Layout Parser

Release 0.3.2

Layout Parser Contributors

Apr 02, 2022

NOTES

CHAPTER

ONE

INSTALLATION

1.1 Install Python

LayoutParser is a Python package that requires Python >= 3.6. If you do not have Python installed on your computer, you might want to turn to the official instruction to download and install the appropriate version of Python.

1.2 Install the LayoutParser library

After several major updates, LayoutParser provides various functionalities and deep learning models from different backends. However, you might only need a fraction of the functions, and it would be redundant for you to install all the dependencies when they are not required. Therefore, we design highly customizable ways for installing the LayoutParser library:

1.2.1 Additional Instruction: Install Detectron2 Layout Model Backend

For Mac OS and Linux Users

If you would like to use the Detectron2 models for layout detection, you might need to run the following command:

```
pip install layoutparser torchvision && pip install "detectron2@git+https://github.
↪com/facebookresearch/detectron2.git@v0.5#egg=detectron2"
```

This might take some time as the command will *compile* the library. If you also want to install a Detectron2 version with GPU support or encounter some issues during the installation process, please refer to the official Detectron2 installation instruction for detailed information.

For Windows users

As reported by many users, the installation of Detectron2 can be rather tricky on Windows platforms. In our extensive tests, we find that it is nearly impossible to provide a one-line installation command for Windows users. As a workaround solution, for now we list the possible challenges for installing Detectron2 on Windows, and attach helpful resources for solving them. We are also investigating other possibilities to avoid installing Detectron2 to use pre-trained models. If you have any suggestions or ideas, please feel free to submit an issue in our repo.

1. Challenges for installing `pycocotools`
   - You can find detailed instructions on this post from Chang Hsin Lee.
   - Another solution is try to install `pycocotools-windows`, see https://github.com/cocodataset/cocoapi/issues/415.

2. Challenges for installing `Detectron2`
    - @ivanpp curates a detailed description for installing `Detectron2` on Windows: Detectron2 walkthrough (Windows)
    - `Detectron2` maintainers claim that they won't provide official support for Windows (see 1 and 2), but Detectron2 is continuously built on windows with CircleCI (see 3). Hopefully this situation will be improved in the future.

1.2.2 Additional Instructions: Install OCR utils

Layout Parser also comes with supports for OCR functions. In order to use them, you need to install the OCR utils via:

```
pip install "layoutparser[ocr]"
```

Additionally, if you want to use the Tesseract-OCR engine, you also need to install it on your computer. Please check the official documentation for detailed installation instructions.

1.3 Known issues

In this case, you have a newer version of the google-cloud-vision. Please consider downgrading the API using:

```
pip install -U layoutparser[ocr]
```

CHAPTER

TWO

MODEL ZOO

We provide a spectrum of pre-trained models on different datasets.

2.1 Example Usage:

```
import layoutparser as lp
model = lp.Detectron2LayoutModel(
            config_path ='lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config', # In model␣
↪catalog
            label_map   ={0: "Text", 1: "Title", 2: "List", 3:"Table", 4:"Figure"}, #␣
↪In model`label_map`
            extra_config=["MODEL.ROI_HEADS.SCORE_THRESH_TEST", 0.8] # Optional
        )
model.detect(image)
```

2.2 Model Catalog

- For PubLayNet models, we suggest using `mask_rcnn_X_101_32x8d_FPN_3x` model as it's trained on the whole training set, while others are only trained on the validation set (the size is only around 1/50). You could expect a 15% AP improvement using the `mask_rcnn_X_101_32x8d_FPN_3x` model.

2.3 Model `label_map`

CHAPTER

THREE

OCR TABLES AND PARSE THE OUTPUT

In this tutorial, we will illustrate how easily the `layoutparser` APIs can be used for

1. Recognizing texts in images and store the results with the specified OCR engine
2. Postprocessing of the textual results to create structured data

```
import layoutparser as lp

import matplotlib.pyplot as plt
%matplotlib inline

import pandas as pd
import numpy as np
import cv2
```

3.1 Initiate GCV OCR engine and check the image

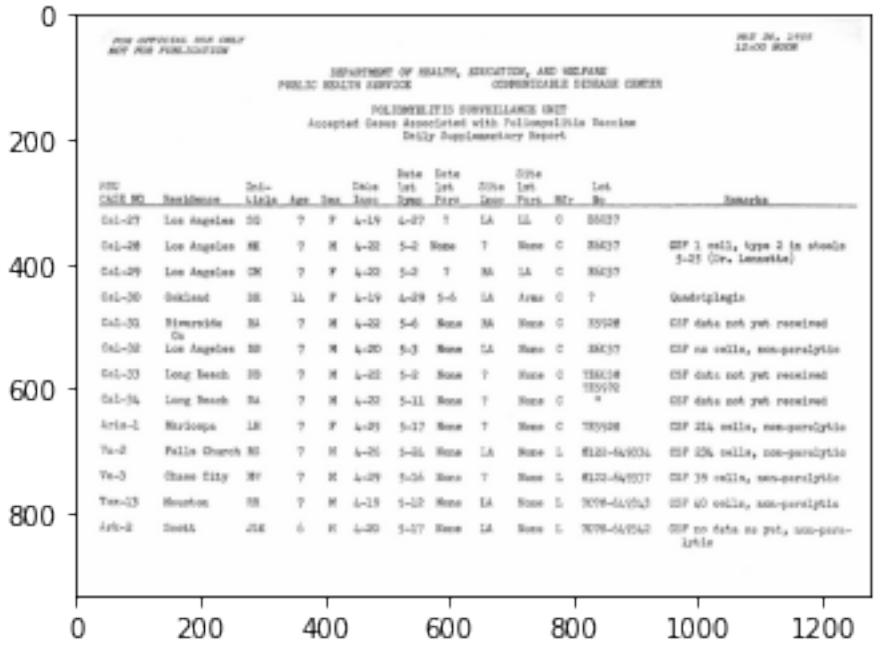
Currently, `layoutparser` supports two types of OCR engines: Google Cloud Vision and Tesseract OCR engine. And we are going to provide more support in the future. In this toturial, we will use the Google Cloud Vision engine as an example.

```
ocr_agent = lp.GCVAgent.with_credential("<path/to/your/credential>",
                                        languages = ['en'])
```

The `language_hints` tells the GCV which langeuage shall be used for OCRing. For a detailed explanation, please check here.

The `example-table` is a scan with complicated table structures from https://stacks.cdc.gov/view/cdc/42482/.

```
image = cv2.imread('data/example-table.jpeg')
plt.imshow(image);
```



3.2 Load images and send for OCR

The `ocr_agent.detect` method can take the image array, or simply the path of the image, for OCR. By default it will return the text in the image, i.e., `text = ocr_agent.detect(image)`.

However, as the layout is complex, the text information is not enough: we would like to directly analyze the response from GCV Engine. We can set the `return_response` to `True`. This feature is also supported for other OCR Engines like `TesseractOCRAgent`.

```
res = ocr_agent.detect(image, return_response=True)

# Alternative
# res = ocr_agent.detect('data/example-table.jpeg', return_response=True)
```

3.3 Parse the OCR output and visualize the layout

As defined by GCV, there are two different types of output in the response:

1. text_annotations:

   In this format, GCV automatically find the best aggregation level for the text, and return the results in a list. We can
   use the `ocr_agent.gather_text_annotations` to reterive this type of information.

2. full_text_annotations

To support better user control, GCV also provides the `full_text_annotation` output, where it returns the hierarchical structure of the output text. To process this output, we provide the `ocr_agent.gather_full_text_annotation` function to aggregate the texts of the given aggregation level.

There are 5 levels specified in `GCVFeatureType`, namely: `PAGE`, `BLOCK`, `PARA`, `WORD`, `SYMBOL`.

```
texts  = ocr_agent.gather_text_annotations(res)
    # collect all the texts without coordinates
layout = ocr_agent.gather_full_text_annotation(res, agg_level=lp.GCVFeatureType.WORD)
    # collect all the layout elements of the `WORD` level
```

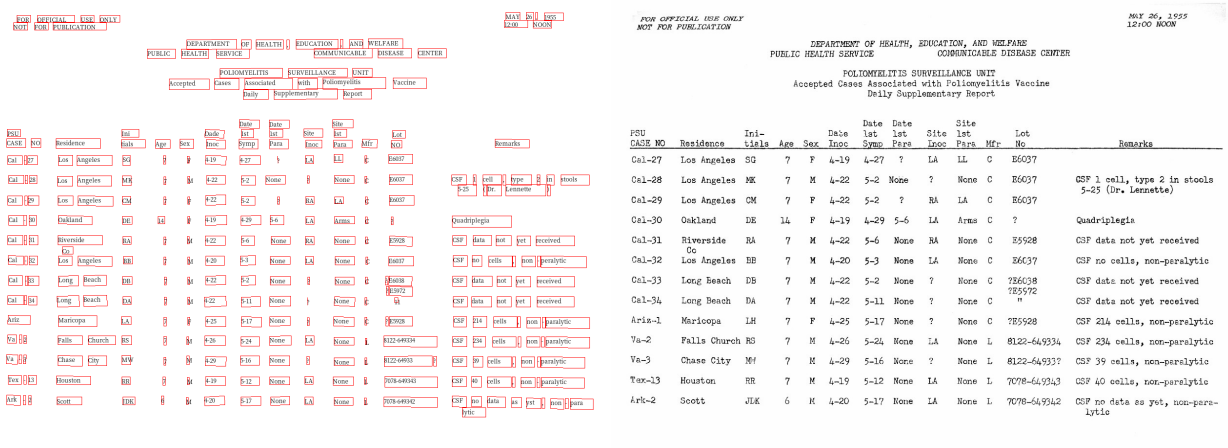
And we can use the `draw_box` or `draw_text` functions to quickly visualize the detected layout and text information.

These functions are highly customizable. You can change styles of the drawn boxes and texts easily. Please check the documentation for the detailed explanation of the configurable parameters.

As shown below, the `draw_text` function generates a visualization that:

1. it draws the detected layout with text on the left side and shows the original image on the right canvas for comparison.
2. on the text canvas (left), it also draws a red bounding box for each text region.

```
lp.draw_text(image, layout, font_size=12, with_box_on_text=True,
             text_box_width=1)
```

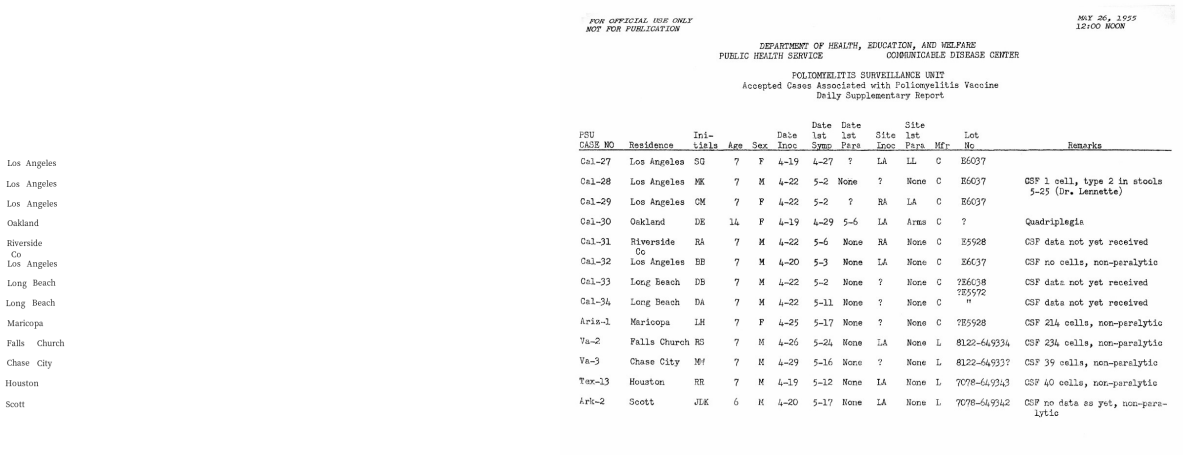


3.4 Filter the returned text blocks

We find the coordinates of residence column are in the range of $y \in (300, 833)$ and $x \in (132, 264)$. The `layout.filter_by` function can be used to fetch the texts in the region.

Note: As the OCR engine usually does not provide advanced functions like table detection, the coordinates are found manually by using some image inspecting tools like GIMP.

```
filtered_residence = layout.filter_by(
    lp.Rectangle(x_1=132, y_1=300, x_2=264, y_2=840)
)
lp.draw_text(image, filtered_residence, font_size=16)
```



Los Angeles
Los Angeles
Los Angeles
Oakland
Riverside Co
Los Angeles
Long Beach
Long Beach
Maricopa
Falls Church
Chase City
Houston
Scott

FOR OFFICIAL USE ONLY
NOT FOR PUBLICATION

MAY 26, 1955
12:00 NOON

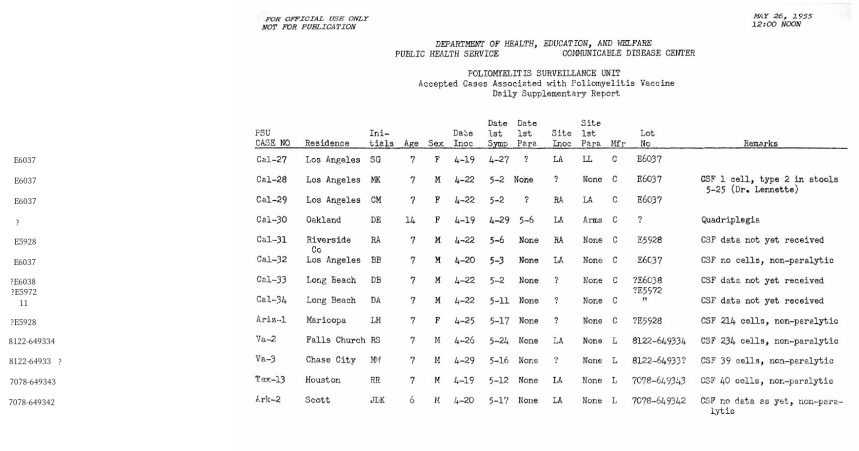
DEPARTMENT OF HEALTH, EDUCATION, AND WELFARE
PUBLIC HEALTH SERVICE COMMUNICABLE DISEASE CENTER

POLIOMYELITIS SURVEILLANCE UNIT
Accepted Cases Associated with Poliomyelitis Vaccine
Daily Supplementary Report

PSU CASE NO	Residence	Ini-tials	Age	Sex	Date Inoc	Date 1st Symp	Date 1st Para	Site Inoc	Site 1st Para	Mfr	Lot No	Remarks
Cal-27	Los Angeles	SG	7	F	4-19	4-27	?	LA	LL	C	E6037	
Cal-28	Los Angeles	MK	7	M	4-22	5-2	None	?	None	C	E6037	CSF 1 cell, type 2 in stools 5-25 (Dr. Lennette)
Cal-29	Los Angeles	CM	7	F	4-22	5-2	?	RA	LA	C	E6037	
Cal-30	Oakland	DE	14	F	4-19	4-29	5-6	LA	Arms	C	?	Quadriplegia
Cal-31	Riverside Co	RA	7	M	4-22	5-6	None	RA	None	C	E5928	CSF data not yet received
Cal-32	Los Angeles	BB	7	M	4-20	5-3	None	LA	None	C	E6037	CSF no cells, non-paralytic
Cal-33	Long Beach	DB	7	M	4-22	5-2	None	?	None	C	?E6038 ?E5972 11	CSF data not yet received
Cal-34	Long Beach	DA	7	M	4-22	5-11	None	?	None	C	"	CSF data not yet received
Ariz-1	Maricopa	LH	7	F	4-25	5-17	None	?	None	C	?E5928	CSF 214 cells, non-perelytic
Va-2	Falls Church	RS	7	M	4-26	5-24	None	LA	None	L	8122-649334	CSF 234 cells, non-paralytic
Va-3	Chase City	MH	7	M	4-29	5-16	None	?	None	L	8122-64933?	CSF 39 cells, non-paralytic
Tex-13	Houston	RR	7	M	4-19	5-12	None	LA	None	L	7078-649343	CSF 40 cells, non-paralytic
Ark-2	Scott	JLK	6	M	4-20	5-17	None	LA	None	L	7078-649342	CSF no data as yet, non-paralytic

And similarily, we can do that for the `lot_number` column. As sometimes there could be irregularities in the layout as well as the OCR outputs, the `layout.filter_by` function also supports a `soft_margin` argument to handle this issue and generate more robust outputs.

```
filter_lotno = layout.filter_by(
    lp.Rectangle(x_1=810, y_1=300, x_2=910, y_2=840),
    soft_margin = {"left":10, "right":20} # Without it, the last 4 rows could not be␣
↪included
)
lp.draw_text(image, filter_lotno, font_size=16)
```



E6037
E6037
E6037
?
E5928
E6037
?E6038
?E5972
11
?E5928
8122-649334
8122-64933 ?
7078-649343
7078-649342

FOR OFFICIAL USE ONLY
NOT FOR PUBLICATION

MAY 26, 1955
12:00 NOON

DEPARTMENT OF HEALTH, EDUCATION, AND WELFARE
PUBLIC HEALTH SERVICE COMMUNICABLE DISEASE CENTER

POLIOMYELITIS SURVEILLANCE UNIT
Accepted Cases Associated with Poliomyelitis Vaccine
Daily Supplementary Report

PSU CASE NO	Residence	Ini-tials	Age	Sex	Date Inoc	Date 1st Symp	Date 1st Para	Site Inoc	Site 1st Para	Mfr	Lot No	Remarks
Cal-27	Los Angeles	SG	7	F	4-19	4-27	?	LA	LL	C	E6037	
Cal-28	Los Angeles	MK	7	M	4-22	5-2	None	?	None	C	E6037	CSF 1 cell, type 2 in stools 5-25 (Dr. Lennette)
Cal-29	Los Angeles	CM	7	F	4-22	5-2	?	RA	LA	C	E6037	
Cal-30	Oakland	DE	14	F	4-19	4-29	5-6	LA	Arms	C	?	Quadriplegia
Cal-31	Riverside Co	RA	7	M	4-22	5-6	None	RA	None	C	E5928	CSF data not yet received
Cal-32	Los Angeles	BB	7	M	4-20	5-3	None	LA	None	C	E6037	CSF no cells, non-paralytic
Cal-33	Long Beach	DB	7	M	4-22	5-2	None	?	None	C	?E6038 ?E5972 11	CSF data not yet received
Cal-34	Long Beach	DA	7	M	4-22	5-11	None	?	None	C	"	CSF data not yet received
Ariz-1	Maricopa	LH	7	F	4-25	5-17	None	?	None	C	?E5928	CSF 214 cells, non-perelytic
Va-2	Falls Church	RS	7	M	4-26	5-24	None	LA	None	L	8122-649334	CSF 234 cells, non-paralytic
Va-3	Chase City	MH	7	M	4-29	5-16	None	?	None	L	8122-64933?	CSF 39 cells, non-paralytic
Tex-13	Houston	RR	7	M	4-19	5-12	None	LA	None	L	7078-649343	CSF 40 cells, non-paralytic
Ark-2	Scott	JLK	6	M	4-20	5-17	None	LA	None	L	7078-649342	CSF no data as yet, non-paralytic

3.4.1 Group Rows based on hard-coded parameteres

As there are 13 rows, we can iterate the rows and fetch the row-based information:

```
y_0 = 307
n_rows = 13
height = 41
y_1 = y_0+n_rows*height

row = []
for y in range(y_0, y_1, height):

    interval = lp.Interval(y,y+height, axis='y')
```

(continues on next page)

(continued from previous page)

```
    residence_row = filtered_residence.\
        filter_by(interval).\
        get_texts()

    lotno_row = filter_lotno.\
        filter_by(interval).\
        get_texts()

    row.append(['\n'.join(residence_row), '\n'.join(lotno_row)])
```

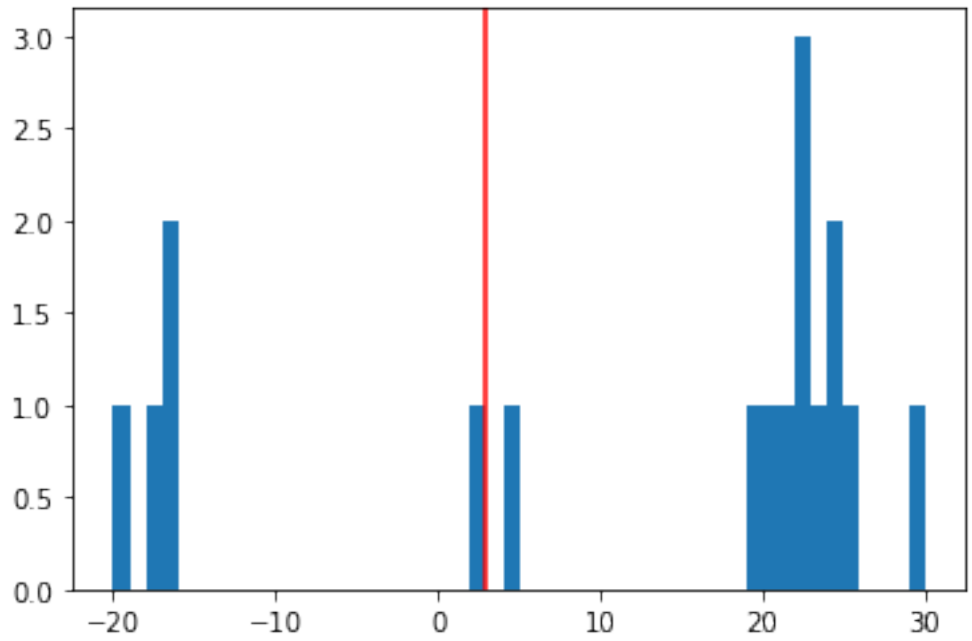
```
row
```

```
[['LosAngeles', 'E6037'],
 ['LosAngeles', 'E6037'],
 ['LosAngeles', 'E6037'],
 ['Oakland', '?'],
 ['Riverside', 'E5928'],
 ['LosAngeles', 'E6037'],
 ['LongBeach', '?E6038'],
 ['LongBeach', '11'],
 ['Maricopa', '?E5928'],
 ['FallsChurch', '8122-649334'],
 ['ChaseCity', '8122-64933?'],
 ['Houston', '7078-649343'],
 ['Scott', '7078-649342']]
```

3.4.2 An Alternative Method - Adaptive Grouping lines based on distances

```
blocks = filter_lotno

blocks = sorted(blocks, key = lambda x: x.coordinates[1])
    # Sort the blocks vertically from top to bottom
distances = np.array([b2.coordinates[1] - b1.coordinates[3] for (b1, b2) in␣
↪zip(blocks, blocks[1:])])
    # Calculate the distances:
    # y coord for the upper edge of the bottom block -
    #   y coord for the bottom edge of the upper block
    # And convert to np array for easier post processing
plt.hist(distances, bins=50);
plt.axvline(x=3, color='r');
    # Let's have some visualization
```



According to the distance distribution plot, as well as the OCR results visualization, we can conclude:

- For the negative distances, it's because there are texts in the same line, e.g., "Los Angeles"
- For the small distances (indicated by the red line in the figure), they are texts in the same table row as the previous one
- For larger distances, they are generated from texts pairs of different rows

```
distance_th = 0

distances = np.append([0], distances) # Append a placeholder for the first word
block_group = (distances>distance_th).cumsum() # Create a block_group based on the␣
↪distance threshold

block_group
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  6,  7,  7,  8,  9,  9, 10, 11, 11, 12,
       13])
```

```
# Group the blocks by the block_group mask
grouped_blocks = [[] for i in range(max(block_group)+1)]
for i, block in zip(block_group, blocks):
    grouped_blocks[i].append(block)
```

Finally let's create a function for them

```
def group_blocks_by_distance(blocks, distance_th):

    blocks = sorted(blocks, key = lambda x: x.coordinates[1])
    distances = np.array([b2.coordinates[1] - b1.coordinates[3] for (b1, b2) in␣
↪zip(blocks, blocks[1:])])

    distances = np.append([0], distances)
    block_group = (distances>distance_th).cumsum()
```

(continues on next page)

(continued from previous page)

```
    grouped_blocks = [lp.Layout([]) for i in range(max(block_group)+1)]
    for i, block in zip(block_group, blocks):
        grouped_blocks[i].append(block)

    return grouped_blocks
```

```
A = group_blocks_by_distance(filtered_residence, 5)
B = group_blocks_by_distance(filter_lotno, 10)

# And finally we combine the outputs
height_th = 30
idxA, idxB = 0, 0

result = []
while idxA < len(A) and idxB < len(B):
    ay = A[idxA][0].coordinates[1]
    by = B[idxB][0].coordinates[1]
    ares, bres = ''.join(A[idxA].get_texts()), ''.join(B[idxB].get_texts())
    if abs(ay - by) < height_th:
        idxA += 1; idxB += 1
    elif ay < by:
        idxA += 1; bres = ''
    else:
        idxB += 1; ares = ''
    result.append([ares, bres])

result
```

```
[['LosAngeles', 'E6037'],
 ['AngelesLos', 'E6037'],
 ['LosAngeles', 'E6037'],
 ['Oakland', '?'],
 ['RiversideCoLosAngeles', 'E5928'],
 ['', 'E6037'],
 ['BeachLong', '?E6038?E597211'],
 ['BeachLong', ''],
 ['Maricopa', '?E5928'],
 ['FallsChurch', '8122-649334'],
 ['ChaseCity', '8122-64933?'],
 ['Houston', '7078-649343'],
 ['Scott', '7078-649342']]
```

As we can find, there are mistakes in the 5th and 6h row - `Riverside Co` and `LosAngeles` are wrongly combined. This is because the extra row `co` disrupted the row segmentation algorithm.

3.5 Save the results as a table

```
df = pd.DataFrame(row, columns=['residence', 'lot no'])
df
```

```
df.to_csv('./data/ocred-example-table.csv', index=None)
```

CHAPTER

FOUR

DEEP LAYOUT PARSING

In this tutorial, we will show how to use the `layoutparser` API to

1. Load Deep Learning Layout Detection models and predict the layout of the paper image
2. Use the coordinate system to parse the output

The `paper-image` is from https://arxiv.org/abs/2004.08686.

```
import layoutparser as lp
import cv2
```

4.1 Use Layout Models to detect complex layout

`layoutparser` can identify the layout of the given document with only 4 lines of code.

```
image = cv2.imread("data/paper-image.jpg")
image = image[..., ::-1]
    # Convert the image from BGR (cv2 default loading style)
    # to RGB
```

```
model = lp.Detectron2LayoutModel('lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config',
                                 extra_config=["MODEL.ROI_HEADS.SCORE_THRESH_TEST", 0.
↪8],
                                 label_map={0: "Text", 1: "Title", 2: "List", 3:"Table
↪", 4:"Figure"})
    # Load the deep layout model from the layoutparser API
    # For all the supported model, please check the Model
    # Zoo Page: https://layout-parser.readthedocs.io/en/latest/notes/modelzoo.html
```

```
layout = model.detect(image)
    # Detect the layout of the input image
```

```
lp.draw_box(image, layout, box_width=3)
    # Show the detected layout of the input image
```

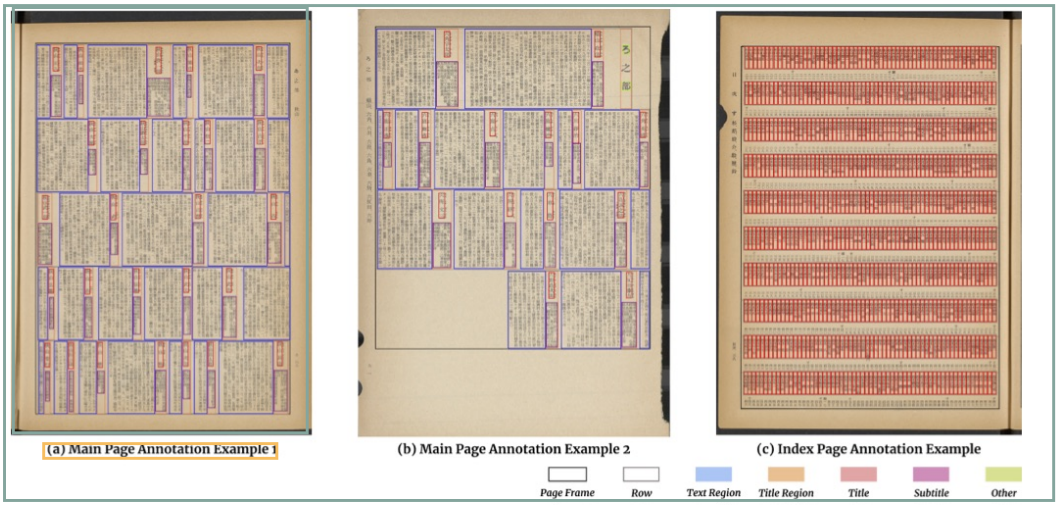


Figure 7: **Annotation Examples in HJDataset.** (a) and (b) show two examples for the labeling of `main` pages. The boxes are colored differently to reflect the layout element categories. Illustrated in (c), the items in each `index` page row are categorized as title blocks, and the annotations are denser.

tion over union (IOU) level [0.50:0.95][2], on the test data. In general, the high mAP values indicate accurate detection of the layout elements. The Faster R-CNN and Mask R-CNN achieve comparable results, better than RetinaNet. Noticeably, the detections for small blocks like title are less precise, and the accuracy drops sharply for the title category. In Figure 8, (a) and (b) illustrate the accurate prediction results of the Faster R-CNN model.

5.2. Pre-training for other datasets

We also examine how our dataset can help with a real-world document digitization application. When digitizing new publications, researchers usually do not generate large scale ground truth data to train their layout analysis models. If they are able to adapt our dataset, or models trained on our dataset, to develop models on their data, they can build their pipelines more efficiently and develop more accurate models. To this end, we conduct two experiments. First we examine how layout analysis models trained on the `main` pages can be used for understanding `index` pages. Moreover, we study how the pre-trained models perform on other historical Japanese documents.

Table 4 compares the performance of five Faster R-CNN models that are trained differently on `index` pages. If the model loads pre-trained weights from HJDataset, it includes information learned from `main` pages. Models trained over all the training data can be viewed as the benchmarks, while training with few samples (five in this case) are considered to mimic real-world scenarios. Given different training data, models pre-trained on HJDataset perform significantly better than those initialized with COCO weights. Intuitively, models trained on more data perform better than those with fewer samples. We also directly use the model trained on `main` to predict `index` pages without fine-tuning. The low zero-shot prediction accuracy indicates the dissimilarity between `index` and `main` pages. The large increase in mAP from 0.344 to 0.471 after the model is

[2] This is a core metric developed for the COCO competition [12] for evaluating the object detection quality.

Table 3: Detection mAP @ IOU [0.50:0.95] of different models for each category on the test set. All values are given as percentages.

Category	Faster R-CNN	Mask R-CNN[a]	RetinaNet
Page Frame	99.046	99.097	99.038
Row	98.831	98.482	95.067
Title Region	87.571	89.483	69.593
Text Region	94.463	86.798	89.531
Title	65.908	71.517	72.566
Subtitle	84.093	84.174	85.865
Other	44.023	39.849	14.371
mAP	81.991	81.343	75.223

[a] For training Mask R-CNN, the segmentation masks are the quadrilateral regions for each block. Compared to the rectangular bounding boxes, they delineate the text region more accurately.

4.2 Check the results from the model

```
type(layout)
```

```
layoutparser.elements.Layout
```

The `layout` variables is a `Layout` instance, which is inherited from list and supports handy methods for layout processing.

```
layout[0]
```

```
TextBlock(block=Rectangle(x_1=646.4182739257812, y_1=1420.1715087890625, x_2=1132.
↪8687744140625, y_2=1479.7222900390625), text=, id=None, type=Text, parent=None,␣
↪next=None, score=0.9996440410614014)
```

`layout` contains a series of `TextBlocks`. They store the coordinates in the `.block` variable and other information of the blocks like block type in `.type`, text in `.text`, etc. More information can be found at the documentation.

4.3 Use the coordinate system to process the detected layout

Firstly we filter text region of specific type:

```
text_blocks = lp.Layout([b for b in layout if b.type=='Text'])
figure_blocks = lp.Layout([b for b in layout if b.type=='Figure'])
```

As there could be text region detected inside the figure region, we just drop them:

```
text_blocks = lp.Layout([b for b in text_blocks \
                  if not any(b.is_in(b_fig) for b_fig in figure_blocks)])
```

Finally sort the text regions and assign ids:

```
h, w = image.shape[:2]

left_interval = lp.Interval(0, w/2*1.05, axis='x').put_on_canvas(image)

left_blocks = text_blocks.filter_by(left_interval, center=True)
left_blocks.sort(key = lambda b:b.coordinates[1], inplace=True)

right_blocks = [b for b in text_blocks if b not in left_blocks]
right_blocks.sort(key = lambda b:b.coordinates[1], inplace=True)

# And finally combine the two list and add the index
# according to the order
text_blocks = lp.Layout([b.set(id = idx) for idx, b in enumerate(left_blocks + right_
↪blocks)])
```

Visualize the cleaned text blocks:

```
lp.draw_box(image, text_blocks,
            box_width=3,
            show_element_id=True)
```

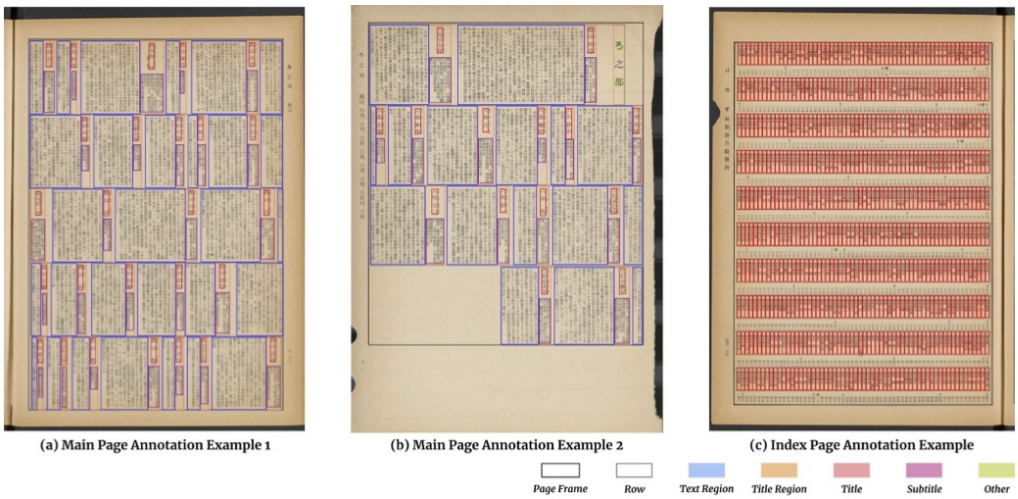


Figure 7: **Annotation Examples in HJDataset.** (a) and (b) show two examples for the labeling of `main` pages. The boxes are colored differently to reflect the layout element categories. Illustrated in (c), the items in each `index` page row are categorized as title blocks, and the annotations are denser.

lion over union (IOU) level [0.50:0.95][2], on the test data. In general, the high mAP values indicate accurate detection of the layout elements. The Faster R-CNN and Mask R-CNN achieve comparable results, better than RetinaNet. Noticeably, the detections for small blocks like title are less precise, and the accuracy drops sharply for the title category. In Figure 8, (a) and (b) illustrate the accurate prediction results of the Faster R-CNN model.

5.2. Pre-training for other datasets

We also examine how our dataset can help with a real-world document digitization application. When digitizing new publications, researchers usually do not generate large scale ground truth data to train their layout analysis models. If they are able to adapt our dataset, or models trained on our dataset, to develop models on their data, they can build their pipelines more efficiently and develop more accurate models. To this end, we conduct two experiments. First we examine how layout analysis models trained on the `main` pages can be used for understanding `index` pages. Moreover, we study how the pre-trained models perform on other historical Japanese documents.

Table 4 compares the performance of five Faster R-CNN models that are trained differently on `index` pages. If the model loads pre-trained weights from HJDataset, it includes information learned from `main` pages. Models trained over all the training data can be viewed as the benchmarks, while training with few samples (five in this case) are considered to mimic real-world scenarios. Given different training data, models pre-trained on HJDataset perform significantly better than those initialized with COCO weights. Intuitively, models trained on more data perform better than those with fewer samples. We also directly use the model trained on `main` to predict `index` pages without fine-tuning. The low zero-shot prediction accuracy indicates the dissimilarity between `index` and `main` pages. The large increase in mAP from 0.344 to 0.471 after the model is

Table 3: Detection mAP @ IOU [0.50:0.95] of different models for each category on the test set. All values are given as percentages.

Category	Faster R-CNN	Mask R-CNN[a]	RetinaNet
Page Frame	99.046	99.097	99.038
Row	98.831	98.482	95.067
Title Region	87.571	89.483	69.593
Text Region	94.463	86.798	89.531
Title	65.908	71.517	72.566
Subtitle	84.093	84.174	85.865
Other	44.023	39.849	14.371
mAP	81.991	81.343	75.223

[a] For training Mask R-CNN, the segmentation masks are the quadrilateral regions for each block. Compared to the rectangular bounding boxes, they delineate the text region more accurately.

[2] This is a core metric developed for the COCO competition [12] for evaluating the object detection quality.

4.4 Fetch the text inside each text region

We can also combine with the OCR functionality in `layoutparser` to fetch the text in the document.

```
ocr_agent = lp.TesseractAgent(languages='eng')
    # Initialize the tesseract ocr engine. You might need
    # to install the OCR components in layoutparser:
    # pip install layoutparser[ocr]
```

```
for block in text_blocks:
    segment_image = (block
                        .pad(left=5, right=5, top=5, bottom=5)
                        .crop_image(image))
        # add padding in each image segment can help
        # improve robustness

    text = ocr_agent.detect(segment_image)
    block.set(text=text, inplace=True)
```

```
for txt in text_blocks.get_texts():
    print(txt, end='\n---\n')
```

```
Figure 7: Annotation Examples in HJDataset. (a) and (b) show two examples for the␣
↪labeling of main pages. The boxes
are colored differently to reflect the layout element categories. Illustrated in (c),␣
↪the items in each index page row are
categorized as title blocks, and the annotations are denser.
---
tion over union (IOU) level [0.50:0.95]', on the test data. In
general, the high mAP values indicate accurate detection of
the layout elements. The Faster R-CNN and Mask R-CNN
achieve comparable results, better than RetinaNet. Notice-
ably, the detections for small blocks like title are less pre-
cise, and the accuracy drops sharply for the title category. In
Figure 8, (a) and (b) illustrate the accurate prediction results
of the Faster R-CNN model.
---
We also examine how our dataset can help with
world document digitization application. When digitizing
new publications, researchers usually do not generate large
scale ground truth data to train their layout analysis models.
If they are able to adapt our dataset, or models trained on
our dataset, to develop models on their data, they can build
their pipelines more efficiently and develop more accurate
models. To this end, we conduct two experiments. First we
examine how layout analysis models trained on the main
pages can be used for understanding index pages. More-
over, we study how the pre-trained models perform on other
historical Japanese documents.
---
Table 4 compares the performance of five Faster R-CNN
models that are trained differently on index pages. If the
model loads pre-trained weights from HJDataset, it includes
information learned from main pages. Models trained over
---
?This is a core metric developed for the COCO competition [| 2] for
```

(continues on next page)

(continued from previous page)

```
evaluating the object detection quality.
---
all the training data can be viewed as the benchmarks, while
training with few samples (five in this case) are consid-
ered to mimic real-world scenarios. Given different train-
ing data, models pre-trained on HJDataset perform signifi-
cantly better than those initialized with COCO weights. In-
tuitively, models trained on more data perform better than
those with fewer samples. We also directly use the model
trained on main to predict index pages without fine-
tuning. The low zero-shot prediction accuracy indicates the
dissimilarity between index and main pages. The large
increase in mAP from 0.344 to 0.471 after the model is
---
Table 3: Detection mAP @ IOU [0.50:0.95] of different
models for each category on the test set. All values are given
as percentages.
---
* For training Mask R-CNN, the segmentation masks are the quadri-
lateral regions for each block. Compared to the rectangular bounding
boxes, they delineate the text region more accurately.
---
```

CHAPTER

FIVE

LOAD COCO LAYOUT ANNOTATIONS

5.1 Preparation

In this notebook, I will illustrate how to use LayoutParser to load and visualize the layout annotation in the COCO format.

Before starting, please remember to download PubLayNet annotations and images from their website (let's just use the validation set for now as the training set is very large). And let's put all extracted files in the `data/publaynet/annotations` and `data/publaynet/val` folder.

And we need to install an additional library for conveniently handling the COCO data format:

```
pip install pycocotools
```

OK - Let's get on the code:

5.2 Loading and visualizing layouts using Layout-Parser

```
from pycocotools.coco import COCO
import layoutparser as lp
import random
import cv2
```

```
def load_coco_annotations(annotations, coco=None):
    """
    Args:
        annotations (List):
            a list of coco annotaions for the current image
        coco (`optional`, defaults to `False`):
            COCO annotation object instance. If set, this function will
            convert the loaded annotation category ids to category names
            set in COCO.categories
    """
    layout = lp.Layout()

    for ele in annotations:

        x, y, w, h = ele['bbox']

        layout.append(
            lp.TextBlock(
```

(continues on next page)

(continued from previous page)

```
                block = lp.Rectangle(x, y, w+x, h+y),
                type  = ele['category_id'] if coco is None else coco.cats[ele[
↪'category_id']]['name'],
                id = ele['id']
            )
        )

    return layout
```

The `load_coco_annotations` function will help convert COCO annotations into the layoutparser objects.

```
COCO_ANNO_PATH = 'data/publaynet/annotations/val.json'
COCO_IMG_PATH  = 'data/publaynet/val'

coco = COCO(COCO_ANNO_PATH)
```

```
loading annotations into memory...
Done (t=1.17s)
creating index...
index created!
```

```
color_map = {
    'text':   'red',
    'title':  'blue',
    'list':   'green',
    'table':  'purple',
    'figure': 'pink',
}


for image_id in random.sample(coco.imgs.keys(), 1):
    image_info = coco.imgs[image_id]
    annotations = coco.loadAnns(coco.getAnnIds([image_id]))

    image = cv2.imread(f'{COCO_IMG_PATH}/{image_info["file_name"]}')
    layout = load_coco_annotations(annotations, coco)

    viz = lp.draw_box(image, layout, color_map=color_map)
    display(viz) # show the results
```

Alzahawi et al. Progress in Orthodontics 2014, 15:35
http://www.progressinorthodontics.com/content/15/1/35

Table 3 Root resorption in millimeters and percentage of initial root length for lateral and central incisors in maxilla and mandible

	Mean	SD	Mean	SD	95% CI	P	Mean	SD	Mean	SD	95% CI	P
Pairs	12		11				22		21			
	(N = 37)						(N = 37)					
mm	0.97	1.00	0.74	0.87	−0.13 to 0.60	0.20	0.71	0.57	0.62	0.94	−0.23 to 0.42	0.5
%	5.79	5.60	4.30	5.02	−0.54 to 3.52	0.15	4.36	3.45	3.42	5.57	−1.04 to 2.92	0.34
Pairs	42		41				32		31			
	(N = 36)						(N = 30)					
mm	1.17	0.74	0.73	0.72	0.21 to 0.68	<0.01	1.04	0.82	0.79	0.65	−0.08 to 0.58	0.14
%	6.65	4.02	4.54	4.22	0.75 to 3.47	<0.01	6.14	4.78	5.07	4.02	−0.88 to 3.01	0.27

Discussion

Only a limited number of teeth from each incisor planned to be examined were finally analyzed due to inappropriate radiographs. That may have a negative effect on the power of the study, since statistical analysis was performed for each tooth separately. This drawback could have been eliminated by providing additional training to the dental assistants for the specific requirements of the particular X-rays, mainly related to the full appearance of the incisal edge of the teeth.

The two groups did not differ considering the characteristics of malocclusion recorded, except for invagination for the left incisor teeth. Mild type of invagination, however, has been shown not to be a risk factor for orthodontic root resorption [11]. The severity of crowding has not been considered at this study since crowding determines to a large extent the amount of tooth movement during leveling, which could have influenced the results.

Negative values for root resorption indicating root length increase have also previously been reported and attributed either to a real increase in root length [4,12] or to method error [13]. Error in registration of the CEJ, which is difficult to be identified, should also be considered.

The present study was performed in a sample of Norwegian orthodontic patients who were treated with straight-wire edgewise technique and conventional brackets. It could be assumed that the results from this study could be applicable for any patient treated with the same technique.

The end of leveling phase has been chosen as the time point for the evaluation of the effect of the two different arch wire materials, due to the possibility to use one type of arch wire material only, before the continuation of treatment that would require both types of arch wires. Additionally, the amount of root shortening 6 to 12 months after bracket placement is of high predictive value for the severity of root resorption after the completion of treatment [2]. Incisors were selected for this study because they are the most prone to tooth resorption, while a small reduction of their length is easier to be detected by conventional radiography. Most of the patients included in this study (90% of the examined teeth) exhibited signs of root resorption after initial treatment of 6 to 10 months. For the majority of the cases, the amount of resorption was minor, but present. According to Makedonas et al. [14], root resorption diagnosed by cone beam computed tomography after 6 months of treatment was clinically significant only for the 4% of the patients.

Leveling was accomplished sooner by super-elastic than stainless steel arch wires in our sample. Similarly, Weiland [15] has shown greater amount of buccal premolar

Table 4 Regression analysis for teeth 12 and 22

	Tooth 12				Tooth 22			
	Constant	Crossbite 12	Age	R^2 (adjusted) %	Constant	Crossbite 22	ANB	R^2 (adjusted) %
1	0.783	**0.967**		13.9	0.675	**0.600**		11.3
		0.012				*0.019*		
2	−1.149	**1.052**	**0.143**	17.6	0.975	**0.492**	**−0.091**	16.8
		0.006	*0.116*			*0.052*	*0.060*	
95% CI		0.32 to 1.79	−0.04 to 0.32			0.00 to 0.99	−0.19 to 0.01	

The figures in the first column denote the number of explanatory variables included in the regression analysis by first applying the best subset regression analysis among all variables. For each regression analysis, the partial regression coefficient of the variable is given in bold and the *P* value of the partial regression coefficient in italics.

You could add more information in the visualization.

```
lp.draw_box(image,
               [b.set(id=f'{b.id}/{b.type}') for b in layout],
               color_map=color_map,
               show_element_id=True, id_font_size=10,
               id_text_background_color='grey',
               id_text_color='white')
```

Alzahawi et al. Progress in Orthodontics 2014, 15:35
http://www.progressinorthodontics.com/content/15/1/35

3759767/text esorption in millimeters and percentage of initial root length for lateral and central incisors in maxilla and mandible

3759770/table

	SD	Mean	SD	95% CI	P	Mean	SD	Mean	SD	95% CI	P
Pairs	12		11			22		21			
	(N = 37)					(N = 37)					
mm	0.97	1.00	0.74	0.87	−0.13 to 0.60	0.20	0.71	0.57	0.62	0.94	−0.23 to 0.42	0.55
%	5.79	5.60	4.30	5.02	−0.54 to 3.52	0.15	4.36	3.45	3.42	5.57	−1.04 to 2.92	0.34
Pairs	42		41			32		31			
	(N = 36)					(N = 30)					
mm	1.17	0.74	0.73	0.72	0.21 to 0.68	<0.01	1.04	0.82	0.79	0.65	−0.08 to 0.58	0.14
%	6.65	4.02	4.54	4.22	0.75 to 3.47	<0.01	6.14	4.78	5.07	4.02	−0.88 to 3.01	0.27

3759772/title

3759760/text number of teeth from each incisor planned to be examined were finally analyzed due to inappropriate radiographs. That may have a negative effect on the power of the study, since statistical analysis was performed for each tooth separately. This drawback could have been eliminated by providing additional training to the dental assistants for the specific requirements of the particular X-rays, mainly related to the full appearance of the incisal edge of the teeth.

3759761/text oups did not differ considering the characteristics of malocclusion recorded, except for invagination for the left incisor teeth. Mild type of invagination, however, has been shown not to be a risk factor for orthodontic root resorption [11]. The severity of crowding has not been considered at this study since crowding determines to a large extent the amount of tooth movement during leveling, which could have influenced the results.

3759762/text lues for root resorption indicating root length increase have also previously been reported and attributed either to a real increase in root length [4,12] or to method error [13]. Error in registration of the CEJ, which is difficult to be identified, should also be considered.

3759764/text t study was performed in a sample of Norwegian orthodontic patients who were treated with

3759763/text dgewise technique and conventional brackets. It could be assumed that the results from this study could be applicable for any patient treated with the same technique.

3759765/text leveling phase has been chosen as the time point for the evaluation of the effect of the two different arch wire materials, due to the possibility to use one type of arch wire material only, before the continuation of treatment that would require both types of arch wires. Additionally, the amount of root shortening 6 to 12 months after bracket placement is of high predictive value for the severity of root resorption after the completion of treatment [2]. Incisors were selected for this study because they are the most prone to tooth resorption, while a small reduction of their length is easier to be detected by conventional radiography. Most of the patients included in this study (90% of the examined teeth) exhibited signs of root resorption after initial treatment of 6 to 10 months. For the majority of the cases, the amount of resorption was minor, but present. According to Makedonas et al. [14], root resorption diagnosed by cone beam computed tomography after 6 months of treatment was clinically significant only for the 4% of the patients.

3759766/text accomplished sooner by super-elastic than stainless steel arch wires in our sample. Similarly, Weiland [15] has shown greater amount of buccal premolar

3759768/text sion analysis for teeth 12 and 22

3759771/table

	Tooth 12				Tooth 22			
	Constant	Crossbite 12	Age	R^2 (adjusted) %	Constant	Crossbite 22	ANB	R^2 (adjusted) %
1	0.783	**0.967**		13.9	0.675	**0.600**		11.3
		0.012				*0.019*		
2	−1.149	**1.052**	**0.143**	17.6	0.975	**0.492**	**−0.091**	16.8
		0.006	*0.116*			*0.052*	*0.060*	
95% CI		0.32 to 1.79	−0.04 to 0.32			0.00 to 0.99	−0.19 to 0.01	

3759769/text rst column denote the number of explanatory variables included in the regression analysis by first applying the best subset regression analysis. For each regression analysis, the partial regression coefficient of the variable is given in bold and the *P* value of the partial regression coefficient in italics.

5.3 Model Predictions on loaded data

We could also check how the trained layout model performs on the input image. Following this instruction, we could conveniently load a layout prediction model and run predictions on the existing image.

```
model = lp.Detectron2LayoutModel('lp://PubLayNet/faster_rcnn_R_50_FPN_3x/config',
                                 extra_config=["MODEL.ROI_HEADS.SCORE_THRESH_TEST", 0.
↪8],
                                 label_map={0: "text", 1: "title", 2: "list", 3:"table
↪", 4:"figure"})
```

```
layout_predicted = model.detect(image)
```

```
lp.draw_box(image,
            [b.set(id=f'{b.type}/{b.score:.2f}') for b in layout_predicted],
            color_map=color_map,
            show_element_id=True, id_font_size=10,
            id_text_background_color='grey',
            id_text_color='white')
```

text/0.99 ot resorption in millimeters and percentage of initial root length for lateral and central incisors in maxilla and mandible

table/0.98

	Mean	SD	Mean	SD	95% CI	P	Mean	SD	Mean	SD	95% CI	P
Pairs	12		11				22		21			
	(N = 37)						(N = 37)					
mm	0.97	1.00	0.74	0.87	−0.13 to 0.60	0.20	0.71	0.57	0.62	0.94	−0.23 to 0.42	0.55
%	5.79	5.60	4.30	5.02	−0.54 to 3.52	0.15	4.36	3.45	3.42	5.57	−1.04 to 2.92	0.34
Pairs	42		41				32		31			
	(N = 36)						(N = 30)					
mm	1.17	0.74	0.73	0.72	0.21 to 0.68	<0.01	1.04	0.82	0.79	0.65	−0.08 to 0.58	0.14
%	6.65	4.02	4.54	4.22	0.75 to 3.47	<0.01	6.14	4.78	5.07	4.02	−0.88 to 3.01	0.27

title/0.99 n

text/1.00 ited number of teeth from each incisor planned o be examined were finally analyzed due to inappropriate adiographs. That may have a negative effect on the power of the study, since statistical analysis was performed for each tooth separately. This drawback could have been eliminated by providing additional training to the dental assistants for the specific requirements of the particular X-rays, mainly related to the full appearance of the incisal edge of the teeth.

text/1.00 groups did not differ considering the characteristics of malocclusion recorded, except for invagination for the left incisor teeth. Mild type of invagination, however, has been shown not to be a risk factor for orthodontic root resorption [11]. The severity of crowding has not been considered at this study since crowding determines to a large extent the amount of tooth movement during leveling, which could have influenced the results.

text/1.00 values for root resorption indicating root length increase have also previously been reported and attributed either to a real increase in root length [4,12] or to method error [13]. Error in registration of the CEJ, which is difficult to be identified, should also be considered.

text/1.00 esent study was performed in a sample of Norwegian orthodontic patients who were treated with

text/0.99 ire edgewise technique and conventional brackets. It could be assumed that the results from this study could be applicable for any patient treated with the same technique.

text/1.00 of leveling phase has been chosen as the time point for the evaluation of the effect of the two different arch wire materials, due to the possibility to use one type of arch wire material only, before the continuation of treatment that would require both types of arch wires. Additionally, the amount of root shortening 6 to 12 months after bracket placement is of high predictive value for the severity of root resorption after the completion of treatment [2]. Incisors were selected for this study because they are the most prone to tooth resorption, while a small reduction of their length is easier to be detected by conventional radiography. Most of the patients included in this study (90% of the examined teeth) exhibited signs of root resorption after initial treatment of 6 to 10 months. For the majority of the cases, the amount of resorption was minor, but present. According to Makedonas et al. [14], root resorption diagnosed by cone beam computed tomography after 6 months of treatment was clinically significant only for the 4% of the patients.

text/0.99 s was accomplished sooner by super-elastic than stainless steel arch wires in our sample. Similarly, Weiland [15] has shown greater amount of buccal premolar

text/1.00 egression analysis for teeth 12 and 22

table/0.99

	Tooth 12				Tooth 22			
	Constant	Crossbite 12	Age	R^2 (adjusted) %	Constant	Crossbite 22	ANB	R^2 (adjusted) %
1	0.783	**0.967**		13.9	0.675	**0.600**		11.3
		0.012				*0.019*		
2	−1.149	**1.052**	**0.143**	17.6	0.975	**0.492**	**−0.091**	16.8
		0.006	*0.116*			*0.052*	*0.060*	
95% CI		0.32 to 1.79	−0.04 to 0.32			0.00 to 0.99	−0.19 to 0.01	

text/1.00 the first column denote the number of explanatory variables included in the regression analysis by first applying the best subset regression analysis variables. For each regression analysis, the partial regression coefficient of the variable is given in bold and the *P* value of the partial regression coefficient in italics.

CHAPTER

SIX

LAYOUT ELEMENTS

6.1 Coordinate System

class layoutparser.elements.**Interval**(*start*, *end*, *axis*, *canvas_height=None*, *canvas_width=None*)

Bases: layoutparser.elements.base.BaseCoordElement

This class describes the coordinate system of an interval, a block defined by a pair of start and end point on the designated axis and same length as the base canvas on the other axis.

Parameters

- **start** (numeric) – The coordinate of the start point on the designated axis.
- **end** (numeric) – The end coordinate on the same axis as start.
- **axis** (str) – The designated axis that the end points belong to.
- **canvas_height** (numeric, *optional*, defaults to 0) – The height of the canvas that the interval is on.
- **canvas_width** (numeric, *optional*, defaults to 0) – The width of the canvas that the interval is on.

property height

Calculate the height of the interval. If the interval is along the x-axis, the height will be the height of the canvas, otherwise, it will be the difference between the start and end point.

Returns Output the numeric value of the height.

Return type numeric

property width

Calculate the width of the interval. If the interval is along the y-axis, the width will be the width of the canvas, otherwise, it will be the difference between the start and end point.

Returns Output the numeric value of the width.

Return type numeric

property coordinates

This method considers an interval as a rectangle and calculates the coordinates of the upper left and lower right corners to define the interval.

Returns Output the numeric values of the coordinates in a Tuple of size four.

Return type Tuple(numeric)

property points

Return the coordinates of all four corners of the interval in a clockwise fashion starting from the upper left.

Returns A Numpy array of shape 4x2 containing the coordinates.

Return type `Numpy array`

property center
Calculate the mid-point between the start and end point.

Returns Returns of coordinate of the center.

Return type `Tuple(numeric)`

property area
Return the area of the covered region of the interval. The area is bounded to the canvas. If the interval is put on a canvas, the area equals to interval width * canvas height (axis='x') or interval height * canvas width (axis='y'). Otherwise, the area is zero.

put_on_canvas(*canvas*)
Set the height and the width of the canvas that the interval is on.

Parameters **canvas** (`Numpy array` or `BaseCoordElement` or `PIL.Image.Image`) – The base element that the interval is on. The numpy array should be the format of *[height, width]*.

Returns A copy of the current Interval with its canvas height and width set to those of the input canvas.

Return type *Interval*

condition_on(*other*)
Given the current element in relative coordinates to another element which is in absolute coordinates, generate a new element of the current element in absolute coordinates.

Parameters **other** (`BaseCoordElement`) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the absolute coordinate system.

Return type `BaseCoordElement`

relative_to(*other*)
Given the current element and another element both in absolute coordinates, generate a new element of the current element in relative coordinates to the other element.

Parameters **other** (`BaseCoordElement`) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the relative coordinate system.

Return type `BaseCoordElement`

is_in(*other*, *soft_margin={}*, *center=False*)
Identify whether the current element is within another element.

Parameters

- **other** (`BaseCoordElement`) – The other layout element involved in the geometric operations.
- **soft_margin** (`dict`, *optional*, defaults to *{}*) – Enlarge the other element with wider margins to relax the restrictions.

- **center** (bool, *optional*, defaults to *False*) – The toggle to determine whether the center (instead of the four corners) of the current element is in the other element.

Returns Returns *True* if the current element is in the other element and *False* if not.

Return type bool

intersect(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Intersect the current shape with the other object, with operations defined in *Shape Operations*.

union(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Union the current shape with the other object, with operations defined in *Shape Operations*.

pad(*left=0*, *right=0*, *top=0*, *bottom=0*, *safe_mode=True*)
Pad the layout element on the four sides of the polygon with the user-defined pixels. If safe_mode is set to True, the function will cut off the excess padding that falls on the negative side of the coordinates.

Parameters

- **left** (int, *optional*, defaults to 0) – The number of pixels to pad on the upper side of the polygon.
- **right** (int, *optional*, defaults to 0) – The number of pixels to pad on the lower side of the polygon.
- **top** (int, *optional*, defaults to 0) – The number of pixels to pad on the left side of the polygon.
- **bottom** (int, *optional*, defaults to 0) – The number of pixels to pad on the right side of the polygon.
- **safe_mode** (bool, *optional*, defaults to True) – A bool value to toggle the safe_mode.

Returns The padded BaseCoordElement object.

Return type BaseCoordElement

shift(*shift_distance*)
Shift the interval by a user specified amount along the same axis that the interval is defined on.

Parameters **shift_distance** (numeric) – The number of pixels used to shift the interval.

Returns The shifted Interval object.

Return type BaseCoordElement

scale(*scale_factor*)
Scale the layout element by a user specified amount the same axis that the interval is defined on.

Parameters **scale_factor** (numeric) – The amount for downscaling or upscaling the element.

Returns The scaled Interval object.

Return type BaseCoordElement

crop_image(*image*)
Crop the input image according to the coordinates of the element.

Parameters **image** (Numpy array) – The array of the input image.

Returns The array of the cropped image.

Return type Numpy array

to_rectangle()
Convert the Interval to a Rectangle element.

Returns The converted Rectangle object.

Return type *Rectangle*

to_quadrilateral()
Convert the Interval to a Quadrilateral element.

Returns The converted Quadrilateral object.

Return type *Quadrilateral*

class layoutparser.elements.**Rectangle**(*x_1*, *y_1*, *x_2*, *y_2*)
Bases: layoutparser.elements.base.BaseCoordElement

This class describes the coordinate system of an axial rectangle box using two points as indicated below:

```
(x_1, y_1) ----
|              |
|              |
|              |
---- (x_2, y_2)
```

Parameters

- **x_1** (numeric) – x coordinate on the horizontal axis of the upper left corner of the rectangle.
- **y_1** (numeric) – y coordinate on the vertical axis of the upper left corner of the rectangle.
- **x_2** (numeric) – x coordinate on the horizontal axis of the lower right corner of the rectangle.
- **y_2** (numeric) – y coordinate on the vertical axis of the lower right corner of the rectangle.

property height
Calculate the height of the rectangle.

Returns Output the numeric value of the height.

Return type numeric

property width
Calculate the width of the rectangle.

Returns Output the numeric value of the width.

Return type numeric

property coordinates
Return the coordinates of the two points that define the rectangle.

Returns Output the numeric values of the coordinates in a Tuple of size four.

Return type Tuple(numeric)

property points
Return the coordinates of all four corners of the rectangle in a clockwise fashion starting from the upper left.

Returns A Numpy array of shape 4x2 containing the coordinates.

Return type Numpy array

property center
Calculate the center of the rectangle.

Returns Returns of coordinate of the center.

Return type `Tuple(numeric)`

property area
Return the area of the rectangle.

condition_on(*other*)
Given the current element in relative coordinates to another element which is in absolute coordinates, generate a new element of the current element in absolute coordinates.

Parameters **other** (`BaseCoordElement`) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the absolute coordinate system.

Return type `BaseCoordElement`

relative_to(*other*)
Given the current element and another element both in absolute coordinates, generate a new element of the current element in relative coordinates to the other element.

Parameters **other** (`BaseCoordElement`) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the relative coordinate system.

Return type `BaseCoordElement`

is_in(*other*, *soft_margin={}*, *center=False*)
Identify whether the current element is within another element.

Parameters

- **other** (`BaseCoordElement`) – The other layout element involved in the geometric operations.
- **soft_margin** (`dict`, *optional*, defaults to *{}*) – Enlarge the other element with wider margins to relax the restrictions.
- **center** (`bool`, *optional*, defaults to *False*) – The toggle to determine whether the center (instead of the four corners) of the current element is in the other element.

Returns Returns *True* if the current element is in the other element and *False* if not.

Return type `bool`

intersect(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Intersect the current shape with the other object, with operations defined in *Shape Operations*.

union(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Union the current shape with the other object, with operations defined in *Shape Operations*.

pad(*left=0*, *right=0*, *top=0*, *bottom=0*, *safe_mode=True*)
Pad the layout element on the four sides of the polygon with the user-defined pixels. If safe_mode is set to True, the function will cut off the excess padding that falls on the negative side of the coordinates.

Parameters

- **left** (`int`, *optional*, defaults to 0) – The number of pixels to pad on the upper side of the polygon.

- **right** (int, *optional*, defaults to 0) – The number of pixels to pad on the lower side of the polygon.
- **top** (int, *optional*, defaults to 0) – The number of pixels to pad on the left side of the polygon.
- **bottom** (int, *optional*, defaults to 0) – The number of pixels to pad on the right side of the polygon.
- **safe_mode** (bool, *optional*, defaults to True) – A bool value to toggle the safe_mode.

Returns The padded BaseCoordElement object.

Return type BaseCoordElement

shift(*shift_distance=0*)

Shift the layout element by user specified amounts on x and y axis respectively. If shift_distance is one numeric value, the element will by shifted by the same specified amount on both x and y axis.

Parameters **shift_distance** (numeric or Tuple(numeric) or List[numeric]) – The number of pixels used to shift the element.

Returns The shifted BaseCoordElement of the same shape-specific class.

Return type BaseCoordElement

scale(*scale_factor=1*)

Scale the layout element by a user specified amount on x and y axis respectively. If scale_factor is one numeric value, the element will by scaled by the same specified amount on both x and y axis.

Parameters **scale_factor** (numeric or Tuple(numeric) or List[numeric]) – The amount for downscaling or upscaling the element.

Returns The scaled BaseCoordElement of the same shape-specific class.

Return type BaseCoordElement

crop_image(*image*)

Crop the input image according to the coordinates of the element.

Parameters **image** (Numpy array) – The array of the input image.

Returns The array of the cropped image.

Return type Numpy array

to_interval(*axis*, ***kwargs*)

to_quadrilateral()

class layoutparser.elements.**Quadrilateral**(*points: Union[numpy.ndarray, List, List[List]]*, *height=None*, *width=None*)

Bases: layoutparser.elements.base.BaseCoordElement

This class describes the coodinate system of a four-sided polygon. A quadrilateral is defined by the coordinates of its 4 corners in a clockwise order starting with the upper left corner (as shown below):

```
points[0] -...- points[1]
|                       |
.                       .
.                       .
.                       .
|                       |
points[3] -...- points[2]
```

Parameters

- **points** (Numpy array or *list*) – A *np.ndarray* of shape 4x2 for four corner coordinates or a list of length 8 for in the format of *[p0_x, p0_y, p1_x, p1_y, p2_x, p2_y, p3_x, p3_y]* or a list of length 4 in the format of *[[p0_x, p0_y], [p1_x, p1_y], [p2_x, p2_y], [p3_x, p3_y]]*.
- **height** (numeric, *optional*, defaults to *None*) – The height of the quadrilateral. This is to better support the perspective transformation from the OpenCV library.
- **width** (numeric, *optional*, defaults to *None*) – The width of the quadrilateral. Similarly as height, this is to better support the perspective transformation from the OpenCV library.

property height

Return the user defined height, otherwise the height of its circumscribed rectangle.

Returns Output the numeric value of the height.

Return type numeric

property width

Return the user defined width, otherwise the width of its circumscribed rectangle.

Returns Output the numeric value of the width.

Return type numeric

property coordinates

Return the coordinates of the upper left and lower right corners points that define the circumscribed rectangle.

Returns Tuple(numeric): Output the numeric values of the coordinates in a Tuple of size four.

property points

Return the coordinates of all four corners of the quadrilateral in a clockwise fashion starting from the upper left.

Returns A Numpy array of shape 4x2 containing the coordinates.

Return type Numpy array

property center

Calculate the center of the quadrilateral.

Returns Returns of coordinate of the center.

Return type Tuple(numeric)

property area

Return the area of the quadrilateral.

property mapped_rectangle_points

property perspective_matrix

map_to_points_ordering(*x_map*, *y_map*)

condition_on(*other*)

Given the current element in relative coordinates to another element which is in absolute coordinates, generate a new element of the current element in absolute coordinates.

Parameters **other** (BaseCoordElement) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the absolute coordinate system.

Return type BaseCoordElement

relative_to(*other*)
Given the current element and another element both in absolute coordinates, generate a new element of the current element in relative coordinates to the other element.

Parameters **other** (BaseCoordElement) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the relative coordinate system.

Return type BaseCoordElement

is_in(*other*, *soft_margin={}*, *center=False*)
Identify whether the current element is within another element.

Parameters

- **other** (BaseCoordElement) – The other layout element involved in the geometric operations.
- **soft_margin** (dict, *optional*, defaults to *{}*) – Enlarge the other element with wider margins to relax the restrictions.
- **center** (bool, *optional*, defaults to *False*) – The toggle to determine whether the center (instead of the four corners) of the current element is in the other element.

Returns Returns *True* if the current element is in the other element and *False* if not.

Return type bool

intersect(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Intersect the current shape with the other object, with operations defined in *Shape Operations*.

union(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Union the current shape with the other object, with operations defined in *Shape Operations*.

pad(*left=0*, *right=0*, *top=0*, *bottom=0*, *safe_mode=True*)
Pad the layout element on the four sides of the polygon with the user-defined pixels. If safe_mode is set to True, the function will cut off the excess padding that falls on the negative side of the coordinates.

Parameters

- **left** (int, *optional*, defaults to 0) – The number of pixels to pad on the upper side of the polygon.
- **right** (int, *optional*, defaults to 0) – The number of pixels to pad on the lower side of the polygon.
- **top** (int, *optional*, defaults to 0) – The number of pixels to pad on the left side of the polygon.
- **bottom** (int, *optional*, defaults to 0) – The number of pixels to pad on the right side of the polygon.
- **safe_mode** (bool, *optional*, defaults to True) – A bool value to toggle the safe_mode.

Returns The padded BaseCoordElement object.

Return type BaseCoordElement

shift(*shift_distance=0*)
Shift the layout element by user specified amounts on x and y axis respectively. If shift_distance is one numeric value, the element will by shifted by the same specified amount on both x and y axis.

Parameters **shift_distance** (`numeric` or `Tuple(numeric)` or `List[numeric]`) – The number of pixels used to shift the element.

Returns The shifted BaseCoordElement of the same shape-specific class.

Return type `BaseCoordElement`

scale(*scale_factor=1*)
Scale the layout element by a user specified amount on x and y axis respectively. If scale_factor is one numeric value, the element will by scaled by the same specified amount on both x and y axis.

Parameters **scale_factor** (`numeric` or `Tuple(numeric)` or `List[numeric]`) – The amount for downscaling or upscaling the element.

Returns The scaled BaseCoordElement of the same shape-specific class.

Return type `BaseCoordElement`

crop_image(*image*)
Crop the input image using the points of the quadrilateral instance.

Parameters **image** (`Numpy array`) – The array of the input image.

Returns The array of the cropped image.

Return type `Numpy array`

to_interval(*axis*, ***kwargs*)

to_rectangle()

to_dict() → Dict[str, Any]
Generate a dictionary representation of the current object:

```
{
    "block_type": "quadrilateral",
    "points": [
        p[0,0], p[0,1],
        p[1,0], p[1,1],
        p[2,0], p[2,1],
        p[3,0], p[3,1]
    ],
    "height": value,
    "width": value
}
```

6.2 TextBlock

class layoutparser.elements.**TextBlock**(*block*, *text=None*, *id=None*, *type=None*, *parent=None*, *next=None*, *score=None*)
Bases: `layoutparser.elements.base.BaseLayoutElement`

This class constructs content-related information of a layout element in addition to its coordinate definitions (i.e. Interval, Rectangle or Quadrilateral).

Parameters

- **block** (BaseCoordElement) – The shape-specific coordinate systems that the text block belongs to.
- **text** (str, *optional*, defaults to None) – The ocr'ed text results within the boundaries of the text block.
- **id** (int, *optional*, defaults to *None*) – The id of the text block.
- **type** (int, *optional*, defaults to *None*) – The type of the text block.
- **parent** (int, *optional*, defaults to *None*) – The id of the parent object.
- **next** (int, *optional*, defaults to *None*) – The id of the next block.
- **score** (numeric, defaults to *None*) – The prediction confidence of the block

property height
Return the height of the shape-specific block.

Returns Output the numeric value of the height.

Return type numeric

property width
Return the width of the shape-specific block.

Returns Output the numeric value of the width.

Return type numeric

property coordinates
Return the coordinates of the two corner points that define the shape-specific block.

Returns Output the numeric values of the coordinates in a Tuple of size four.

Return type Tuple(numeric)

property points
Return the coordinates of all four corners of the shape-specific block in a clockwise fashion starting from the upper left.

Returns A Numpy array of shape 4x2 containing the coordinates.

Return type Numpy array

property area
Return the area of associated block.

condition_on(*other*)
Given the current element in relative coordinates to another element which is in absolute coordinates, generate a new element of the current element in absolute coordinates.

Parameters **other** (BaseCoordElement) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the absolute coordinate system.

Return type BaseCoordElement

relative_to(*other*)
Given the current element and another element both in absolute coordinates, generate a new element of the current element in relative coordinates to the other element.

Parameters **other** (BaseCoordElement) – The other layout element involved in the geometric operations.

Raises **Exception** – Raise error when the input type of the other element is invalid.

Returns The BaseCoordElement object of the original element in the relative coordinate system.

Return type BaseCoordElement

is_in(*other*, *soft_margin={}*, *center=False*)
Identify whether the current element is within another element.

Parameters

- **other** (BaseCoordElement) – The other layout element involved in the geometric operations.
- **soft_margin** (dict, *optional*, defaults to *{}*) – Enlarge the other element with wider margins to relax the restrictions.
- **center** (bool, *optional*, defaults to *False*) – The toggle to determine whether the center (instead of the four corners) of the current element is in the other element.

Returns Returns *True* if the current element is in the other element and *False* if not.

Return type bool

union(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Union the current shape with the other object, with operations defined in *Shape Operations*.

intersect(*other: layoutparser.elements.base.BaseCoordElement*, *strict: bool = True*)
Intersect the current shape with the other object, with operations defined in *Shape Operations*.

shift(*shift_distance*)
Shift the layout element by user specified amounts on x and y axis respectively. If shift_distance is one numeric value, the element will by shifted by the same specified amount on both x and y axis.

Parameters **shift_distance** (numeric or Tuple(numeric) or List[numeric]) – The number of pixels used to shift the element.

Returns The shifted BaseCoordElement of the same shape-specific class.

Return type BaseCoordElement

pad(*left=0*, *right=0*, *top=0*, *bottom=0*, *safe_mode=True*)
Pad the layout element on the four sides of the polygon with the user-defined pixels. If safe_mode is set to True, the function will cut off the excess padding that falls on the negative side of the coordinates.

Parameters

- **left** (int, *optional*, defaults to 0) – The number of pixels to pad on the upper side of the polygon.
- **right** (int, *optional*, defaults to 0) – The number of pixels to pad on the lower side of the polygon.
- **top** (int, *optional*, defaults to 0) – The number of pixels to pad on the left side of the polygon.
- **bottom** (int, *optional*, defaults to 0) – The number of pixels to pad on the right side of the polygon.
- **safe_mode** (bool, *optional*, defaults to True) – A bool value to toggle the safe_mode.

Returns The padded BaseCoordElement object.

Return type BaseCoordElement

scale(*scale_factor*)

Scale the layout element by a user specified amount on x and y axis respectively. If scale_factor is one numeric value, the element will by scaled by the same specified amount on both x and y axis.

Parameters **scale_factor** (numeric or Tuple(numeric) or List[numeric]) – The amount for downscaling or upscaling the element.

Returns The scaled BaseCoordElement of the same shape-specific class.

Return type BaseCoordElement

crop_image(*image*)

Crop the input image according to the coordinates of the element.

Parameters **image** (Numpy array) – The array of the input image.

Returns The array of the cropped image.

Return type Numpy array

to_interval(*axis: Optional[str] = None*, ***kwargs*)

to_rectangle()

to_quadrilateral()

to_dict() → Dict[str, Any]

Generate a dictionary representation of the current textblock of the format:

```
{
    "block_type": <name of self.block>,
    <attributes of self.block combined with
        non-empty self._features>
}
```

classmethod from_dict(*data: Dict[str, Any]*) → layoutparser.elements.layout_elements.TextBlock

Initialize the textblock based on the dictionary representation. It generate the block based on the *block_type* and *block_attr*, and loads the textblock specific features from the dict.

Parameters **data** (dict) – The dictionary representation of the object

6.3 Layout

class layoutparser.elements.**Layout**(*blocks: Optional[List] = None*, ***, *page_data: Dict = None*)

Bases: collections.abc.MutableSequence

The *Layout* class id designed for processing a list of layout elements on a page. It stores the layout elements in a list and the related *page_data*, and provides handy APIs for processing all the layout elements in batch. `

Parameters

- **blocks** (list) – A list of layout element blocks
- **page_data** (*Dict, optional*) – A dictionary storing the page (canvas) related information like *height*, *width*, etc. It should be passed in as a keyword argument to avoid any confusion. Defaults to None.

insert (*key*, *value*)
S.insert(index, value) – insert value before index

copy ()

relative_to (*other*)

condition_on (*other*)

is_in (*other*, *soft_margin={}*, *center=False*)

sort (*key=None*, *reverse=False*, *inplace=False*) → Optional[layoutparser.elements.layout.Layout]
Sort the list of blocks based on the given

Parameters

- **key** (*[type], optional*) – key specifies a function of one argument that
- **used to extract a comparison key from each list element.** (*is*) –
- **to None.** (*Defaults*) –
- **reverse** (*bool, optional*) – reverse is a boolean value. If set to True,
- **the list elements are sorted as if each comparison were reversed.** (*then*) –
- **to False.** (*Defaults*) –
- **inplace** (*bool, optional*) – whether to perform the sort inplace. If set
- **False, it will return another object instance with _block sorted in** (*to*) –
- **order. Defaults to False.** (*the*) –

Examples::

```
>>> import layoutparser as lp
>>> i = lp.Interval(4, 5, axis="y")
>>> l = lp.Layout([i, i.shift(2)])
>>> l.sort(key=lambda x: x.coordinates[1], reverse=True)
```

filter_by (*other*, *soft_margin={}*, *center=False*)
Return a *Layout* object containing the elements that are in the *other* object.

Parameters **other** (`BaseCoordElement`) – The block to filter the current elements.

Returns A new layout object after filtering.

Return type *Layout*

shift (*shift_distance*)
Shift all layout elements by user specified amounts on x and y axis respectively. If shift_distance is one numeric value, the element will by shifted by the same specified amount on both x and y axis.

Parameters **shift_distance** (`numeric` or `Tuple(numeric)` or `List[numeric]`) – The number of pixels used to shift the element.

Returns A new layout object with all the elements shifted in the specified values.

Return type *Layout*

pad(*left=0*, *right=0*, *top=0*, *bottom=0*, *safe_mode=True*)
Pad all layout elements on the four sides of the polygon with the user-defined pixels. If safe_mode is set to True, the function will cut off the excess padding that falls on the negative side of the coordinates.

Parameters

- **left** (int, *optional*, defaults to 0) – The number of pixels to pad on the upper side of the polygon.
- **right** (int, *optional*, defaults to 0) – The number of pixels to pad on the lower side of the polygon.
- **top** (int, *optional*, defaults to 0) – The number of pixels to pad on the left side of the polygon.
- **bottom** (int, *optional*, defaults to 0) – The number of pixels to pad on the right side of the polygon.
- **safe_mode** (bool, *optional*, defaults to True) – A bool value to toggle the safe_mode.

Returns A new layout object with all the elements padded in the specified values.

Return type *Layout*

scale(*scale_factor*)
Scale all layout element by a user specified amount on x and y axis respectively. If scale_factor is one numeric value, the element will by scaled by the same specified amount on both x and y axis.

Parameters **scale_factor** (numeric or Tuple(numeric) or List[numeric]) – The amount for downscaling or upscaling the element.

Returns A new layout object with all the elements scaled in the specified values.

Return type *Layout*

crop_image(*image*)

get_texts()
Iterate through all the text blocks in the list and append their ocr'ed text results.

Returns A list of text strings of the text blocks in the list of layout elements.

Return type List[str]

get_info(*attr_name*)
Given user-provided attribute name, check all the elements in the list and return the corresponding attribute values.

Parameters **attr_name** (str) – The text string of certain attribute name.

Returns The list of the corresponding attribute value (if exist) of each element in the list.

Return type List

to_dict() → Dict[str, Any]
Generate a dict representation of the layout object with the page_data and all the blocks in its dict representation.

Returns The dictionary representation of the layout object.

Return type Dict

get_homogeneous_blocks() → List[layoutparser.elements.base.BaseLayoutElement]
Convert all elements into blocks of the same type based on the type casting rule:

```
Interval < Rectangle < Quadrilateral < TextBlock
```

Returns A list of base layout elements of the maximal compatible type

Return type List[BaseLayoutElement]

to_dataframe(*enforce_same_type=False*) → pandas.core.frame.DataFrame
Convert the layout object into the dataframe. Warning: the page data won't be exported.

Parameters **enforce_same_type** (bool, optional) – If true, it will convert all the contained blocks to the maximal compatible data type. Defaults to False.

Returns The dataframe representation of layout object

Return type pd.DataFrame

CHAPTER

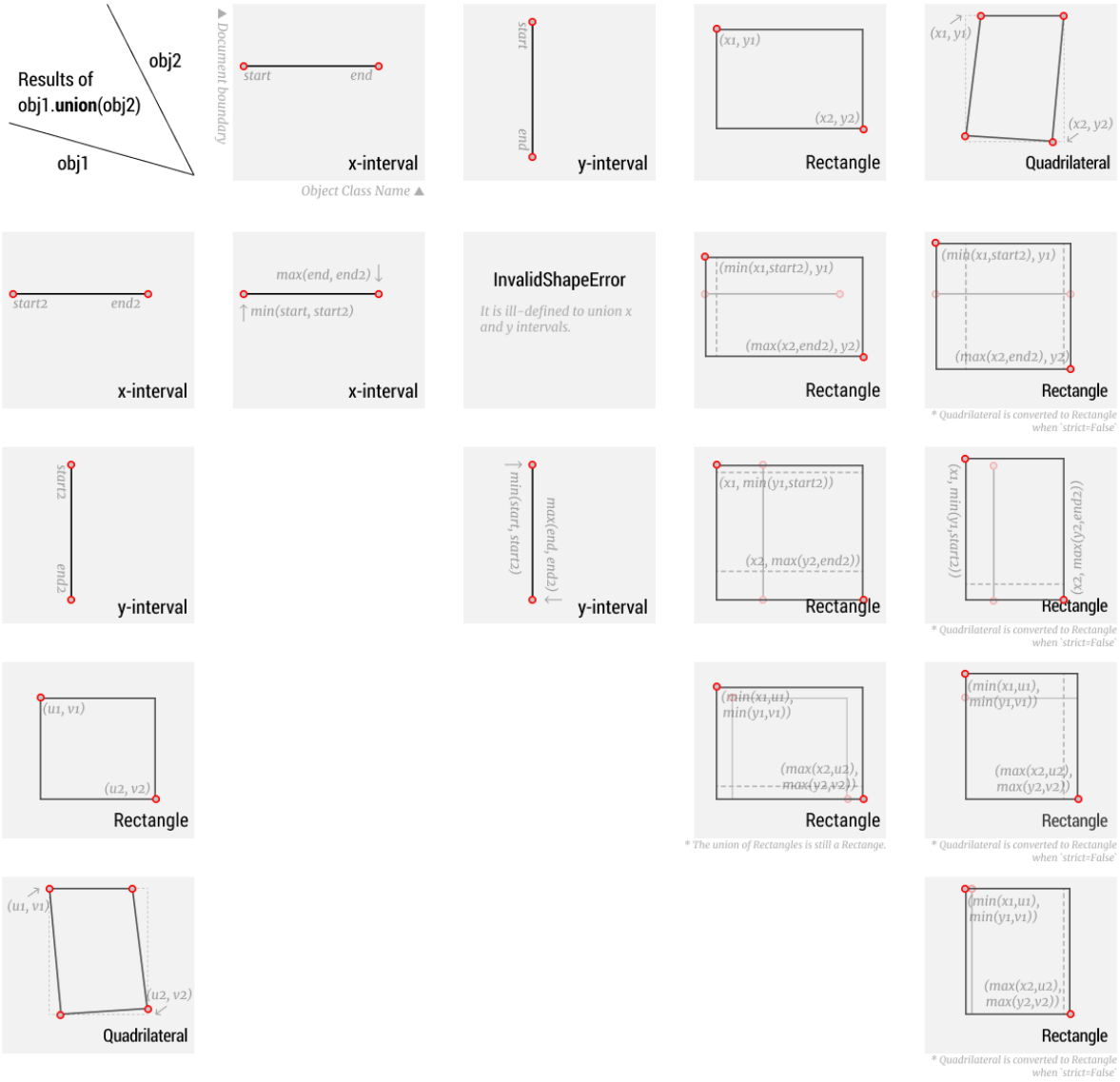
SEVEN

SHAPE OPERATIONS

[BETA: the API and behavior *will* be changed in the future.]

Starting from v0.2, Layout Parser provides supports for two types of shape operations, `union` and `intersection`, across all `BaseCoordElement`s and `TextBlock`. We've made some design choices to construct a set of generalized APIs across different shape classes, detailed as follows:

7.1 The `union` Operation

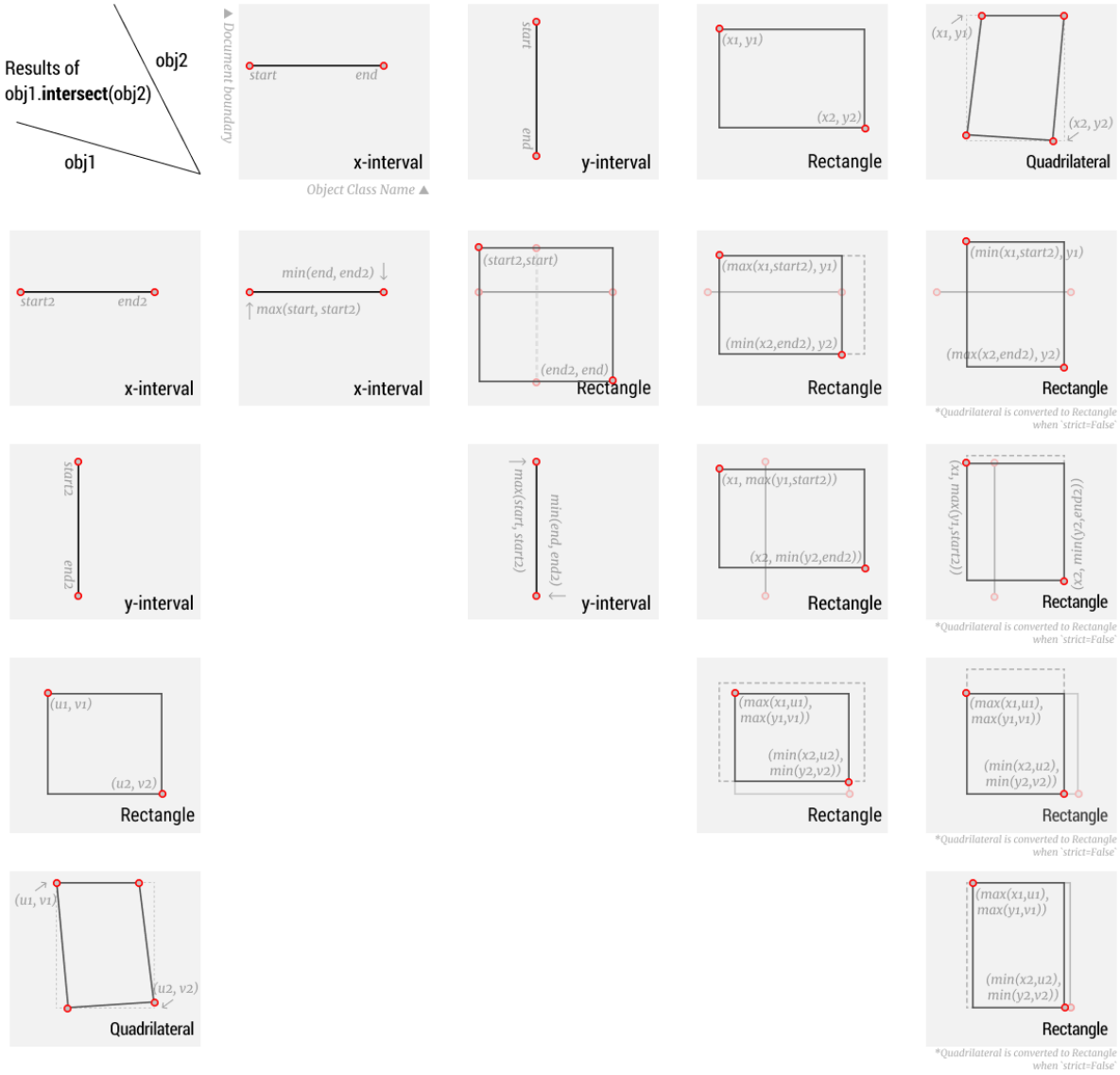


The Illustration of Union Operations. The resulting matrix are symmetric so only the lower triangular region is left empty. Each cell shows the visualization of the shape objects, their coordinates, and their object class. For the output visualization, the gray and dashed line delineates the original obj1 and obj2, respectively, for reference.

Notes:

1. The x-interval and y-interval are both from the `Interval` Class but with different axes. It's ill-defined to union two intervals from different axes so in this case Layout Parser will raise an `InvalidShapeError`.
2. The union of two rectangles is still a rectangle, which is the minimum covering rectangle of the two input rectangles.
3. For the outputs associated with `Quadrilateral` inputs, please see details in the *Problems related to the Quadrilateral Class* section.

7.2 The `intersect` Operation



The Illustration of Union Operations. Similar to the previous visualization, the resulting matrix are symmetric so only the lower triangular region is left empty. Each cell shows the visualization of the shape objects, their coordinates, and their object class. For the output visualization, the gray and dashed line delineates the original obj1 and obj2, respectively, for reference.

7.3 Problems related to the `Quadrilateral` Class

It is possible to generate arbitrary shapes when performing shape operations on `Quadrilateral` objects. Currently Layout Parser does not provide the support for `Polygon` objects (but we plan to support that object in the near future), thus it becomes tricky to add support for these operations for `Quadrilateral`. The temporary solution is that:

1. When performing shape operations on `Quadrilateral` objects, Layout Parser will raise `NotSupportedShapeError`.
2. A workaround is to set `strict=False` in the input (i.e., `obj1.union(obj2, strict=False)`). In this case, any quadrilateral objects will be converted to `Rectangles` first and the operation is executed. The results may not be *strictly* equivalent to those performed on the original objects.

CHAPTER

EIGHT

TEXT RECOGNITION TOOL

8.1 Google Cloud Vision API

class layoutparser.ocr.**GCVFeatureType**
Bases: layoutparser.ocr.base.BaseOCRElementType

The element types from Google Cloud Vision API

PAGE = 0

BLOCK = 1

PARA = 2

WORD = 3

SYMBOL = 4

property child_level

class layoutparser.ocr.**GCVAgent**(*languages=None*, *ocr_image_decode_type='.png'*)
Bases: layoutparser.ocr.base.BaseOCRAgent

A wrapper for Google Cloud Vision (GCV) Text Detection APIs.

Note: Google Cloud Vision API returns the output text in two types:

- *text_annotations*:

  In this format, GCV automatically find the best aggregation level for the text, and return the results in a list. We use *gather_text_annotations* to reterive this type of information.

- *full_text_annotation*:

  To support better user control, GCV also provides the *full_text_annotation* output, where it returns the hierarchical structure of the output text. To process this output, we provide the *gather_full_text_annotation* function to aggregate the texts of the given aggregation level.

Create a Google Cloud Vision OCR Agent.

Parameters

- **languages** (list, optional) – You can specify the language code of the documents to detect to improve accuracy. The supported language and their code can be found on this page. Defaults to None.

- **ocr_image_decode_type** (str, optional) – The format to convert the input image to before sending for GCV OCR. Defaults to *".png"*.
  - *".png"* is suggested as it does not compress the image.
  - But *".jpg"* could also be a good choice if the input image is very large.

DEPENDENCIES = ['google-cloud-vision']

classmethod with_credential(*credential_path*, ***kwargs*)
Specifiy the credential to use for the GCV OCR API.

Parameters **credential_path** (str) – The path to the credential file

detect(*image*, *return_response=False*, *return_only_text=False*, *agg_output_level=None*)
Send the input image for OCR.

Parameters

- **image** (np.ndarray or str) – The input image array or the name of the image file
- **return_response** (bool, optional) – Whether directly return the google cloud response. Defaults to *False*.
- **return_only_text** (bool, optional) – Whether return only the texts in the OCR results. Defaults to *False*.
- **agg_output_level** (*GCVFeatureType*, optional) – When set, aggregate the GCV output with respect to the specified aggregation level. Defaults to *None*.

static gather_text_annotations(*response*)
Convert the text_annotations from GCV output to an Layout object.

Parameters **response** (AnnotateImageResponse) – The returned Google Cloud Vision AnnotateImageResponse object.

Returns The reterived layout from the response.

Return type Layout

static gather_full_text_annotation(*response*, *agg_level*)
Convert the full_text_annotation from GCV output to an Layout object.

Parameters

- **response** (AnnotateImageResponse) – The returned Google Cloud Vision AnnotateImageResponse object.
- **agg_level** (*GCVFeatureType*) – The layout level to aggregate the text in full_text_annotation.

Returns The reterived layout from the response.

Return type Layout

load_response(*filename*)

save_response(*res*, *file_name*)

8.2 Tesseract OCR API

class layoutparser.ocr.**TesseractFeatureType**
Bases: layoutparser.ocr.base.BaseOCRElementType

The element types for Tesseract Detection API

PAGE = 0

BLOCK = 1

PARA = 2

LINE = 3

WORD = 4

property group_levels

class layoutparser.ocr.**TesseractAgent**(*languages='eng'*, ***kwargs*)
Bases: layoutparser.ocr.base.BaseOCRAgent

A wrapper for Tesseract Text Detection APIs based on PyTesseract.

Create a Tesseract OCR Agent.

Parameters **languages** (list or str, optional) – You can specify the language code(s) of the documents to detect to improve accuracy. The supported language and their code can be found on its github repo. It supports two formats: 1) you can pass in the languages code as a string of format like *"eng+fra"*, or 2) you can pack them as a list of strings *["eng", "fra"]*. Defaults to 'eng'.

DEPENDENCIES = ['pytesseract']

classmethod with_tesseract_executable(*tesseract_cmd_path*, ***kwargs*)

detect(*image*, *return_response=False*, *return_only_text=True*, *agg_output_level=None*)
Send the input image for OCR.

Parameters

- **image** (np.ndarray or str) – The input image array or the name of the image file
- **return_response** (bool, optional) – Whether directly return all output (string and boxes info) from Tesseract. Defaults to *False*.
- **return_only_text** (bool, optional) – Whether return only the texts in the OCR results. Defaults to *False*.
- **agg_output_level** (*TesseractFeatureType*, optional) – When set, aggregate the GCV output with respect to the specified aggregation level. Defaults to *None*.

static gather_data(*response*, *agg_level*)
Gather the OCR'ed text, bounding boxes, and confidence in a given aggeragation level.

static load_response(*filename*)

static save_response(*res*, *file_name*)

CHAPTER

NINE

LAYOUT DETECTION MODELS

class layoutparser.models.**Detectron2LayoutModel**(*config_path*, *model_path=None*, *label_map=None*, *extra_config=None*, *enforce_cpu=None*, *device=None*)

Bases: layoutparser.models.base_layoutmodel.BaseLayoutModel

Create a Detectron2-based Layout Detection Model

Parameters

- **config_path** (str) – The path to the configuration file.
- **model_path** (str, None) – The path to the saved weights of the model. If set, overwrite the weights in the configuration file. Defaults to *None*.
- **label_map** (dict, optional) – The map from the model prediction (ids) to real word labels (strings). If the config is from one of the supported datasets, Layout Parser will automatically initialize the label_map. Defaults to *None*.
- **device** (str, optional) – Whether to use cuda or cpu devices. If not set, LayoutParser will automatically determine the device to initialize the models on.
- **extra_config** (list, optional) – Extra configuration passed to the Detectron2 model configuration. The argument will be used in the merge_from_list function. Defaults to *[]*.

Examples::

```
>>> import layoutparser as lp
>>> model = lp.Detectron2LayoutModel('lp://HJDataset/faster_rcnn_R_50_FPN_3x/
↪config')
>>> model.detect(image)
```

DEPENDENCIES = ['detectron2']

DETECTOR_NAME = 'detectron2'

MODEL_CATALOG = {'HJDataset': {'faster_rcnn_R_50_FPN_3x': 'https://www.dropbox.com/s

gather_output(*outputs*)

detect(*image*)

Detect the layout of a given image.

Parameters **image** (np.ndarray or *PIL.Image*) – The input image to detect.

Returns The detected layout of the input image

Return type Layout

image_loader(*image: Union[np.ndarray, Image.Image]*)
It will process the input images appropriately to the target format.

CHAPTER

TEN

LAYOUT AND TEXT VISUALIZATION

layoutparser.visualization.**draw_box**(*canvas*, *layout*, *box_width=None*, *box_alpha=0*, *color_map=None*, *show_element_id=False*, *show_element_type=False*, *id_font_size=None*, *id_font_path=None*, *id_text_color=None*, *id_text_background_color=None*, *id_text_background_alpha=1*)

Draw the layout region on the input canvas(image).

Parameters

- **canvas** (ndarray or Image) – The canvas to draw the layout boxes.
- **layout** (Layout or list) – The layout of the canvas to show.
- **box_width** (int, optional) – Set to change the width of the drawn layout box boundary. Defaults to None, when the boundary is automatically calculated as the the DEFAULT_BOX_WIDTH_RATIO * the maximum of (height, width) of the canvas.
- **box_alpha** (float, optional) – A float range from 0 to 1. Set to change the alpha of the drawn layout box. Defaults to 0 - the layout box will be fully transparent.
- **color_map** (*dict, optional*) – A map from *block.type* to the colors, e.g., *{1: 'red'}*. You can set it to *{}* to use only the DEFAULT_OUTLINE_COLOR for the outlines. Defaults to None, when a color palette is is automatically created based on the input layout.
- **show_element_id** (*bool, optional*) – Whether to display *block.id* on the top-left corner of the block. Defaults to False.
- **show_element_type** (*bool, optional*) – Whether to display *block.type* on the top-left corner of the block. Defaults to False.
- **id_font_size** (*int, optional*) – Set to change the font size used for drawing *block.id*. Defaults to None, when the size is set to DEFAULT_FONT_SIZE.
- **id_font_path** (str, optional) – Set to change the font used for drawing *block.id*. Defaults to None, when the DEFAULT_FONT_OBJECT is used.
- **id_text_color** (str, optional) – Set to change the text color used for drawing *block.id*. Defaults to None, when the color is set to DEFAULT_TEXT_COLOR.
- **id_text_background_color** (str, optional) – Set to change the text region background used for drawing *block.id*. Defaults to None, when the color is set to DEFAULT_TEXT_BACKGROUND.
- **id_text_background_alpha** (float, optional) – A float range from 0 to 1. Set to change the alpha of the drawn text. Defaults to 1 - the text box will be solid.

Returns A Image object containing the *layout* draw upon the input *canvas*.

Return type PIL.Image.Image

layoutparser.visualization.**draw_text**(*canvas*, *layout*, *arrangement='lr'*, *font_size=None*, *font_path=None*, *text_color=None*, *text_background_color=None*, *text_background_alpha=1*, *vertical_text=False*, *with_box_on_text=False*, *text_box_width=None*, *text_box_color=None*, *text_box_alpha=0*, *with_layout=False*, ***kwargs*)

Draw the (detected) text in the *layout* according to their coordinates next to the input *canvas* (image) for better comparison.

Parameters

- **canvas** (ndarray or Image) – The canvas to draw the layout boxes.
- **layout** (Layout or list) – The layout of the canvas to show.
- **arrangement** (*{'lr', 'ud'}*, optional) – The arrangement of the drawn text canvas and the original image canvas: * *lr* - left and right * *ud* - up and down

  Defaults to 'lr'.
- **font_size** (str, optional) – Set to change the size of the font used for drawing *block.text*. Defaults to None, when the size is set to DEFAULT_FONT_SIZE.
- **font_path** (str, optional) – Set to change the font used for drawing *block.text*. Defaults to None, when the DEFAULT_FONT_OBJECT is used.
- **text_color** (*[type], optional*) – Set to change the text color used for drawing *block.text*. Defaults to None, when the color is set to DEFAULT_TEXT_COLOR.
- **text_background_color** (*[type], optional*) – Set to change the text region background used for drawing *block.text*. Defaults to None, when the color is set to DEFAULT_TEXT_BACKGROUND.
- **text_background_alpha** (float, optional) – A float range from 0 to 1. Set to change the alpha of the background of the canvas. Defaults to 1 - the text box will be solid.
- **vertical_text** (*bool, optional*) – Whether the text in a block should be drawn vertically. Defaults to False.
- **with_box_on_text** (*bool, optional*) – Whether to draw the layout box boundary of a text region on the text canvas. Defaults to False.
- **text_box_width** (int, optional) – Set to change the width of the drawn layout box boundary. Defaults to None, when the boundary is automatically calculated as the the DEFAULT_BOX_WIDTH_RATIO * the maximum of (height, width) of the canvas.
- **text_box_alpha** (float, optional) – A float range from 0 to 1. Set to change the alpha of the drawn text box. Defaults to 0 - the text box will be fully transparent.
- **text_box_color** (int, optional) – Set to change the color of the drawn layout box boundary. Defaults to None, when the color is set to DEFAULT_OUTLINE_COLOR.
- **with_layout** (*bool, optional*) – Whether to draw the layout boxes on the input (image) canvas. Defaults to False. When set to true, you can pass in the arguments in *draw_box* to change the style of the drawn layout boxes.

Returns A Image object containing the drawn text from *layout*.

Return type PIL.Image.Image

CHAPTER

ELEVEN

LOAD AND EXPORT LAYOUT DATA

11.1 *Dataframe* and CSV

layoutparser.io.**load_dataframe**(*df: pandas.core.frame.DataFrame*, *block_type: str = None*) → layoutparser.elements.layout.Layout

Load the Layout object from the given dataframe.

Parameters

- **df** (*pd.DataFrame*) –
- **block_type** (*str*) – If there's no block_type column in the CSV file, you must pass in a block_type variable such that layout parser can appropriately detect the type of the layout elements.

Returns The parsed Layout object from the CSV file.

Return type *Layout*

layoutparser.io.**load_csv**(*filename: str*, *block_type: str = None*) → layoutparser.elements.layout.Layout

Load the Layout object from the given CSV file.

Parameters

- **filename** (*str*) – The name of the CSV file. A row of the table represents an individual layout element.
- **block_type** (*str*) – If there's no block_type column in the CSV file, you must pass in a block_type variable such that layout parser can appropriately detect the type of the layout elements.

Returns The parsed Layout object from the CSV file.

Return type *Layout*

11.2 *Dict* and JSON

layoutparser.io.**load_dict**(*data: Union[Dict, List[Dict]]*) → Union[layoutparser.elements.base.BaseLayoutElement, layoutparser.elements.layout.Layout]

Load a dict of list of dict representations of some layout data, automatically parse its type, and save it as any of BaseLayoutElement or Layout datatype.

Parameters **data** (*Union[Dict, List]*) – A dict of list of dict representations of the layout data

Raises

- **ValueError** – If the data format is incompatible with the layout-data-JSON format, raise a *ValueError*.
- **ValueError** – If any *block_type* name is not in the available list of layout element names defined in *BASECOORD_ELEMENT_NAMEMAP*, raise a *ValueError*.

Returns Based on the dict format, it will automatically parse the type of the data and load it accordingly.

Return type Union[BaseLayoutElement, *Layout*]

layoutparser.io.**load_json**(*filename: str*) → Union[layoutparser.elements.base.BaseLayoutElement, layoutparser.elements.layout.Layout]

Load a JSON file and save it as a layout object with appropriate data types.

Parameters **filename** (*str*) – The name of the JSON file.

Returns Based on the JSON file format, it will automatically parse the type of the data and load it accordingly.

Return type Union[BaseLayoutElement, *Layout*]

11.3 PDF

layoutparser.io.**load_pdf**(*filename: str*, *load_images: bool = False*, *x_tolerance: int = 1.5*, *y_tolerance: int = 2*, *keep_blank_chars: bool = False*, *use_text_flow: bool = True*, *horizontal_ltr: bool = True*, *vertical_ttb: bool = True*, *extra_attrs: Optional[List[str]] = None*, *dpi: int = 72*) → Union[List[layoutparser.elements.layout.Layout], Tuple[List[layoutparser.elements.layout.Layout], List[Image.Image]]]

Load all tokens for each page from a PDF file, and save them in a list of Layout objects with the original page order.

Parameters

- **filename** (*str*) – The path to the PDF file.
- **load_images** (*bool, optional*) – Whether load screenshot for each page of the PDF file. When set to true, the function will return both the layout and screenshot image for each page. Defaults to False.
- **x_tolerance** (*int, optional*) – The threshold used for extracting "word tokens" from the pdf file. It will merge the pdf characters into a word token if the difference between the x_2 of one character and the x_1 of the next is less than or equal to x_tolerance. See details in pdf2plumber's documentation. Defaults to 1.5.
- **y_tolerance** (*int, optional*) – The threshold used for extracting "word tokens" from the pdf file. It will merge the pdf characters into a word token if the difference between the y_2 of one character and the y_1 of the next is less than or equal to y_tolerance. See details in pdf2plumber's documentation. Defaults to 2.
- **keep_blank_chars** (*bool, optional*) – When keep_blank_chars is set to True, it will treat blank characters are treated as part of a word, not as a space between words. See details in pdf2plumber's documentation. Defaults to False.
- **use_text_flow** (*bool, optional*) – When use_text_flow is set to True, it will use the PDF's underlying flow of characters as a guide for ordering and segmenting the words, rather than presorting the characters by x/y position. (This mimics how dragging a cursor

highlights text in a PDF; as with that, the order does not always appear to be logical.) See details in pdf2plumber's documentation. Defaults to True.

- **horizontal_ltr** (*bool, optional*) – When horizontal_ltr is set to True, it means the doc should read text from left to right, vice versa. Defaults to True.
- **vertical_ttb** (*bool, optional*) – When vertical_ttb is set to True, it means the doc should read text from top to bottom, vice versa. Defaults to True.
- **extra_attrs** (*Optional[List[str]], optional*) – Passing a list of extra_attrs (e.g., ["fontname", "size"]) will restrict each words to characters that share exactly the same value for each of those attributes extracted by pdfplumber, and the resulting word dicts will indicate those attributes. See details in pdf2plumber's documentation. Defaults to *["fontname", "size"]*.
- **dpi** (*int, optional*) – When loading images of the pdf, you can also specify the resolution (or DPI, dots per inch) for rendering the images. Higher DPI values mean clearer images (also larger file sizes). Setting dpi will also automatically resizes the extracted pdf_layout to match the sizes of the images. Therefore, when visualizing the pdf_layouts, it can be rendered appropriately. Defaults to *DEFAULT_PDF_DPI=72*, which is also the default rendering dpi from the pdfplumber PDF parser.

Returns

When *load_images=False*, it will only load the pdf_tokens from the PDF file. Each element of the list denotes all the tokens appeared on a single page, and the list is ordered the same as the original PDF page order.

Tuple[List[Layout], List["Image.Image"]]: When *load_images=True*, besides the *all_page_layout*, it will also return a list of page images.

Return type List[*Layout*]

Examples::

```
>>> import layoutparser as lp
>>> pdf_layout = lp.load_pdf("path/to/pdf")
>>> pdf_layout[0] # the layout for page 0
>>> pdf_layout, pdf_images = lp.load_pdf("path/to/pdf", load_images=True)
>>> lp.draw_box(pdf_images[0], pdf_layout[0])
```

11.4 Other Formats

Stay tuned! We are working on to support more formats.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- search

PYTHON MODULE INDEX

INDEX

H

I

L

M

P

Q

R

S

T

U

W