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

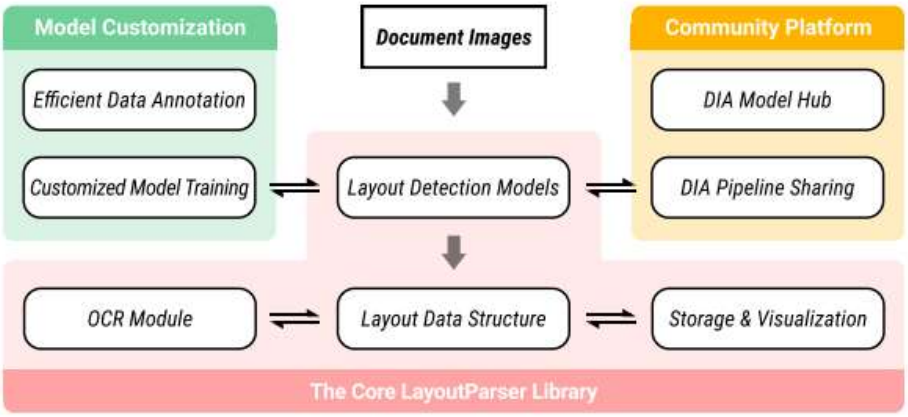
Paper introduces Layout Parser, an open source library for streamlining the usage of Deep Learning(DL) in Document Image Analysis(DIA) research and applications.

Practical difficulties in DL-based approach:

1. Existing models are developed using distinct frameworks like Tensorflw or Pytorch and the high level parameters can be obfuscated by implementation details. It can be time consuming to debug, reproduce and adapt existing models for DIA and many researchers lack the technical background to implement them from scratch.
2. Document images contain diverse and disparate patterns across domains and customized training is often 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 desired output. Often document analysis and processing are performed as other tasks separate from DL model tasks and hence they are not included or documented well enough in the projects. This makes it difficult for the researchers to learn how full pipelines are implemented and leads them to invest significant resources in reinventing the DIA wheel.

Components of Layout Parser:

- ➔ An off the shelf toolkit for applying DL models for layout detection, character recognition and other tasks.
- ➔ A rich repo of pre-trained models that underlines the off-the-shelf usage.
- ➔ Comprehensive tools for efficient document image data annotation and model tuning to support different levels of customisation.
- ➔ A DL model hub and community platform for easy sharing of DIA models and pipelines, to promote reusability, reproducibility and extensibility.



- **Layout Detection Model**

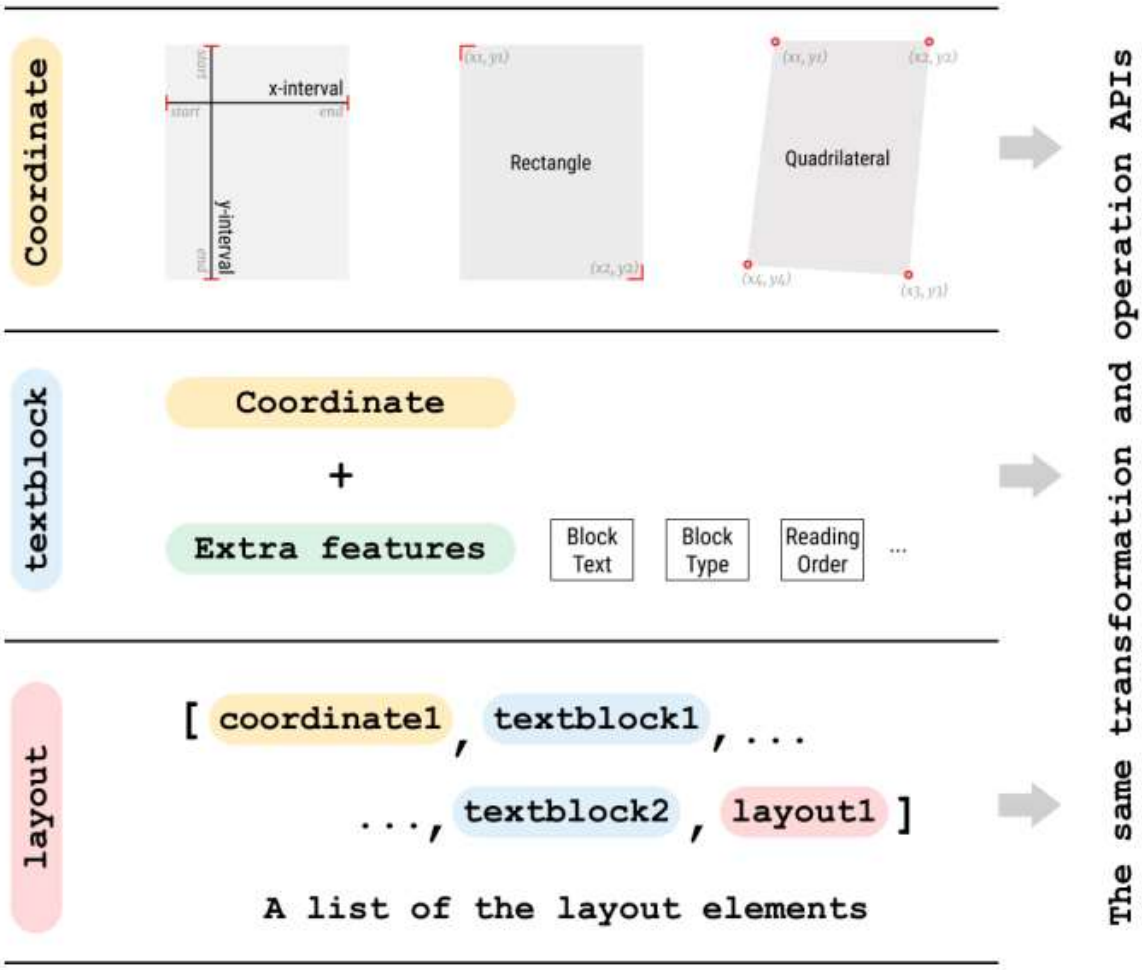
In LAyoutParsor, a layout detection model takes a document image as an input and generates a list of rectangular boxes for the target content regions. It relies on Deep Convolutional Neural Networks rather than manually created rules to identify content regions.
It is formulated as an Object Detection Problem with models like Faster-RCNN, Mask-RCNN. This yields prediction results of high accuracy and makes it possible to build a concise, generalized interface for layout detection.

Table 1: Current layout detection models in the `LayoutParser` model zoo

Dataset	Base Model[1]	Large Model	Notes
PubLayNet [38]	F / M	M	Layouts of modern scientific documents
PRImA [3]	M	-	Layouts of scanned modern magazines and scientific reports
Newspaper [17]	F	-	Layouts of scanned US newspapers from the 20th century
TableBank [18]	F	F	Table region on modern scientific and business document
HJDataset [31]	F / M	-	Layouts of history Japanese documents

LayoutParser currently hosts 9 pre-trained models trained on 5 different datasets. It also supports training customized layout models.

- **Layout Data Structures**



All model outputs from LayoutParser will be stored in carefully engineered data types optimized for further processing.
Here, there are three key components in data structures:

- **Coordinates** support three kinds of variations; **Interval** and **Rectangle** specify 1D and 2D regions within a document. A Quadrilateral class is implemented to support a more generalized version of the Rectangular region when the document is skewed or distorted. A total of 8 degrees of freedom are supported.
- **TextBlock** consists of the coordinate information(position) 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 provide different levels of abstraction on the layout data and a set of APIs are supported for transformations and operations on these classes.

- **OCR**

LayoutParser builds a series of wrappers among the existing OCR engines and provides nearly the same index for using them, making it effortless to switch, evaluate, and compare different OCR modules.

Currently, LayoutParser supports **Tesseract** and **Google Cloud Vision** OCR engines. LayoutParser also comes with a **DL-based CNN-RNN OCR** model trained with the Connectionist Temporal Classification loss which can be used like any other OCRs and trained further with custom dataset.

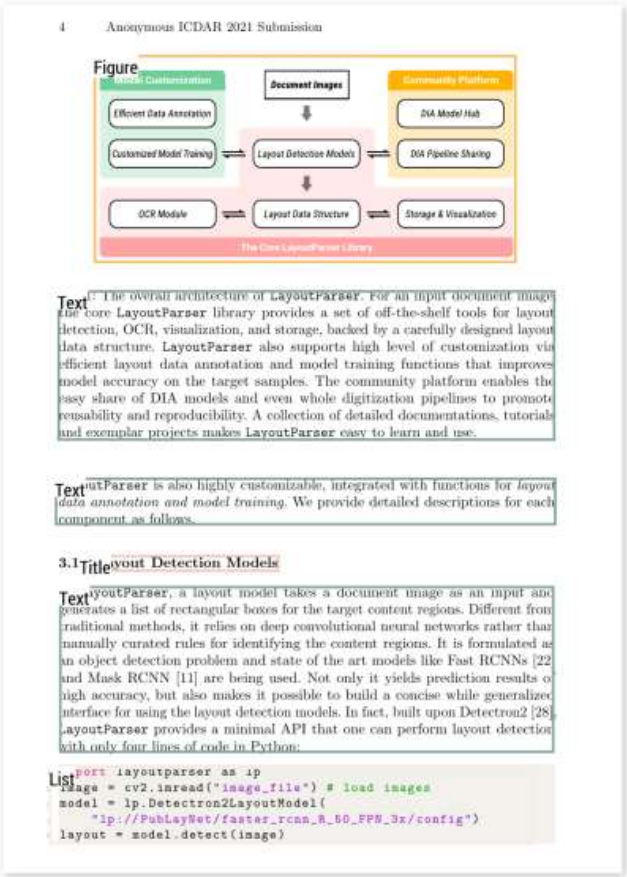
Operations supported by the layout elements:

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 intersection 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

- **Storage and Visualization**

The end goal of DIA is to transform the image based document into a structured database. LayoutParser supports exporting layout data into JSON, csv, and will add support for the METS/ALTO XML format.
It can also load datasets from layout analysis formats like COCO and the Page Format for training layout models.

LayoutParser is built with an integrated API for displaying the layout information along with the original documentation image:



Mode I: Showing Layout on the Original Image

Mode II: Drawing OCR'd Text at the Correspoding Position

- **Customized Model Training**

LayoutParser incorporates a toolkit optimized for annotating document layouts using object-level active learning. With the help of 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 small newly-labeled 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.

LayoutParser Community Platform

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 could be loaded into a similar interface.
LayoutParser also promotes the sharing of entire document digitization pipelines to promote the discussion and reuse of techniques. 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.

Use Cases

The core step of LayoutParser is to make it easier to create both large-scale and lightweight document digitization pipelines.

A Comprehensive Historical Document Digitization Pipeline
LayoutParser was used to develop a comprehensive pipeline to generate high-quality structured data from historical Japanese firm financial tables with complicated layouts. The pipeline applies two layout models to identify different levels of document structures and two customized OCR engines for optimized character recognition accuracy

A light-weight Visual Table Extractor

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 PDF image.By applying the line detection functions within the table 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 duplicate rows with extremely small gaps. 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.

Conclusion

LayoutParser provides a comprehensive toolkit for deep learning-based document image analysis.