Published in Social Impact Analytics

Jonathan Lee Follow

Apr 9, 2021 · 5 min read · Listen

Extract Text From Unsearchable PDFs Using OCR, Tesseract, and Python

Photo by Loverna Journey on Unsplash

At ***Social Impact Analytics Institute***, we do a lot of text extraction on PDF files. Problems arise when the PDF files are scanned documents, because that means general extraction libraries like Pdfminer, PyPDF2, or PyMuPDF are not able to extract text correctly. In order to read the files, make the text content of the files searchable, and be able to do further NLP data analysis, an OCR process must be used.

In this article, I'm going to demonstrate how to use an open source OCR engine (Optical Character Recognition) called ***Tesseract*** and its ***Python APIs*** to conduct text extraction and then put the text into a pandas data frame for further data analysis.

Quick view of Contents

1. Install Tesseract engine.
2. Install Python Packages: PyTesseract, OCRmyPDF.
3. Code Example for Text Extraction

Tesseract Installation

Tesseract is a free command line application powered by Google. It has been widely used for OCR tasks, however, it is not a python library so it must be installed separately. I suggest installing Tesseract on ***Mac OS, Ubuntu*** or another Linux-like system. However, if you are using Windows, this article shows how to install it on Windows 10.

Mac : `brew install tesseract`

If you already have homebrew, simply enter this code on your terminal and then you are done. If not, check homebrew's home page to install it.

Ubuntu: `sudo apt-get install tesseract-ocr`

Python Packages

There are 2 ways to use the Tesseract engine in this article: through ***Pytesseract*** or through ***OCRmyPDF.*** Both require the installation of additional libraries. I will show how these work together in the following sections. First, install the packages through the pip command in a python environment:

```
pip install ocrmypdf
pip install pytesseract
pip install opencv-python
pip install pdf2image
```

1. Pytesseract

Pytesseract reads the input file as an image, so ***opencv-python*** and ***pdf2image*** are included to help transfer PDF files into images. The steps will look like this:

1. Read PDF files
2. Convert PDFs into images
3. Image preprocessing to check orientation, deskew, gray scale, etc.
4. Run OCR function from its API
5. Extract all/part of the text

The syntax of the main OCR function is:

```
pytesseract.image_to_string(page_arr)
```

The code, including image processing steps and how to put the resulting text into a pandas data frame, is shown below. The code for the deskew function is referenced here.

```
# import libs
try:
    from PIL import Image
except ImportError:
    import Image
import cv2
import pytesseract
import os
import numpy as np
import pandas as pd
import re
from pdf2image import convert_from_bytes

# Some help functions
def get_conf(page_gray):
    '''return a average confidence value of OCR result '''
    df = pytesseract.image_to_data(page_gray,output_type='data.frame')
    df.drop(df[df.conf==-1].index.values,inplace=True)
    df.reset_index()
    return df.conf.mean()

def deskew(image):
    '''deskew the image'''
    gray = cv2.bitwise_not(image)
    temp_arr = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)[1]
    coords = np.column_stack(np.where(temp_arr > 0))
    angle = cv2.minAreaRect(coords)[-1]
    if angle < -45:
        angle = -(90 + angle)
    else:
        angle = -angle
    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC, borderMode=cv2.BORDER_REPLIC
    return rotated

'''
Main part of OCR:
pages_df: save eextracted text for each pdf file, index by page
OCR_dic : dict for saving df of each pdf, filename is the key
'''

%%time
OCR_dic={}
```

```
OCR_dic={}
for file in file_list:
    # convert pdf into image
    pdf_file = convert_from_bytes(open(os.path.join(PATH,file), 'rb').read())
    # create a df to save each pdf's text
    pages_df = pd.DataFrame(columns=['conf','text'])
    for (i,page) in enumerate(pdf_file) :
        try:
```

```
OCR_dic[file_list[0]]
```

	conf	text
0	92.349593	Case 5:09-cv-00032-TBR Document97 Filed 01/25/...
1	94.150246	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
2	93.122507	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
3	94.162963	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
4	93.985437	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
5	94.814318	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
6	93.959239	Case 5:09-cv-00032-TBR Document97 Filed 01/25/...
7	93.922252	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
8	92.991803	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
9	95.027473	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
10	94.216617	Case 5:09-cv-00032-TBR Document97 Filed 01/25/...
11	94.326923	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...
12	93.669725	Case 5:09-cv-00032-TBR Document 97 Filed 01/25...

Seems odd that all the text files start with identical wording. This is a clue that a header may be in use.

2.1 Remove Header and Footer

After displaying the result, it seems that the header was included and it may not be wanted or required. Here I will show a simple code to exclude certain contents from

our extraction results by using ***pytesseract.image_to_data*** syntax, which generates meta-data like box boundaries, confidence and other information.

```
pytesseract.image_to_data(page_arr,output_type=Output.DICT)
```

The keys of the output dictionary will look like this:

```
dict_keys(['level', 'page_num', 'block_num', 'par_num', 'line_num',
'word_num', 'left', 'top', 'width', 'height', 'conf', 'text'])
```

Note that ***level*** is critical to interpret these results. It will be an integer between 1~5 and each of them stands for

1. page
2. block
3. paragraph
4. line
5. word

Let see how it works on the PDF. Here I use a file's first page as an example. I want to see how many blocks will be generated and want to see if it includes the header and footer.

```python
page_arr = np.asarray(pdf_file[0])

page_arr_gray = cv2.cvtColor(page_arr,cv2.COLOR_BGR2GRAY)

d = pytesseract.image_to_data(page_arr_gray,output_type=pytesseract.Output.DICT)

d_df = pd.DataFrame.from_dict(d)

n_boxes = len(d['text'])

for i in range(n_boxes):
    # level ==2 : show all the blocks recognized by tesseract
    if d['level'][i] ==2:

        (x, y, w, h) = (d['left'][i], d['top'][i], d['width'][i], d['height'][i])
        # draw green lines on boxes
        img = cv2.rectangle(page_arr, (x, y), (x + w, y + h), (0, 255, 0), 2)

Image.fromarray(img)
```

header.py hosted with ❤ by GitHub view raw

```python
d_df[d_df['level']==2]
```

	level	page_num	block_num	par_num	line_num	word_num	left	top	width	height	conf	text
1	2	1	1	0	0	0	233	65	1227	35	-1	
16	2	1	2	0	0	0	511	246	673	63	-1	
32	2	1	3	0	0	0	226	321	1080	624	-1	
86	2	1	4	0	0	0	542	949	613	25	-1	
93	2	1	5	0	0	0	541	974	615	7	-1	
97	2	1	6	0	0	0	226	1023	1248	764	-1	
259	2	1	7	0	0	0	230	1828	1244	32	-1	
277	2	1	8	0	0	0	231	1908	1212	58	-1	

Blocks generated by Tesseract — notice there are a total 8 blocks. This can be checked by looking at the block_num column.

IN THE UNITED STATES DISTRICT COURT
FOR THE WESTERN DISTRICT OF KENTUCKY
AT PADUCAH

MICHAEL ADAM CARNEAL,	)	
	)	
Petitioner,	)	Case No.: 5:09-cv-00032-TBR
	)	
vs.	)	**Electronically Filed**
	)	
J. DAVID DONAHUE, WARDEN,	)	
	)	

PETITIONER'S PRE-HEARING AND BRIEFING STATEMENT AND EVIDENTIARY ISSUE STATEMENT

Comes Petitioner, by counsel, and hereby complies with this Court's Order of September 10, 2010 and submits his Prehearing and Briefing Statement and Evidentiary Issue Statement. See DN 60. In this Court's Order granting an evidentiary hearing, this Court stated: "Carneal has presented at least a colorable argument that equitable tolling/actual innocence applies in this case based on his alleged mental incompetency." DN 32, p. 2. This Court then went on to state: "Based on the record thus far, the Court finds an evidentiary hearing would allow Carneal the opportunity to fully develop his claim of equitable tolling and is necessary for the Court to properly decide the pending motion to dismiss." Id., p. 3. Thus, Petitioner plans to put forward evidence at the hearing evidence to support equitable tolling.[1]

STATEMENT OF THE CASE

On the morning of December 1, 1997, Petitioner, Michael Carneal, opened fire on a group of students at the Heath High School in Paducah, Kentucky, killing three people and

[1] The Sixth Circuit treats actual innocence as a sub-category of equitable tolling. McSwain v. Davis, 287 F.Appx 450, 461-463 (6th Cir. 2008).

Blocks highlighted by green lines

The image and data frame above show that 8 blocks were generated from OCR, and fortunately the header and footer are included in the first and the last blocks. If I want to remove both of them, it is easy to achieve by removing these two blocks.

```
OCR_dic[file_list[0]]
```

	conf	text
0	91.450980	IN THE UNITED STATES DISTRICT COURT FOR THE WE...
1	94.316832	wounding five others. He was fourteen years ol...
2	93.270655	would “creep up and look in my windows.” TRSE,...
3	94.972840	[Carneal's] plea of guilty but mentally ill.” ...
4	93.837379	extent of his pervasive delusional system. TRS...
5	95.286353	have offered a stronger opinion about his ment...
6	93.874659	the motion - which was filed three years to th...
7	94.066845	innocence issues addressed in state court. The...
8	93.706849	This Court has already ruled that in this case...
9	94.906336	innocence exception to the dictates of 2244(d)...
10	94.310651	his ability to understand himself and make rat...
11	93.193050	as to their perceptions of his mental health d...
12	94.055556	Fax: 502/222-3177 David.Harshaw@ky.gov w/permi...

Now the extracted text does not include the header or footer. However, for different PDFs, this code may need to be modified or headers and footers may not be so easily excluded.

2. OCRmyPDF

It is more simple to use OCRmyPDF when compared to Pytesseract because OCRmyPDF doesn’t require image processing steps. The output result of OCRmyPDF will be a PDF file, though, so additional steps are required if one wants to extract its text data. Here I use PyMuPDF to do that, which provides multiple output format options like xml, html or json. Choose whichever type works best for your purposes. The steps of the process are:

1. Run the OCRmyPDF function to create new, searchable PDF files.

2. Use PyMuPDF to extract text from the PDF.

The code for using OCRmyPDF will look like this:

To show the result of the first PDF file:

```
extraction_pdfs[ocr_file_list[0]]
```

Conclusion

In this article, I've shared code for how to use two popular Tesseract python APIs to conduct OCR on PDF files. My suggestions for which API should be used are:

1. For simple OCR tasks where text extraction is a minor part of the whole project, you should use OCRmyPDF. It is faster and easier to conduct the OCR process without the extra image processing steps, and you can leave meta-data extraction to PyMuPDF once the PDF files are transferred into searchable PDF.
2. If the goal is to do customized text extraction such as getting the date/time from a group of receipts or removing headers and footers as I did in this article, then using Pytesseract will be more appropriate. Pytesseract prevents you from having to read the file twice, and filter conditions can be written during the OCR.

Reference

https://tesseract-ocr.github.io/

https://pypi.org/project/pytesseract/

https://ocrmypdf.readthedocs.io/en/latest/index.html

https://www.pyimagesearch.com/2017/02/20/text-skew-correction-opencv-python/

Python Pytesseract Social Impact NLP Ocr


Download on the
App Store


GET IT ON
Google Play