DL-based table parsing.

---

[5]The source code, data, and/or other artifacts for the complete TableParser pipeline have been made available at `https://github.com/DS3Lab/TableParser`.
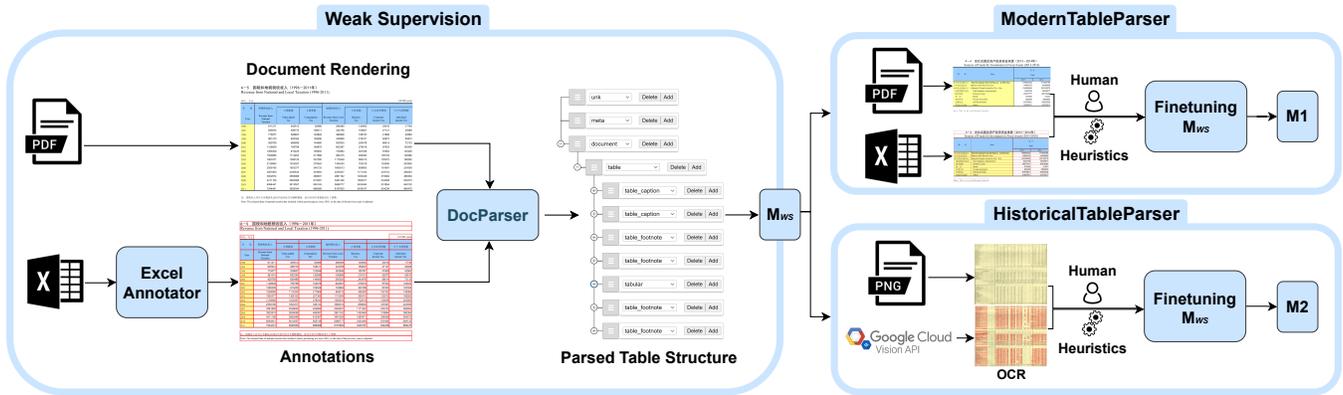
Figure 2: System design of TableParser: Weak supervision, ExcelAnnotator, ModernTableParser and HistoricalTableParser. M_{WS}: the pretrained model with the dataset ZHYearbooks-Excel-WS. M1: for ModernTableParser, fine-tuned on Excel-rendered images; M2: for HistoricalTableParser, fine-tuned on scan images.

4. We contribute **TableAnnotator** as a graphical interface to assist table structure understanding and manual labeling.

5. We open-source the spreadsheet weak supervision dataset and the pipeline of TableParser to encourage further research in this direction.

2 TableParser System

2.1 Problem Description

Following the hierarchical document parsing in DocParser, our objective is to generate a hierarchical structure for a table which consists of the entities (*table, tabular, table_caption, table_row, table_column, table_footnote*) and their relations in the document tree.

Our ultimate goal of table structure parsing is (1) to establish row/column relationships between the table cells, and (2) post-process the established structure and cell content (e.g., with PDFMiner[6] or OCR engines) to enable a CSV export function. In this paper, we emphasize (1) and are still in development to enable (2). Our work will enable a user to parse a table stored in a PDF format and obtain (i) the location of a certain cell (specified by its row range and column range) and (ii) the cell content mapped to the cell location.

2.2 System Components

We introduce the main system components in TableParser, incl. TableAnnotator, ExcelAnnotator, ModernTableParser, and HistoricalTableParser.

TableAnnotator. In Figure 1 we show TableAnnotator, which is mainly composed of two parts: image panel (left) and document tree (right). In the code repository[7], there is a manual describing its functionalities in details. In a nutshell, annotators can draw bounding boxes on the left panel and create their entities and relationships on the right. In Figure

---

1, the highlighted bounding box (the red thick contour on the left) corresponds to the *table_cell* on the second row and second column, indexed by 1-1, 1-1 (the blue highlight on the right). Note that TableAnnotator is versatile and can be used to annotate not only tables, but also generic documents. The output of the tree is in JSON format, like the code snippet shown in Listing 1.

Listing 1: TableParser JSON format.

```
1  [{"id": 28,              7    {"id": 29,
2    "category": "table_cell",  8     "category": "box",
3    "properties": "1-1,1-1",   9     "page": 0,
4    "row_range": [1,1],    10    "bbox": [365,332,299,27],
5    "col_range": [1,1],    11    "parent": 28}]
6    "parent": 9},
```

ModernTableParser. We train ModernTableParser using the data generated by weak supervision signals from Excel sheets and fine-tuned by high-quality manual annotations in this domain. In Figure 2, we show the system design following the underlying components of DocParser.[8] We denote the model that produces ModernTableParser as **M1**.

Weak Supervision with ExcelAnnotator. Now we present the crucial steps in generating weak supervision (the model M_{WS} in Figure 2) for TableParser. These steps are mainly conducted by ExcelAnnotator in Figure 2 (left). Take a worksheet-like Figure 3 (a) from our ZHYearbook-Excel-WS dataset (cf. Section 3), where we see caption, tabular, and footnote areas. We subsequently use DeExcelerator (Eberius et al., 2013) to extract relations from the spreadsheets. We utilize DeExcelerator to categorize the content, such that we can differentiate among table captions, table footnotes and tabular data and create a correct auxiliary file

---

(a) Example worksheet from ZHYearbook-Excel-WS.

(b) Annotations with DeExcelerator.

(c) Representing bounding boxes in Excel.

(d) Visualization of bounding boxes with TableAnnotator.

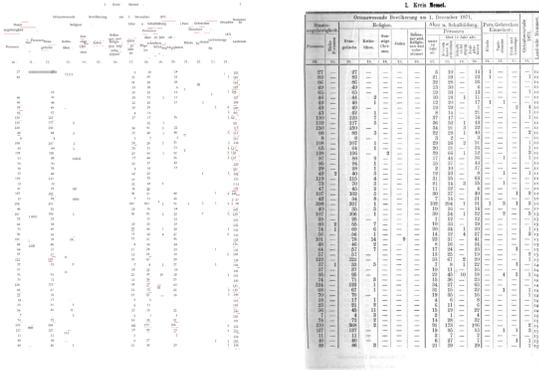Figure 3: A working example in ExcelAnnotator.

Figure 4: Google Vision OCR API output (left) of image (right): bad quality of OCR.
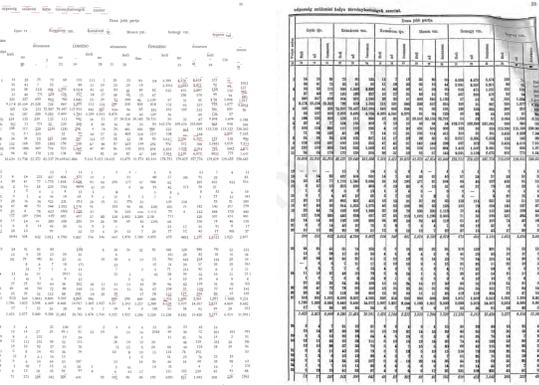

Figure 5: Google Vision OCR API output (left) of image (right): good quality of OCR.

to each PDF containing the structural information of the represented table(s). Illustrated in Figure 3 (b), in this case we annotate the table caption and footnote as 'meta', and mark the range of content with 'content' and 'empty'. We use Py-Win32 in Python to interact with Excel, so that intermediate representations like Figure 3 (c) can be created to retrieve entity locations in the PDF rendering. Concretely, we mark neighboring cells with distinct colors, remove all borders, and set the font color to white. To summarize, ExcelAnnotator detects spreadsheet metadata and cell types, as well as retrieves entity locations via intermediate representations.

Finally, we are able to load the annotations into TableAnnotator to inspect the quality of weak supervision (Figure 3 (d)).

HistoricalTableParser. We use the OCR engine from Google Vision API to recognize the text bounding boxes. Then we convert bounding boxes into the input format TableParser requires. Now we are able to manually adjust the bounding boxes in TableAnnotator to produce high-quality annotations. Note that the quality of OCR highly depends on the table layout (Figures 4 vs. 5), we often need to adjust the locations of bounding boxes and redraw the bounding boxes of individual cells.

In Figure 2 (lower right), we show the system design by adding an OCR component and a fine-tuning component for domain adaptation. We denote the model that produces HistoricalTableParser as **M2**. Take Figure 6 (a) as input, TableParser can produce a parsed layout-like Figure 6 (b) which can be combined with the OCR bounding boxes in the subsequent steps and export as a CSV file (Figure 6 (c)).[9]

For domain adaptation, we assume that an out-of-domain model performs worse than an in-domain model in one domain. Namely, we would expect ModernTableParser to work better on Excel-rendered PDFs or tables created similarly; on the contrary, we would expect HistoricalTableParser to perform better on older table scans.

3 Datasets

We have compiled various datasets to train, fine-tune, test, and evaluate TableParser.

ZHYearbooks-Excel. We create three datasets from this source: ZHYearbooks-Excel-WS, ZHYearbooks-Excel-FT, and ZHYearbooks-Excel-Test, with 16'041, 17, and 20 tables in each set. On average, it takes 3 minutes 30 seconds for an annotator to produce high-quality annotations of a table. The manual annotations are done with automatically generated bounding boxes and document tree as aid.

ZHYearbooks-OCR. We create the dataset ZHYearbook-OCR-Test, with 20 tables. On average, it takes 2 minutes and

---

[9]c.f. The performance of LayoutParser is quite poor on the tabular data in Figure 6 (d) using the best model from its model zoo (PubLayNet/faster_rcnn_R_50_FPN_3x). Input and annotated figures of original size can be found under https://github.com/DS3Lab/TableParser/tree/main/figures.
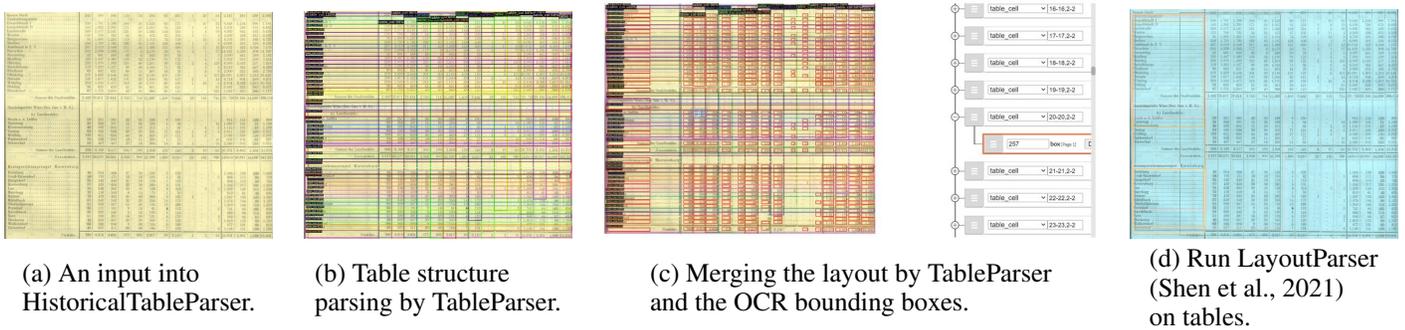
(a) An input into HistoricalTableParser.

(b) Table structure parsing by TableParser.

(c) Merging the layout by TableParser and the OCR bounding boxes.

(d) Run LayoutParser (Shen et al., 2021) on tables.

Figure 6: A working example in HistoricalTableParser.

45 seconds to annotate a table with the similar annotation aids mentioned above.

EUYearbooks-OCR. We create two datasets from this source: EUYearbook-OCR-FT and EUYearbook-OCR-Test, with 17 and 10 tables, respectively. Note that these datasets contain various languages like Hungarian and German, with various formats depending on the language. On average, it takes 8 minutes and 15 seconds to annotate a table with the similar annotation aids mentioned above.

Miscellaneous historical yearbooks. We ran ModernTableParser and HistoricalTableParser on Chinese and Korean historical yearbooks and inspect their outputs qualitatively (see Section 5.2).

Human labeling efforts. We observe a large variance in labeling intensity across the datasets. The EUYearbooks-OCR datasets require more corrections per table compared to the datasets of modern tables. Moreover, they also require more iterations of human annotations with heuristics as aid.

4 Computational Setup

4.1 Mask R-CNN

In line with DocParser, we use the same model but with an updated backend implementation. Namely, we utilize Detectron2 to apply an updated version of Mask R-CNN (He et al., 2016). For technical details of Mask R-CNN, we refer to DocParser (Rausch et al., 2021).

Training Procedure: Weak Supervision + Fine-Tuning. All neural models are initialized with weights trained on the MS COCO dataset. We first pretrain on the weak supervision data ZHYearbook-Excel-WS for 540k iterations, then fine-tune on our target datasets ZHYearbook-Excel-FT and EUYearbook-OCR-FT for M1 and M2, respectively. We then fine-tune each model across three phrases for a total of 30k iterations. This is split into 22k, 4k, 4k iterations, respectively. The performance is measured every 500 iterations via the IoU with a threshold of 0.5. We train all models in a multi-GPU setting, using 8 GPUs with a vRAM of 12 GB. Each GPU was fed with one image per training iteration. Accordingly, the batch size per training iteration is set to 8. Furthermore, we use stochastic gradient descent with a learning rate of 0.005 and learning momentum of 0.9.

Parameter Settings. During training, we sampled randomly 100 entities from the ground truth per document image (i.e., up to 100 entities, as some document images might have less). In Mask R-CNN, the maximum number of entity predictions per image is set to 100. During prediction, we only keep entities with a confidence score of 0.5 or higher.

5 Results and Discussion

Here, we evaluate the performance of TableParser in two domains quantitatively and qualitatively.

5.1 Quantitative assessment

Metric. We first introduce the evaluation metric for the object detection/classification tasks. The metric we report is Average Precision (AP), which corresponds to an Intersection over Union rate of IoU=.50:.05:.95.[10] IoU ranges from 0 to 1 and specifies the amount of overlap between the predicted and ground truth bounding box. It is a common metric used when calculating AP.

Performances in various domains. As we discussed in Section 2, we have developed ModernTableParser to parse tables with input images rendered by Excel (M1). Then, to work with historical tables in scans, we adapt the pretrained TableParser by fine-tuning it on scanned documents (M2). Now, we present the performances of M1 and M2 in two different domains in the following aspects:

1. **(P1)** the performances on fine-tuning sets on M1 and M2 in Table 1;

2. **(P2)** the performances on fine-tuning sets as test sets on M1 and M2 in Table 1;[11]

3. **(P3)** the performances on three test sets from two domains on M1 and M2 in Table 2.

(P1) & (P2). We want to study the impact of fine-tuning of a pretrained model (using a large body of tables generated by weak supervision signals). The instances used to fine-tune must be high-quality in-domain data. Concretely, we create in-domain annotations for modern tables (rendered by Excel) and historical tables (from scans) with high human

---

[10]We refer readers to https://cocodataset.org/#detection-eval for more details on the evaluation metrics (last accessed: Nov. 1, 2021).

[11]This means we evaluate the performance of M1 on the fine-tuned set for M2 (as a test set for M1) and vice versa.

Table 1: Fine-tuning results of M1 and M2. M1: for ModernTableParser, fine-tuned on Excel-rendered images; M2: for HistoricalTableParser, fine-tuned on scan images; FT: fine-tune.

ZHYearbook-Excel-FT				EUYearbook-OCR-FT			
Category	**# instances**	**Average Precision**		**Category**	**# instances**	**Average Precision**	
		M1 (FT)	**M2** (Test)			**M1** (Test)	**M2** (FT)
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)
table	17	90.973	38.034	*table*	17	67.467	93.011
tabular	17	100.000	57.897	*tabular*	17	76.423	100.000
table_column	134	96.730	15.253	*table_column*	260	24.930	81.376
table_row	548	79.228	39.485	*table_row*	1180	19.256	60.899

Table 2: Test results of M1 and M2 on various data sets.

ZHYearbook-Excel-Test				ZHYearbook-OCR-Test				EUYearbook-OCR-Test			
Category	**# instances**	**Average Precision**		**Category**	**# instances**	**Average Precision**		**Category**	**# instances**	**Average Precision**	
		M1	**M2**			**M1**	**M2**			**M1**	**M2**
(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)	(10)	(11)	(12)
table	20	85.407	32.821	*table*	10	56.942	53.356	table	10	57.151	81.907
tabular	21	80.193	43.801	*tabular*	10	64.175	52.563	*tabular*	10	85.956	91.429
table_column	176	73.277	14.927	*table_column*	74	43.094	21.997	*table_column*	136	36.616	40.509
table_row	513	83.528	48.912	*table_row*	226	50.055	36.619	*table_row*	665	25.645	40.229

efforts assisted by automatic preprocessing: ZHYearbook-Excel-FT and EUYearbook-OCR-FT, each with 17 tables. Note that the latter has much denser rows and columns than the former (see the tables in Figures 3 (a) vs. 6 (a) for an illustration). It is apparent from Table 1 that the AP performance of models on the fine-tuning sets is highly optimized (columns (3) and (8) in Table 1), and it should be better than using those datasets as test sets. This means, if we run M1 (fine-tuned by modern tables) on EUYearbook-OCR-FT (column 7), its performance is worse than fine-tuning; and if we run M2 (fine-tuned by historical tables) on ZHYearbook-Excel-FT (column 4), it performs worse than fine-tuning. Interestingly, if we compare the performance of M2 on modern tables (column (4)) with the performance of M1 on historical tables (column (7)), we clearly see that the latter has a better performance in all other categories than the class of *table_row*. This can be explained by the fact that the model trained on modern tables is robust in annotating historical tables, at least on the column level. We see this in Figures 9 and 10, where ModernTableParser clearly performs better. **However, the algorithm has problems in delineating narrow and less clearly separated rows.** This could be due to the setting of the maximum number of entities being 100 when predicting per table (Section 4.1).

(P3). In Table 2, we show the performances of three test sets from two domains (Excel-rendered PDFs and historical scans), namely, ZHYearbook-Excel-Test, ZHYearbook-OCR-Test, and EUYearbook-OCR-Test. We see that M2 which is fine-tuned by historical scans performs worse than M1 on both ZHYearbook-Excel-Test and ZHYearbook-OCR-Test. Vice versa, M1 that is fine-tuned by Excel-rendered PDFs performs worse than M2 on EUYearbook-OCR-Test. This suggests that domain adaptation by fine-tuning the pretrained TableParser with in-domain high-quality data works well.

Additionally, if we compare the $\Delta AP|(M1 - M2)|$ under each test set (e.g., the differences of columns (3) and (4), of (7) and (8), of (11) and (12)), the ΔAP on *-OCR-

Test in all categories is smaller than ZHYearbook-Excel-Test, with M1 already achieving medium-high performance on the test set. Although M1 is not fine-tuned by in-domain historical images, ModernTableParser is still able to parse historical scans with moderate performance. This suggests that TableParser trained on modern table structures can be used to parse the layout of tabular historical scans. Because the cost is often too high in generating a large amount of training data of historical scans (see Section 3 for the discussion of labeling efforts), our approach shows a promising direction in first developing TableParser that works well for modern tables, and then adapting TableParser to the historical domain by fine-tuning on only a few manually annotated historical scans of good quality.

5.2 Qualitative Assessment

In Figures 7, 8, 9, and 10, we show the qualitative outputs of ModernTableParser and HistoricalTableParser on various types of inputs.[12] The quality of structure parsing varies across inputs, but overall, the quality is high. Even if we simply use ModernTableParser to parse old scans, it achieves a moderate performance, sometimes better than HistoricalTableParser (see Figures 9 and 10). This substantiates our claim that knowing the table structure (caption, tabular, row, column, multi-cell, etc.) is of foremost importance for parsing tables. We see that the performance of LayoutParser is quite poor on the tabular data in Figure 6 (d) using the best model from its model zoo (PubLayNet/faster_rcnn_R_50_FPN_3x).

6 Related Work

Table Annotation. TableLab (Wang, Burdick, and Li, 2021) provides an active learning based annotation GUI for users to jointly optimize the model performance under the hood. LayoutParser (Shen et al., 2021) has also promoted

---

[12]Input and annotated figures of original size can be found under `https://github.com/DS3Lab/TableParser/tree/main/figures`.

Figure 7: A Hungarian table parsed by ModernTableParser (left) and HistoricalTableParser (right).

Figure 8: A German table parsed by ModernTableParser (left) and HistoricalTableParser (right).

Figure 9: A Korean table parsed by ModernTableParser (left) and HistoricalTableParser (right).

Figure 10: A Chinese table parsed by ModernTableParser (left) and HistoricalTableParser (right).

an interactive document annotation tool[13], but the tool is not optimized for table annotations.

Table Structure Parsing. As pioneering works in table structure parsing, Kieninger and Dengel (1998) and Schreiber et al. (2017) have both included a review of works in table structure recognition prior to DL. Prior methods typically required high human efforts in creating the feature extraction. After Schreiber et al. (2017), researchers have started to revisit table structure parsing with DL methods, which turned out highly promising compared to the rule-based (e.g., Kieninger and Dengel (1998); Pivk et al. (2007)) and ML-based methods (e.g., Pinto et al. (2003); Wang, Phillips, and Haralick (2004); Farrukh et al. (2017)).

The success of DL has marked the revisiting of table structure parsing by Schreiber et al. (2017), which inspired follow-up research (Chi et al., 2019; Rausch et al., 2021; Prasad et al., 2020; Zhong, ShafieiBavani, and Ji-

---

[13]See `https://github.com/Layout-Parser/annotation-service` (last accessed: Nov. 1, 2021).

meno Yepes, 2020; Li et al., 2021; Xue et al., 2021; Long et al., 2021; Jiang et al., 2021; Nazir et al., 2021; Zheng et al., 2021; Luo et al., 2021; Raja, Mondal, and Jawahar, 2020). To highlight a few, Zhong, ShafieiBavani, and Ji-meno Yepes (2020) proposed EDD (encoder-dual-decoder) to covert table images into HTML code, and they evaluate table recognition (parsing both table structures and cell contents) using a newly devised metric, TEDS (Tree-Edit-Distance-based Similarity). Xue et al. (2021) proposed TGRNet as an effective end-to-end trainable table graph construction network, which encodes a table by combining the cell location detection and cell relation prediction. Li et al. (2021) used bi-LSTM on table cell detection by encoding rows/columns in neural networks before the softmax layer. Researchers also started discussing effectively parsing tables in the wild (Long et al., 2021), which is relevant to the perturbation tests we want to conduct for historical tables. Tab-CellNet by Jiang et al. (2021) adopts a Hybrid Task Cascade network, interweaving object detection and instance segmentation tasks to progressively improve model performance. We see from the previous works, the most effective methods (Raja, Mondal, and Jawahar, 2020; Zheng et al., 2021; Jiang et al., 2021) always jointly optimize the cell lo-

cations and cell relationships. In our work, we consider these two aspects by learning the row and column alignments in a hierarchical structure, where we know the relationship of entities in the table (row, column, cell, caption, footnote).

7 Discussion and Conclusion

7.1 Efficiency

PyWin32 uses the component object model (COM), which only supports single-thread processing and only runs under Windows. But with 20 VMs, we managed to process a large amount of files. This is a one-time development cost. On average – on the fastest machine used (with 16 GB memory, 6 cores, each of 4.8GHz max (2.9 base)) – it took 15.25 seconds to process one document (a worksheet in this case). To fine-tune a pretrained TableParser with 17 images, it takes 3-4 hours to fine-tune the model with 30k iterations.

7.2 Future Work

Based on our findings, we will further improve the parsing performance on table row/column/cell. Besides, we plan to enable a CSV-export functionality in TableParser, which allows users to export a CSV file that attends to both bounding boxes generated by the OCR'ed and the hierarchical table structure. We will also benchmark this functionality against human efforts. Another practical functionality we add to facilitate users' assessment of table parsing quality, is that we enable TableParser to compute row and column sums when exporting to the CSV format. Because tables sometimes come with row/column sums in the rendered format, this functionality can help users to assess their manual efforts in post-editing the CSV output. We also plan to conduct perturbation tests of table structures and quantify the robustness of our models in those scenarios. These exercises will be highly valuable because, as we see in Figure 7, we often encounter scan images of tables where the rectangle structures cannot be maintained (the upper right corner). This brings us to another interesting research direction: how to efficiently annotate the non-rectangle elements in a table, e.g., Long et al. (2021) have provided the benchmarking dataset and method for parsing tables in the wild. Finally, we would like to benchmark TableParser using the popular benchmarking datasets such as ICDAR-2013, ICDAR-2019, TableBank, and PubTabNet. Note that since we develop TableParser on top of the DocParser (Rausch et al., 2021), where the reported F1 score has shown superior performance of our method on ICDAR-2013.

7.3 Conclusion

We present in this work our DL-based pipeline to parse table structures and its components: TableAnnotator, TableParser (Modern and Historical), and ExcelAnnotator. We also demonstrate that pre-training TableParser on weakly annotated data allows highly accurate parsing of structured data in real-world table-form data documents. Fine-tuning the pretrained TableParser in various domains has shown large improvements in detection accuracy. We have observed that the state-of-the-art for table extraction is shifting towards DL-based approaches. However, devising suitable tools to facilitate training of such DL approaches for the research community is still lacking. Hence, we provide a pipeline and open-source code and data to invite the active contribution of the community.

References

Chi, Z.; Huang, H.; Xu, H.-D.; Yu, H.; Yin, W.; and Mao, X.-L. 2019. Complicated table structure recognition. *arXiv preprint arXiv:1908.04729*.

Eberius, J.; Werner, C.; Thiele, M.; Braunschweig, K.; Dannecker, L.; and Lehner, W. 2013. Deexcelerator: a framework for extracting relational data from partially structured documents. 2477–2480.

Everingham, M.; Van Gool, L.; Williams, C. K.; Winn, J.; and Zisserman, A. 2010. The pascal visual object classes (voc) challenge. *International journal of computer vision* 88(2):303–338.

Farrukh, W.; Foncubierta-Rodriguez, A.; Ciubotaru, A.-N.; Jaume, G.; Bejas, C.; Goksel, O.; and Gabrani, M. 2017. Interpreting data from scanned tables. In *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*, volume 2, 5–6. IEEE.

Fernandes, J.; Simsek, M.; Kantarci, B.; and Khan, S. 2022. Tabledet: An end-to-end deep learning approach for table detection and table image classification in data sheet images. *Neurocomputing* 468:317–334.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

He, K.; Gkioxari, G.; Dollár, P.; and Girshick, R. 2017. Mask r-cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, 2980–2988.

Jiang, J.; Simsek, M.; Kantarci, B.; and Khan, S. 2021. Tab-cellnet: Deep learning-based tabular cell structure detection. *Neurocomputing* 440:12–23.

Kieninger, T., and Dengel, A. 1998. The t-recs table recognition and analysis system. In *International Workshop on Document Analysis Systems*, 255–270. Springer.

Li, M.; Cui, L.; Huang, S.; Wei, F.; Zhou, M.; and Li, Z. 2019. Tablebank: A benchmark dataset for table detection and recognition.

Li, Y.; Huang, Y.; Zhu, Z.; Pan, L.; Huang, Y.; Du, L.; Tang, Z.; and Gao, L. 2021. Rethinking table structure recognition using sequence labeling methods. In *International Conference on Document Analysis and Recognition*, 541–553. Springer.

Lin, T.-Y.; Maire, M.; Belongie, S.; Bourdev, L.; Girshick, R.; Hays, J.; Perona, P.; Ramanan, D.; Zitnick, C. L.; and Dollár, P. 2015. Microsoft coco: Common objects in context.

Long, R.; Wang, W.; Xue, N.; Gao, F.; Yang, Z.; Wang, Y.; and Xia, G.-S. 2021. Parsing table structures in the wild. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 944–952.

Long, J.; Shelhamer, E.; and Darrell, T. 2015. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 3431–3440.

Luo, S.; Wu, M.; Gong, Y.; Zhou, W.; and Poon, J. 2021. Deep structured feature networks for table detection and tabular data extraction from scanned financial document images. *arXiv preprint arXiv:2102.10287*.

Nazir, D.; Hashmi, K. A.; Pagani, A.; Liwicki, M.; Stricker, D.; and Afzal, M. Z. 2021. Hybridtabnet: Towards better table detection in scanned document images. *Applied Sciences* 11(18):8396.

Pinto, D.; McCallum, A.; Wei, X.; and Croft, W. B. 2003. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, 235–242.

Pivk, A.; Cimiano, P.; Sure, Y.; Gams, M.; Rajkovič, V.; and Studer, R. 2007. Transforming arbitrary tables into logical form with tartar. *Data & Knowledge Engineering* 60(3):567–595.

Prasad, D.; Gadpal, A.; Kapadni, K.; Visave, M.; and Sultanpure, K. 2020. Cascadetabnet: An approach for end to end table detection and structure recognition from image-based documents. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 572–573.

Qasim, S. R.; Mahmood, H.; and Shafait, F. 2019. Rethinking table recognition using graph neural networks.

In *2019 International Conference on Document Analysis and Recognition (ICDAR)*, 142–147. IEEE.

Raja, S.; Mondal, A.; and Jawahar, C. 2020. Table structure recognition using top-down and bottom-up cues. In *European Conference on Computer Vision*, 70–86. Springer.

Rausch, J.; Martinez Bermudez, J. O.; Bissig, F.; Zhang, C.; and Feuerriegel, S. 2021. Docparser: Hierarchical document structure parsing from renderings. In *35th AAAI Conference on Artificial Intelligence (AAAI-21)(virtual)*.

Ren, S.; He, K.; Girshick, R.; and Sun, J. 2015. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems* 28:91–99.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision* 115(3):211–252.

Schreiber, S.; Agne, S.; Wolf, I.; Dengel, A.; and Ahmed, S. 2017. Deepdesrt: Deep learning for detection and structure recognition of tables in document images. In *2017 14th IAPR international conference on document analysis and recognition (ICDAR)*, volume 1, 1162–1167. IEEE.

Shen, Z.; Zhang, R.; Dell, M.; Lee, B. C. G.; Carlson, J.; and Li, W. 2021. Layoutparser: A unified toolkit for deep learning based document image analysis. *arXiv preprint arXiv:2103.15348*.

Wang, N. X. R.; Burdick, D.; and Li, Y. 2021. Tablelab: An interactive table extraction system with adaptive deep learning. In *26th International Conference on Intelligent User Interfaces*, 87–89.

Wang, Y.; Phillips, I. T.; and Haralick, R. M. 2004. Table structure understanding and its performance evaluation. *Pattern recognition* 37(7):1479–1497.

Wu, Y.; Kirillov, A.; Massa, F.; Lo, W.-Y.; and Girshick, R. 2019. Detectron2. https://github.com/facebookresearch/detectron2.

Xue, W.; Yu, B.; Wang, W.; Tao, D.; and Li, Q. 2021. Tgrnet: A table graph reconstruction network for table structure recognition. *arXiv preprint arXiv:2106.10598*.

Zheng, X.; Burdick, D.; Popa, L.; Zhong, X.; and Wang, N. X. R. 2021. Global table extractor (gte): A framework for joint table identification and cell structure recognition using visual context. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 697–706.

Zhong, X.; ShafieiBavani, E.; and Jimeno Yepes, A. 2020. Image-based table recognition: data, model, and evaluation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XXI 16*, 564–580. Springer.