LayoutParser: A Unified Toolkit for Deep Learning Based Document Image Analysis

Zejiang Shen[1], Ruochen Zhang[2], Melissa Dell[3], Benjamin Charles Germain Lee[4], Jacob Carlson[3], and Weining Li[5]

[1] Allen Institute for AI
shannons@allenai.org
[2] Brown University
ruochen_zhang@brown.edu
[3] Harvard University
{melissadell,jacob_carlson}@fas.harvard.edu
[4] University of Washington
bcgl@cs.washington.edu
[5] University of Waterloo
w422li@uwaterloo.ca

Abstract. Recent advances in document image analysis (DIA) have been primarily driven by the application of neural networks. Ideally, research outcomes could be easily deployed in production and extended for further investigation. However, various factors like loosely organized codebases and sophisticated model configurations complicate the easy reuse of important innovations by a wide audience. Though there have been on-going efforts to improve reusability and simplify deep learning (DL) model development in disciplines like natural language processing and computer vision, none of them are optimized for challenges in the domain of DIA. This represents a major gap in the existing toolkit, as DIA is central to academic research across a wide range of disciplines in the social sciences and humanities. This paper introduces LayoutParser, an open-source library for streamlining the usage of DL in DIA research and applications. The core LayoutParser library comes with a set of simple and intuitive interfaces for applying and customizing DL models for layout detection, character recognition, and many other document processing tasks. To promote extensibility, LayoutParser also incorporates a community platform for sharing both pre-trained models and full document digitization pipelines. We demonstrate that LayoutParser is helpful for both lightweight and large-scale digitization pipelines in real-word use cases. The library is publicly available at https://layout-parser.github.io.

Keywords: Document Image Analysis · Deep Learning · Layout Analysis · Character Recognition · Open Source library · Toolkit.

1 Introduction

Deep Learning(DL)-based approaches are the state-of-the-art method for a wide range of document image analysis (DIA) tasks including document image

classification [10, 33], layout detection [34, 19], table detection [22], and scene text detection [3]. A generalized learning-based framework dramatically reduces the need for the manual specification of complicated rules, which is the status quo with traditional methods. It has the potential to transform the DIA pipeline and benefit a broad spectrum of large-scale document digitization projects.

However, there are several practical difficulties for taking advantages of recent advances in DL-based methods: 1) DL models are notoriously convoluted for reuse and extension. Existing models are developed using distinct frameworks like TensorFlow [1] or PyTorch [21], and the high-level parameters can be obfuscated by implementation details [7]. It can be a time-consuming and frustrating experience to debug, reproduce, and adapt existing models for DIA, and *many researchers who would benefit the most from using these methods lack the technical background to implement them from scratch.* 2) Document images contain diverse and disparate patterns across domains, and customized training for these models is required to achieve a desirable detection accuracy. Currently *there is no full-fledged infrastructure for easily curating the target document image datasets and fine-tuning or re-training the models.* 3) DIA usually requires a sequence of models and other processing to obtain the final outputs. Without a framework bridging DL and DIA, the current practice is to use DL models and perform document analysis in separate processes, and these pipelines are not documented in any central location (and often not documented at all). This makes it *difficult for research teams to learn about how full pipelines are implemented* and *leads them to invest significant resources in reinventing the DIA wheel.*

`LayoutParser` provides a unified toolkit to support DL-based document image analysis and processing. To address the aforementioned challenges, `LayoutParser` is built with the following components:

1. An off-the-shelf toolkit for applying DL models for layout detection, character recognition, and many other DIA tasks (Section 3)
2. A rich repository of pre-trained neural network models (Model Zoo) that underlies the off-the-shelf usage
3. Comprehensive tools for efficient document image data annotation and model tuning to support different levels of customization
4. A DL model hub and community platform for the easy sharing, distribution, and discussion of DIA models and pipelines for reusability, reproducibility, and extensibility (Section 4)

The library implements simple and intuitive Python APIs without sacrificing generalizability and versatility, and can be easily installed via pip. Its convenient functions for handling document image data could be seamlessly integrated with existing DIA pipelines. With detailed documentations and carefully curated tutorials, we hope this tool can benefit a variety of end-users, and can lead to advances in applications in both industry and academic research.

`LayoutParser` is well aligned with recent efforts for improving DL model reusability in other disciplines like natural language processing [7, 30] and computer vision [31], but with a focus on unique challenges in DIA. We show `LayoutParser` can be applied for sophisticated and large-scale digitization

projects that require precision, efficiency, and robustness, as well as simple and light-weight document processing tasks focusing on efficacy and flexibility (Section 5). LayoutParser is being actively maintained, and the supports for more deep learning models and novel methods in text-based layout analysis methods [33, 30] are planned.

The rest of the paper is organized as follows. Section 2 provides an overview of related work. The core LayoutParser library, DL Model Zoo, and customized model training are described in Section 3, and the DL model hub and community platform are detailed in Section 4. Section 5 shows two examples of how LayoutParser can be used in practical DIA projects, and Section 6 concludes.

2 Related Work

Recently, various DL models and datasets has been developed for layout analysis tasks. The dhSegment [19] utilizes fully convolutional networks [18] for segmentation tasks on historical documents. Object detection-based methods like Faster R-CNN [24] and Mask R-CNN [11] are used for identifying document elements [34] and detecting tables [26, 22]. Most recently, Graph Neural Networks [25] have also been used in table detection [23]. However, these models are usually implemented individually and there is no unified framework to load and use such models.

There has been a surge of interest in creating open-source tools for document image processing: a search of `document image analysis` in Github leads to 5M relevant code pieces; yet most of them rely on traditional rule-based methods or provide limited functionalities. The closest prior research to our work is the OCR-D project[6], which also tries to build a complete toolkit for DIA. However, it is designed for analyzing historical documents, and provides no supports for recent DL models. The `DocumentLayoutAnalysis` project[7] focuses on processing born-digital PDF documents via analyzing the stored PDF data. Repositories like `DeepLayout`[8] and `Detectron2-PubLayNet`[9] are individual deep learning models trained on layout analysis datasets without the support for the full DIA pipeline. OCR engines like `Tesseract` [13], `easyOCR`[10] and `paddleOCR`[11] usually do not come with comprehensive functionalities for other DIA tasks like layout analysis.

Recent years have also seen numerous efforts to create libraries for promoting reproducibility and reusability in the field of DL. Libraries like Dectectron2 [31], AllenNLP [7] and transformers [30] have provided the community with complete DL-based support for developing and deploying models for general computer vision and natural language processing problems. LayoutParser, on the other hand, specializes specifically in DIA tasks. LayoutParser is also equipped with a community platform inspired by established model hubs such as `Torch Hub` [20]

[6] https://ocr-d.de/en/about
[7] https://github.com/BobLd/DocumentLayoutAnalysis
[8] https://github.com/leonlulu/DeepLayout
[9] https://github.com/hpanwar08/detectron2
[10] https://github.com/JaidedAI/EasyOCR
[11] https://github.com/PaddlePaddle/PaddleOCR

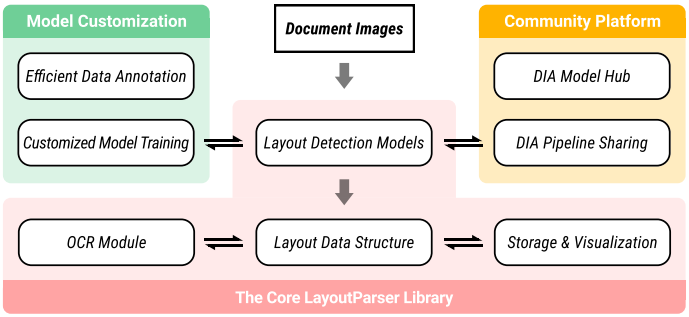


Fig. 1: The overall architecture of `LayoutParser`. For an input document image, the core `LayoutParser` library provides a set of off-the-shelf tools for layout detection, OCR, visualization, and storage, backed by a carefully designed layout data structure. `LayoutParser` also supports high level customization via efficient layout annotation and model training functions. These improve model accuracy on the target samples. The community platform enables the easy sharing of DIA models and whole digitization pipelines to promote reusability and reproducibility. A collection of detailed documentation, tutorials and exemplar projects make `LayoutParser` easy to learn and use.

and `TensorFlow Hub` [1]. It enables the sharing of pretrained models as well as full document processing pipelines that are unique to DIA tasks.

There have been a variety of document data collections to facilite the development of DL models. Some examples include PRImA [2](magazine layouts), PubLayNet [34](academic paper layouts), Table Bank [16](tables in academic papers), Newspaper Navigator Dataset [14, 15](newspaper figure layouts) and `HJDataset` [27](historical Japanese document layouts). A spectrum of models trained on these datasets are currently available in the `LayoutParser` model zoo to support different use cases.

3 The Core `LayoutParser` Library

At the core of `LayoutParser` is an off-the-shelf toolkit that streamlines DL-based document image analysis. Five components support a simple interface with comprehensive functionalities: 1) The *layout detection models* enable using pre-trained or self-trained DL models for layout detection with just four lines of code. 2) The detected layout information is stored in the carefully engineered *layout data structures*, which are optimized for efficiency and versatility. 3) When necessary, users can employ existing or customized OCR models via the unified API provided in the *OCR module*. 4) `LayoutParser` comes with a set of utility functions for *visualization and storage* of the layout data. 5) `LayoutParser` is also

Dataset	Base Model	Large Model	Notes
PubLayNet [34]	F / M	M	Layouts of modern scientific documents
PRImA [2]	M	-	Layouts of scanned modern magazines and scientific reports
Newspaper [15]	F	-	Layouts of scanned US newspapers from the 20th century
TableBank [16]	F	F	Table region on modern scientific and business document
HJDataset [27]	F / M	-	Layouts of history Japanese documents

Table 1: Current layout detection models in the LayoutParser model zoo. For each dataset, we train several models of different sizes for different needs (the trade-off between accuracy vs. computational cost). For "base model" and "large model", we refer to using the ResNet 50 or ResNet 101 backbones [12], respectively. One can train models of different architectures, like Faster R-CNN [24] (F) and Mask R-CNN [11] (M). For example, an F in the Large Model column indicates it has a Faster R-CNN model trained using the ResNet 101 backbone. The platform is maintained and a number of additions will be made to the model zoo in coming months.

highly customizable, via its integration with functions for *layout data annotation and model training*. We now provide detailed descriptions for each component.

3.1 Layout Detection Models

In LayoutParser, a layout model takes a document image as an input and generates a list of rectangular boxes for the target content regions. Different from traditional methods, it relies on deep convolutional neural networks rather than manually curated rules to identify content regions. It is formulated as an object detection problem and state-of-the-art models like Faster R-CNN [24] and Mask R-CNN [11] are used. This yields prediction results of high accuracy and makes it possible to build a concise, generalized interface for layout detection. LayoutParser, built upon Detectron2 [31], provides a minimal API that can perform layout detection with only four lines of code in Python:

```
import layoutparser as lp
image = cv2.imread("image_file") # load images
model = lp.Detectron2LayoutModel(
    "lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config")
layout = model.detect(image)
```

LayoutParser provides a wealth of pre-trained model weights using various datasets covering different languages, time periods, and document types. Due to domain shift [6], the prediction performance can notably drop when models are applied to target samples that are significantly different from the training dataset. As document structures and layouts vary greatly in different domains, it is important to select models trained on a similar dataset to the test samples for better performance. A semantic syntax is used for initializing the model weights in LayoutParser, using both the dataset name and model name `lp://<dataset-name>/<model-architecture-name>`. Shown in Table 1, LayoutParser currently hosts 9 pre-trained models trained on 5 different datasets.

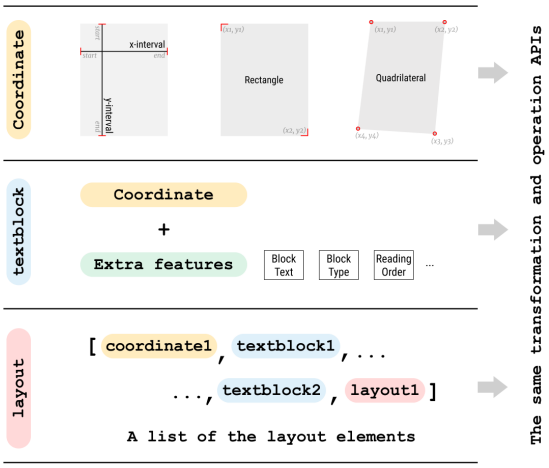


Fig. 2: The relationship between the three types of layout data structures. `Coordinate` supports three kinds of variation; `TextBlock` consists of the coordinate information and extra features like block text, types, and reading orders; a `Layout` object is a list of all possible layout elements, including other `Layout` objects. They all support the same set of transformation and operation APIs for maximum flexibility.

Description of the training dataset is provided alongside with the trained models such that users can quickly identify the most suitable models for their tasks. Additionally, when such a model is not readily available, `LayoutParser` also supports training customized layout models and community sharing of the models (detailed in Section 3.5).

3.2 Layout Data Structures

A critical feature of `LayoutParser` is the implementation of a series of data structures and operations that can be used to efficiently process and manipulate the layout elements. In document image analysis pipelines, various post-processing on the layout analysis model outputs is usually required to obtain the final outputs. Traditionally, this requires exporting DL model outputs and then loading the results into other pipelines. All model outputs from `LayoutParser` will be stored in carefully engineered data types optimized for further processing, which makes it possible to build an end-to-end document digitization pipeline within `LayoutParser`. The are three key components in the data structure, namely the `Coordinate` system, the `TextBlock`, and the `Layout`. They provide different levels of abstraction for the layout data, and a set of APIs are supported for transformations or operations on these classes.

Coordinates are the cornerstones for storing layout information. Currently, three types of `Coordinate` data structures are provided in `LayoutParser`, shown

Operation Name	Description
`block.pad(top, bottom, right, left)`	Enlarge the current block according to the input
`block.scale(fx, fy)`	Scale the current block given the ratio in x and y direction
`block.shift(dx, dy)`	Move the current block with the shift distances in x and y direction
`block1.is_in(block2)`	Whether block1 is inside of block2
`block1.intersect(block2)`	Return the intersected region of block1 and block2. Coordinate type to be determined based on the inputs.
`block1.union(block2)`	Return the union region of block1 and block2. Coordinate type to be determined based on the inputs.
`block1.relative_to(block2)`	Convert the absolute coordinates of block1 to relative coordinates to block2
`block1.condition_on(block2)`	Calculate the absolute coordinates of block1 given the canvas block2's absolute coordinates
`block.crop_image(image)`	Obtain the image segments in the block region

Table 2: All operations supported by the layout elements. The same APIs are supported across different layout element classes including `Coordinate` types and `TextBlock` and `Layout`.

in Figure 2. `Interval` and `Rectangle` are the most common data types and support specifying 1D or 2D regions within a document. They are parameterized with 2 and 4 parameters. A `Quadrilateral` class is also implemented to support a more generalized representation of rectangular regions when the document is skewed or distorted, where the 4 corner points can be specified and a total of 8 degree of freedom is supported. A wide collection of transformations like `shift`, `pad`, and `scale`, and operations like `intersect`, `union`, and `is_in`, are supported for these classes. Notably, it is common to separate a segment of the image and analyze it individually. `LayoutParser` provides full support for this scenario via image cropping operations `crop_image` and coordinate transformations like `relative_to` and `condition_on` that transform coordinates to and from their relative representations. We refer readers to table 2 for a more detailed description of these operations[12].

Based on `Coordinate`s, we implement the `TextBlock` class that stores both the positional and extra features of individual layout elements like text, index, and reading orders. A `Layout` class is built that takes in a list of `TextBlock`s and supports processing the elements in batch. `Layout` could also be nested to support hierarchical layout structures. These support the same operations and transformations as the `Coordinate` classes, minimizing both the learning and deployment effort.

[12] This is also available in the `LayoutParser` documentation pages.

3.3 OCR

`LayoutParser` provides a unified interface for existing OCR tools. Though there are many OCR tools available, they are usually configured differently with distinct APIs or protocols for using them. It is less efficient to add new OCR tools into an existing pipeline, and difficult to make direct comparisons among the available tools to find the best option for a particular project. To this end, `LayoutParser` builds a series of wrappers among existing OCR engines, and provides nearly the same syntax for using them. It supports the plug-and-play style of using the OCR engines, making it effortless to switch, evaluate, and compare different OCR modules:

```
ocr_agent = lp.TesseractAgent()
# Can be easily switched to other OCR software
tokens = ocr_agent.detect(image)
```

The OCR outputs will also be stored in the aforementioned layout data structures and can be seamlessly incorporated into the digitization pipeline. Currently `LayoutParser` supports the Tesseract and Google Cloud Vision OCR engines, and more will be supported in the near future.

`LayoutParser` also comes with a DL-based CNN-RNN OCR model [5] trained with the Connectionist Temporal Classification (CTC) loss [9]. It can be used like the other OCR modules, and can be easily trained on customized datasets.

3.4 Storage and visualization

The end goal of document analysis is to transform the image-based document data into a structured and organized format. `LayoutParser` supports exporting layout data into different formats like `JSON` and `csv`. We also support loading datasets from layout analysis-specific formats like COCO [34] and the Page Format [2] for training layout models (Section 3.5).

Visualization of the layout detection results is critical for both presentation and debugging. `LayoutParser` is built with an integrated API for displaying the layout information along with the original document image. Shown in Figure 3, it enables presenting layout data with rich meta information and features in different modes.

```
lp.draw_box(image, layout) # Mode I
lp.draw_text(image, tokens) # Mode II
```

More detailed information can be found in the online `LayoutParser` documentation page.

3.5 Customized Model Training

Besides the off-the-shelf library, `LayoutParser` is also highly customizable with supports for the most unique and challenging document analysis tasks. Target document images can be vastly different from the existing datasets for training layout models, which leads to low layout detection accuracy. Training data

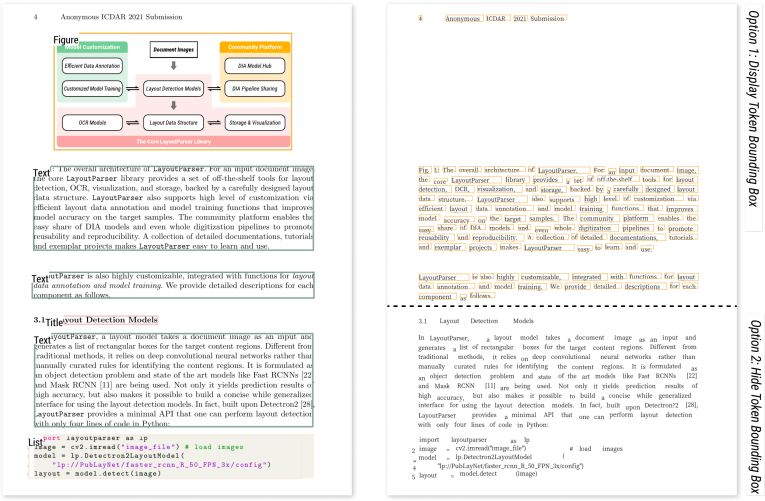


Fig. 3: Layout detection and OCR results visualization generated by the `LayoutParser` APIs. Mode I directly overlays the layout region bounding boxes and categories over the original image. Mode II recreates the original document via drawing the OCR'd texts at their corresponding positions on the image canvas. In this figure, tokens in textual regions are filtered using the API and displayed.

can also be highly sensitive and not sharable publicly. To overcome these challenges, `LayoutParser` is built with rich features for efficient data annotation and customized model training.

`LayoutParser` incorporates a recent toolkit optimized for annotating document layouts using object-level active learning [28]. With the help from a layout detection model trained along with labeling, only the most important layout objects within each image, rather than the whole image, are required for labeling. The rest of the regions are automatically annotated with high confidence predictions from the layout detection model. This allows a layout dataset to be created more efficiently with only around 60% of the labeling budget.

After the training dataset is curated, `LayoutParser` supports different modes for training the layout models. *Fine-tuning* can be used for training models on a newly-labeled *small* dataset by initializing the model with existing pre-trained weights. *Training from scratch* can be helpful when the source dataset and target are significantly different and a large training set is available. However, as suggested in [29], loading pre-trained weights on large-scale datasets like ImageNet [4], even from totally different domains, can still boost model performance. Through the integrated API provided by `LayoutParser`, users can easily compare model performances on the benchmark datasets.

4 LayoutParser Community Platform

Another focus of LayoutParser is promoting the reusability of layout detection models and the whole digitization pipeline. Similar to many existing deep learning libraries, LayoutParser comes with a community model hub for distributing layout models. End-users can upload their self-trained models to the model hub, and these models can be loaded in a similar interface as the currently available LayoutParser pre-trained models. For example, the model trained on the News Navigator dataset [15] has been incorporated in the model hub.

Beyond DL models, LayoutParser also helps share the entire document digitization pipeline. For example, sometimes the pipeline requires the combination of multiple DL models to achieve better accuracy. Currently, pipelines are mainly described in academic papers and implementations are often not publicly available. To this end, the LayoutParser community platform also enables the sharing of layout pipelines to promote the discussion and reuse of techniques. For each shared pipeline, it has a dedicated project page, with links to the source code, documentation, and an outline of the approaches. A discussion panel is provided for exchanging ideas for different approaches, which would benefit both the project authors and the audience. Combined with the core LayoutParser library, users can easily build reusable components based on the shared pipelines and apply them to solve their unique problems.

5 Use Cases

The core objective of LayoutParser is to make it easier to create both large-scale and light-weight document digitization pipelines. Large-scale document processing focuses on precision, efficiency, and robustness. The target documents may have complicated structures, and may require training multiple layout detection models to achieve the optimal accuracy. Light-weight pipelines are built for relatively simple documents, with an emphasis on development ease, speed and flexibility. Ideally one only needs to use existing resources, and model training should be avoided. Through two exemplar projects, we show how practitioners in both academia and industry can easily build such pipelines using LayoutParser and extract high-quality structured document data for their downstream tasks. The source code for these projects will be publicly available in the LayoutParser community hub.

5.1 A Comprehensive Historical Document Digitization Pipeline

The digitization of historical documents can unlock valuable data that can shed light on many important social, economic, and historical questions. Yet due to scan noises, page wearing, and the prevalence of complicated layout structures, obtaining a structured representation of historical document scans is extremely complicated. In this example, a comprehensive pipeline is developed using LayoutParser that generates high-quality structured data from historical

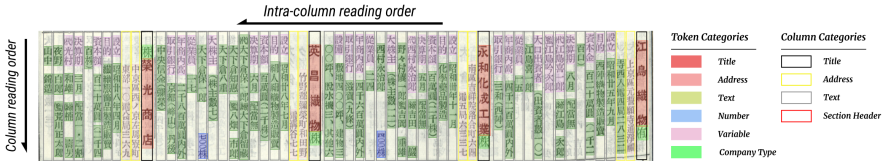


Fig. 4: Illustration of a historical Japanese document showing the column and token bounding boxes and categories (color) detected by the layout models.

Fig. 5: Illustration of a recreated document image that achieves a much better character recognition recall rate. The reorganization algorithm rearranges the tokens based on the their detected bounding boxes given a maximum allowed height.

Japanese firm financial tables with complicated layouts. It applies two layout models to identify different levels of document structures and two customized OCR engines for optimized character recognition accuracy.

As shown in Figure 4, the document contains columns of text written vertically [13], a common style in Japanese. Due to scanning noise and archaic printing technology, the columns can be skewed or have variable width that cannot be easily identified via rule-based methods. Within each column, words are separated by white spaces of variable size, and the vertical positions of objects can be an indicator for their layout type.

To decipher the complicated layout structure, two object detection models have been trained to recognize individual columns and tokens, respectively. A small training set (400 images with approximately 100 annotations each) is curated via the active learning based annotation tool [28] in `LayoutParser`. The models learn to identify both the categories and regions for each column (token) via their distinct visual features. The layout data structure enables easy grouping of the tokens within each column, and rearranging columns to achieve the correct reading orders based on the horizontal position. Errors are identified and rectified via checking the consistency of the model predictions. Therefore, though trained on a small dataset, the pipeline achieves a high level of layout detection accuracy: it achieves a 96.97 AP [17] score across 5 categories for the column detection model, and a 89.23 AP across 4 categories for the token detection model.

[13] A document page consists of eight rows like this. For simplicity we skip the row segmentation discussion and refer readers to the source code when available.

A combination of character recognition methods is developed to tackle the unique challenges in this document. In our experiments, we find that irregular spacing between the tokens leads to low character recognition recall rate, and existing OCR models tend to perform better on densely-arranged texts. To overcome this challenge, we create a document reorganization algorithm that rearranges the text based on the token bounding boxes detected in the layout analysis step. Figure 5 illustrates the generated image of dense text, which is sent to the OCR APIs as a whole to reduce the transaction costs. The flexible coordinate system in `LayoutParser` is used for transforming the OCR results relative to their original positions on the page.

Additionally, it is common that historical documents may use unique fonts with different glyphs, which significantly degrades the accuracy of OCR models trained on modern texts. In this document, a special flat font is used for printing numbers and could not be detected by off-the-shelf OCR engines. Using the highly flexible functionalities from `LayoutParser`, a pipeline approach is constructed that achieves a satisfactory recognition accuracy with minimal effort. As the characters have unique visual structures and are usually clustered together, we train the layout model to identify number regions with a dedicated category. Subsequently, `LayoutParser` crops images within these regions, and identifies characters within them using a self-trained OCR model based on CNN-RNN [5]. The model detects a total of 15 possible categories, and achieves a 0.98 Jaccard score[14] and a 0.17 average Levinstein distances[15] for token prediction on the test set.

Overall, it is possible to create an intricate and highly accurate digitization pipeline for large-scale digitization using `LayoutParser`. The pipeline avoids specifying the complicated rules used in traditional methods, is straightforward to develop, and is robust to outliers. The DL models also generate fine-grained results that enable creative approaches like page reorganization for OCR.

5.2 A light-weight Visual Table Extractor

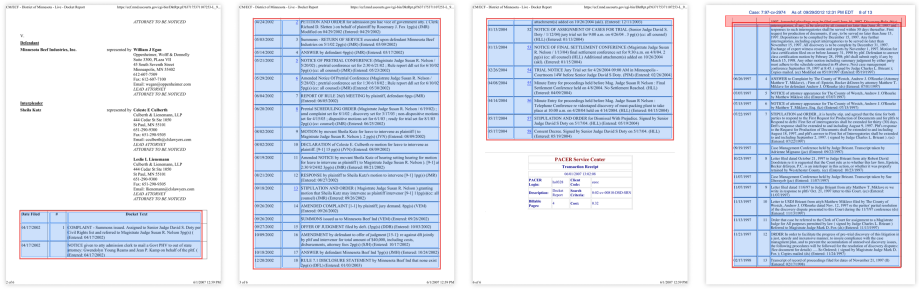
Detecting tables and parsing their structures (table extraction) are of central importance for many document digitization tasks. Many previous works [22, 26, 23] and tools [16] have been developed to identify and parse table structures. Yet they might require training complicated models from scratch, or could only be used for born-digital PDF documents. In this section, we show how `LayoutParser` can help build a light-weight accurate visual table extractor for legal docket tables using the existing resources with minimal effort.

The extractor uses a pre-trained layout detection model for identifying the table regions and some simple rules for pairing the rows and the columns in the

[14] This measures the overlap between the detected and ground-truth characters, and the maximum is 1.

[15] This measures the number of edits from the ground-truth text to the predicted text, and lower is better.

[16] https://github.com/atlanhq/camelot, https://github.com/tabulapdf/tabula



(a) Partial table at the bottom (b) Full page table (c) Partial table at the top (d) Mis-detected text line

Fig. 6: This lightweight table detector can identify tables (outlined in red) and cells (shaded in blue) in different locations on a page. In very few cases (d), it might generate minor error predictions, e.g, failing to capture the top text line of a table.

PDF image. Mask R-CNN [11] trained on the PubLayNet dataset [34] from the LayoutParser Model Zoo can be used for detecting table regions. By filtering out model predictions of low confidence and removing overlapping predictions, LayoutParser can identify the tabular regions on each page, which significantly simplifies the subsequent steps. By applying the line detection functions within the tabular segments, provided in the utility module from LayoutParser, the pipeline can identify the three distinct columns in the tables. A row clustering method is then applied via analyzing the y coordinates of token bounding boxes in the left-most column, which are obtained from the OCR engines. A non-maximal suppression algorithm is used to remove duplicated rows with extremely small gaps. Shown in Figure 6, the built pipeline can detect tables at different positions on a page accurately. Continued tables from different pages are concatenated, and a structured table representation has been easily created.

6 Conclusion

LayoutParser provides a comprehensive toolkit for deep learning-based document image analysis. The off-the-shelf library is easy to install, and can be used to build flexible and accurate pipelines for processing documents with complicated structures. It also supports high-level customization and enables easy labeling and training of DL models on unique document image datasets. The LayoutParser community platform facilitates sharing DL models and DIA pipelines, inviting discussion and promoting code reproducibility and reusability. The LayoutParser team is committed to keeping the library updated continuously and bringing the most recent advances in DL-based DIA, such as multi-model document modeling [33, 32, 8] (an upcoming priority), to a diverse audience of end-users.

Acknowledgements This project is supported in part by NSF Grant OIA-2033558 and fundings from Harvard Data Science Initiative and Harvard Catalyst. Zejiang Shen thanks the suggestions from Doug Downey.

References

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), `https://www.tensorflow.org/`, software available from tensorflow.org

[2] Antonacopoulos, A., Bridson, D., Papadopoulos, C., Pletschacher, S.: A realistic dataset for performance evaluation of document layout analysis. In: 2009 10th International Conference on Document Analysis and Recognition. pp. 296–300. IEEE (2009)

[3] Baek, Y., Lee, B., Han, D., Yun, S., Lee, H.: Character region awareness for text detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. pp. 9365–9374 (2019)

[4] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: ImageNet: A Large-Scale Hierarchical Image Database. In: CVPR09 (2009)

[5] Deng, Y., Kanervisto, A., Ling, J., Rush, A.M.: Image-to-markup generation with coarse-to-fine attention. In: International Conference on Machine Learning. pp. 980–989. PMLR (2017)

[6] Ganin, Y., Lempitsky, V.: Unsupervised domain adaptation by backpropagation. In: International conference on machine learning. pp. 1180–1189. PMLR (2015)

[7] Gardner, M., Grus, J., Neumann, M., Tafjord, O., Dasigi, P., Liu, N., Peters, M., Schmitz, M., Zettlemoyer, L.: Allennlp: A deep semantic natural language processing platform. arXiv preprint arXiv:1803.07640 (2018)

[8] Łukasz Garncarek, Powalski, R., Stanisławek, T., Topolski, B., Halama, P., Graliński, F.: Lambert: Layout-aware (language) modeling using bert for information extraction (2020)

[9] Graves, A., Fernández, S., Gomez, F., Schmidhuber, J.: Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In: Proceedings of the 23rd international conference on Machine learning. pp. 369–376 (2006)

[10] Harley, A.W., Ufkes, A., Derpanis, K.G.: Evaluation of deep convolutional nets for document image classification and retrieval. In: 2015 13th International Conference on Document Analysis and Recognition (ICDAR). pp. 991–995. IEEE (2015)

[11] He, K., Gkioxari, G., Dollár, P., Girshick, R.: Mask r-cnn. In: Proceedings of the IEEE international conference on computer vision. pp. 2961–2969 (2017)

[12] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)

[13] Kay, A.: Tesseract: An open-source optical character recognition engine. Linux J. **2007**(159), 2 (Jul 2007)

[14] Lee, B.C., Weld, D.S.: Newspaper navigator: Open faceted search for 1.5 million images. In: Adjunct Publication of the 33rd Annual ACM Symposium on User Interface Software and Technology. p. 120–122. UIST '20 Adjunct, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3379350.3416143, `https://doi-org.offcampus.lib.washington.edu/10.1145/3379350.3416143`

[15] Lee, B.C.G., Mears, J., Jakeway, E., Ferriter, M., Adams, C., Yarasavage, N., Thomas, D., Zwaard, K., Weld, D.S.: The Newspaper Navigator Dataset: Extracting Headlines and Visual Content from 16 Million Historic Newspaper Pages in Chronicling America, p. 3055–3062. Association for Computing Machinery, New York, NY, USA (2020), https://doi.org/10.1145/3340531.3412767
[16] Li, M., Cui, L., Huang, S., Wei, F., Zhou, M., Li, Z.: Tablebank: Table benchmark for image-based table detection and recognition. arXiv preprint arXiv:1903.01949 (2019)
[17] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., Zitnick, C.L.: Microsoft coco: Common objects in context. In: European conference on computer vision. pp. 740–755. Springer (2014)
[18] Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 3431–3440 (2015)
[19] Oliveira, S.A., Seguin, B., Kaplan, F.: dhsegment: A generic deep-learning approach for document segmentation. In: 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR). pp. 7–12. IEEE (2018)
[20] Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch (2017)
[21] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al.: Pytorch: An imperative style, high-performance deep learning library. arXiv preprint arXiv:1912.01703 (2019)
[22] Prasad, D., Gadpal, A., Kapadni, K., Visave, M., Sultanpure, K.: 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. pp. 572–573 (2020)
[23] Qasim, S.R., Mahmood, H., Shafait, F.: Rethinking table recognition using graph neural networks. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 142–147. IEEE (2019)
[24] Ren, S., He, K., Girshick, R., Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In: Advances in neural information processing systems. pp. 91–99 (2015)
[25] Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE transactions on neural networks **20**(1), 61–80 (2008)
[26] Schreiber, S., Agne, S., Wolf, I., Dengel, A., Ahmed, S.: 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). vol. 1, pp. 1162–1167. IEEE (2017)
[27] Shen, Z., Zhang, K., Dell, M.: A large dataset of historical japanese documents with complex layouts. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops. pp. 548–549 (2020)
[28] Shen, Z., Zhao, J., Dell, M., Yu, Y., Li, W.: Olala: Object-level active learning based layout annotation. arXiv preprint arXiv:2010.01762 (2020)
[29] Studer, L., Alberti, M., Pondenkandath, V., Goktepe, P., Kolonko, T., Fischer, A., Liwicki, M., Ingold, R.: A comprehensive study of imagenet pre-training for historical document image analysis. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 720–725. IEEE (2019)
[30] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., et al.: Huggingface's transformers: State-of-the-art natural language processing. arXiv preprint arXiv:1910.03771 (2019)

[31] Wu, Y., Kirillov, A., Massa, F., Lo, W.Y., Girshick, R.: Detectron2. `https://github.com/facebookresearch/detectron2` (2019)
[32] Xu, Y., Xu, Y., Lv, T., Cui, L., Wei, F., Wang, G., Lu, Y., Florencio, D., Zhang, C., Che, W., et al.: Layoutlmv2: Multi-modal pre-training for visually-rich document understanding. arXiv preprint arXiv:2012.14740 (2020)
[33] Xu, Y., Li, M., Cui, L., Huang, S., Wei, F., Zhou, M.: Layoutlm: Pre-training of text and layout for document image understanding (2019)
[34] Zhong, X., Tang, J., Yepes, A.J.: Publaynet: largest dataset ever for document layout analysis. In: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 1015–1022. IEEE (Sep 2019). https://doi.org/10.1109/ICDAR.2019.00166