Extracting Text from PDF Files

B Dr Booma · Follow
7 min read · Jul 26, 2023

Optical Character Recognition (OCR) *is a technology that enables the extraction of text from images or scanned documents. It plays a crucial role in various applications, including Natural Language Processing (NLP) and text summarization. In this article, we will explore OCR, NLP, and text summarization, and understand how OCR comes into the picture for these tasks.*

1. **Optical Character Recognition (OCR):**

OCR is a technology that converts images containing text into machine-readable text data. It utilizes advanced algorithms and machine learning models to recognize characters, words, and sentences in images and convert them into editable and searchable text. OCR has a wide range of applications, including digitization of printed documents, data extraction from invoices, forms, and receipts, and making scanned books accessible to visually impaired individuals.

2. **Natural Language Processing (NLP):**

NLP is a subfield of artificial intelligence that focuses on the interaction between computers and human language. It involves the analysis, understanding, and generation of natural language text and speech. NLP algorithms enable machines to comprehend and interpret human language, facilitating tasks like sentiment analysis, machine translation, and chatbots.

3. **Text Summarization:**

Text summarization is the process of condensing a lengthy piece of text into a shorter, concise version while retaining its main ideas and key information. There are two main types of text summarization: extractive and abstractive. Extractive summarization involves selecting and extracting important sentences directly from the original text, while abstractive summarization involves generating new sentences to summarize the content.

Role of OCR in NLP and Text Summarization:

1. **NLP Applications:**

OCR plays a vital role in NLP by enabling the extraction of text from images, which can then be processed and analyzed using NLP techniques. For example, in sentiment analysis, OCR can be used to extract text from social media images or memes, allowing sentiment analysis algorithms to understand people's emotions and opinions expressed in images.

2. Data Collection and Preprocessing:

In many NLP tasks, data is collected from various sources, including scanned documents, images, and handwritten notes. OCR comes into the picture by extracting text from these sources, converting them into machine-readable formats, and facilitating further analysis.

3. Accessibility and Inclusivity:

OCR contributes to making digital content more accessible and inclusive. By extracting text from scanned books or images, OCR allows visually impaired individuals to access and interact with the content using text-to-speech technologies.

4. Text Summarization:

OCR is an essential component in the text summarization process, particularly in extractive summarization. It enables the extraction of text from images, making it possible to summarize content present in scanned documents, presentation slides, or newspaper clippings. OCR helps in identifying the relevant sentences and information for generating concise summaries.

Now, to help you understand OCR in a better way, I will walk you through a detailed workflow:

- Read PDF files
- Convert them into images
- Perform image preprocessing to handle orientation and deskew issues
- Finally, extract text from these images using OCR

We will accomplish all these tasks using Python and various libraries, making the process both straightforward and effective.

> *Requirements:*
>
> *Before we start, make sure you have the following libraries installed:*
>
> *1. pdf2image: To convert PDF files into images.*
>
> *2. pytesseract: A Python wrapper for Google's Tesseract OCR engine.*
>
> *3. OpenCV: For image preprocessing tasks like deskewing and grayscale conversion.*
>
> *4. pandas: For storing extracted text data in a structured manner.*

Step 1: Reading PDF Files

To start our workflow, we need to read the PDF files. For this purpose, we will use the `pdf2image` library, which converts PDF pages into images.

```
from pdf2image import convert_from_path

# Replace 'input_file.pdf' with the path to your PDF file
pdf_file = 'input_file.pdf'
pages = convert_from_path(pdf_file)
```

Here, we import the `convert_from_path` function from the `pdf2image` library. This function is used to convert PDF pages into images. We specify the path to the input PDF file in the `pdf_file` variable, and then we call `convert_from_path(pdf_file)` to obtain a list of image objects corresponding to each page of the PDF.

Step 2: Image Preprocessing

Once we have the images from the PDF pages, we may encounter some issues like skewed or rotated pages. To handle these issues, we can perform image preprocessing using OpenCV. We will implement the `deskew` function to correct the orientation of the images.

```
import cv2
import numpy as np

def deskew(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.bitwise_not(gray)
    coords = np.column_stack(np.where(gray > 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

    return rotated
```

Above, we import the OpenCV library as `cv2`. The `deskew` function is defined to correct the orientation of the image. It takes the input image and returns the deskewed image. The actual implementation of the deskewing process is not provided in the code snippet, but it involves transforming the image to handle skew or rotation issues.

Step 3: Running OCR using pytesseract

Now that our images are preprocessed, we can run the OCR process using pytesseract. This library acts as a wrapper around Google's Tesseract OCR engine.

```
import pytesseract

def extract_text_from_image(image):
    text = pytesseract.image_to_string(image)
    return text
```

We import the `pytesseract` library. The `extract_text_from_image` function is defined to perform OCR on the input image and return the extracted text as

a string. It uses the `image_to_string` function from `pytesseract`, which takes an image as input and runs the OCR process on it to extract text.

Step 4: Text Extraction

With the OCR process completed, we can now extract the text from the images.

```
# Create a list to store extracted text from all pages
extracted_text = []

for page in pages:
    # Step 2: Preprocess the image (deskew)
    preprocessed_image = deskew(np.array(page))

    # Step 3: Extract text using OCR
    text = extract_text_from_image(preprocessed_image)
    extracted_text.append(text)
```

Above, `deskew` function is defined to correct the orientation of the image. It takes the input image and returns the deskewed image. The actual implementation of the deskewing process is not provided in the code snippet, but it involves transforming the image to handle skew or rotation issues.

Step 5: Text Extraction with Additional Preprocessing

The below process includes additional preprocessing to exclude the header and footer regions from the OCR process.

```
def process_page(page):
    try:
        # Transfer image of pdf_file into array
        page_arr = np.array(page)
        # Transfer into grayscale
        page_arr_gray = cv2.cvtColor(page_arr, cv2.COLOR_BGR2GRAY)
        # Deskew the page
        page_deskew = deskew(page_arr_gray)
        # Cal confidence value
        page_conf = get_conf(page_deskew)
        # Extract string
        d = pytesseract.image_to_data(page_deskew, output_type=pytesseract.Outpu
        d_df = pd.DataFrame.from_dict(d)
        # Get block number
        block_num = int(d_df.loc[d_df['level'] == 2, 'block_num'].max())
        # Drop header and footer by index
        header_index = d_df[d_df['block_num'] == 1].index.values
        footer_index = d_df[d_df['block_num'] == block_num].index.values
        # Combine text in dataframe, excluding header and footer regions
        text = ' '.join(d_df.loc[(d_df['level'] == 5) & (~d_df.index.isin(header
        return page_conf, text
    except Exception as e:
        # If can't extract then give some notes into df
        if hasattr(e, 'message'):
            return -1, e.message
        else:
            return -1, str(e)
```

In this above code, we import the necessary libraries and define the `process_page` function, which performs the OCR process on each page. It takes an image of a page as input and performs the following steps:

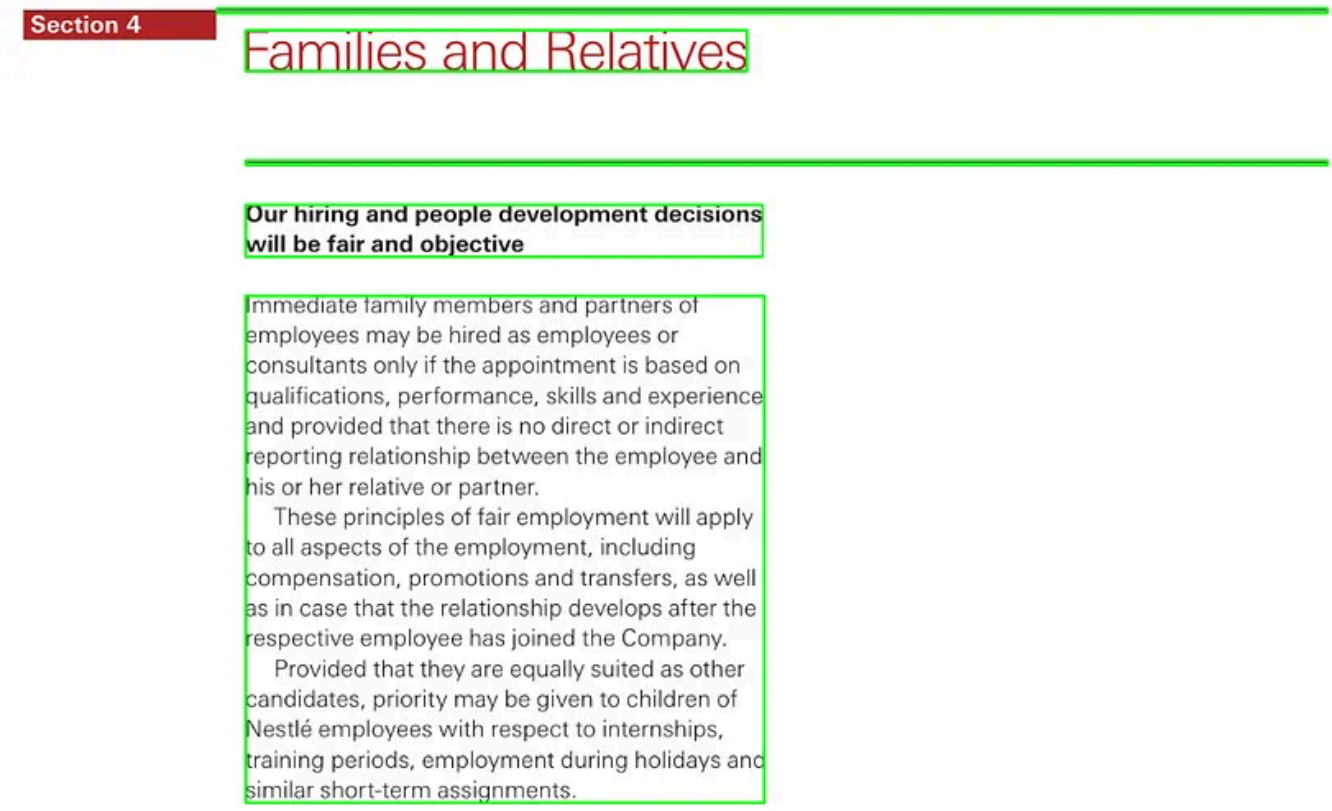
1. Transfers the image to an array and converts it into grayscale.
2. Deskews the page using the `deskew` function.
3. Calculates the confidence value (page_conf) for the extracted text using the `get_conf` function (not provided in the code snippet).

4. Extracts the text using `pytesseract.image_to_data` and stores it in a pandas DataFrame (d_df).
5. Identifies the block number of the last block in the OCR result to determine the footer's position.
6. Excludes the header and footer regions by their index in the DataFrame and combines the text using the `join` function.

In the main part of the code, we loop through each PDF file in `file_list`. For each file, we convert it into images, preprocess each page using the `process_page` function, and extract the text. We store the extracted text in a DataFrame `pages_df`, and then concatenate all the page DataFrames into a single DataFrame. Finally, we store the combined DataFrame for each PDF file in the dictionary `OCR_dic` with the filename as the key. The process is repeated for all the PDF files in the `file_list`, and the extracted text is saved in `OCR_dic`.

> *Note: The `poppler_path` parameter for `convert_from_bytes` function is mentioned but not provided in the code snippet. Make sure to include the correct path to the Poppler library if you encounter any issues related to it.*

Conclusion



showing all the blocks recognized by tesseract

In this article, I have walked you through a detailed workflow to extract text from PDF files using OCR. We started by reading the PDF files and converting them into images using `pdf2image`. Next, we performed image preprocessing tasks like deskewing using OpenCV to ensure the text recognition accuracy. Finally, we used pytesseract to run the OCR process and extract the text from the images.

Refer the github link for full code

With this workflow, you can now efficiently extract text from PDF documents, making them accessible for further analysis and processing in

various applications. OCR technology has proven to be an indispensable tool in modern data processing pipelines and can greatly enhance efficiency and productivity when dealing with scanned documents or images with embedded text.

Happy coding!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

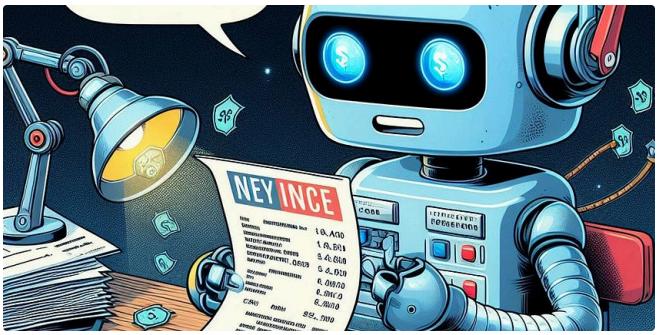
NLP Text Summarization AI Text Extraction Ocr

Written by Dr Booma

47 Followers

Follow

More from Dr Booma



Dr Booma

Microsoft Azure AI Document Intelligence with Custom build...

Transforming Your Documents into Actionable Data with Azure AI Document...

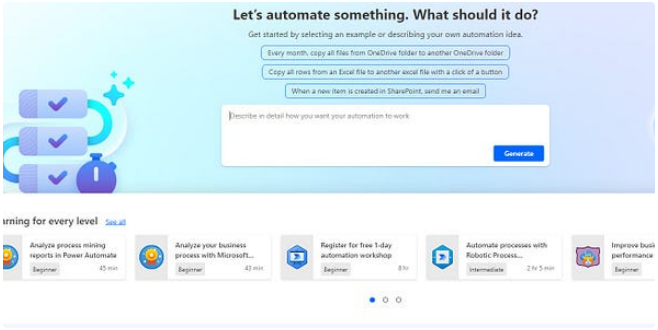
Mar 20 4

Dr Booma

A Python Guide to Crafting Dynamic Question & Answer...

In the ever-expanding landscape of information, the ability to derive meaningful...

Nov 27, 2023 6

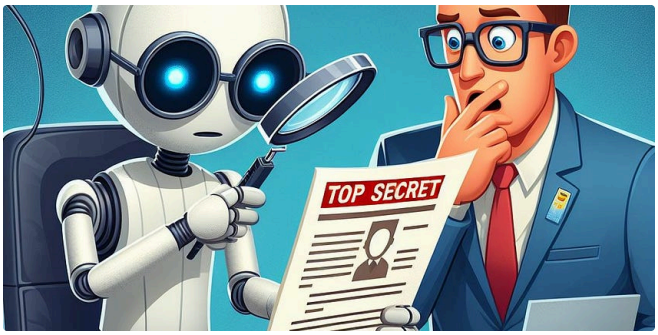


Dr Booma

Mastering Entity Extraction from Email: A Step-by-Step Guide with...

Embark on a journey of effortless efficiency with Power Automate, where simplicity meet...

Mar 1 1 1



Dr Booma

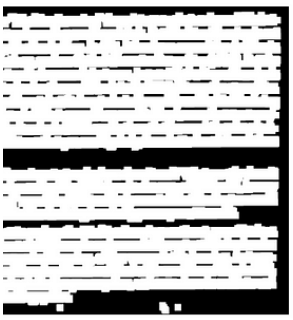
Microsoft Azure AI Document Intelligence with Pre-built Models...

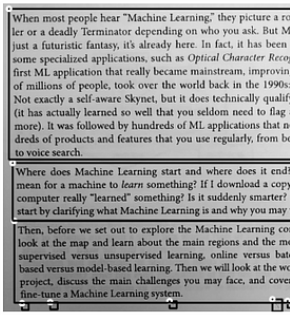
Navigating the digital era requires a keen understanding of how to extract valuable...

Mar 18 25

See all from Dr Booma

Recommended from Medium





siromer

Extracting Text from Images(OCR) using OpenCV & Pytesseract

Text Extraction from Pages & Online Documentations. OpenCV, Python,...

Mar 17 63

Ingrid Stevens

Extract Structured Data from Unstructured Text using LLMs

Using LangChain's create_extraction_chain and PydanticOutputParser

Jan 1 644 3

Lists

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 444 saves

Natural Language Processing

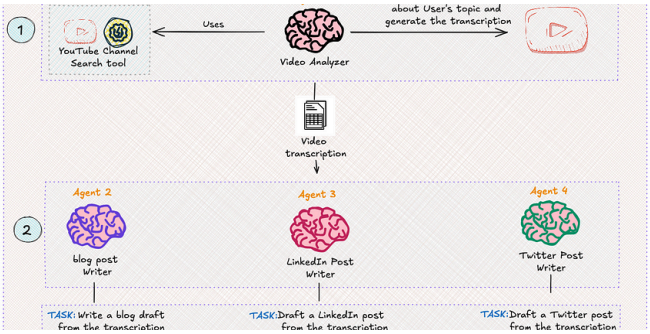
1656 stories · 1231 saves

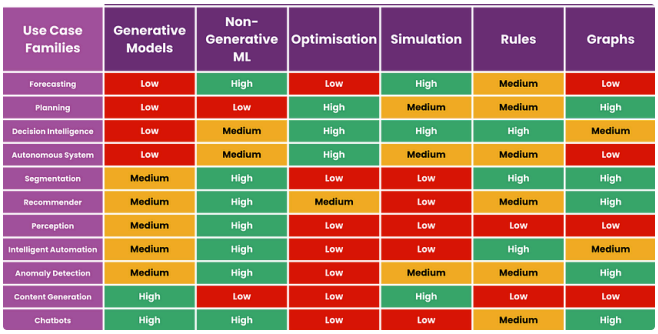
Generative AI Recommended Reading

52 stories · 1291 saves

What is ChatGPT?

9 stories · 424 saves





Use Case Families	Generative Models	Non-Generative ML	Optimisation	Simulation	Rules	Graphs
Forecasting	Low	High	Low	High	Medium	Low
Planning	Low	Low	High	Medium	Medium	High
Decision Intelligence	Low	Medium	High	High	High	Medium
Autonomous System	Low	Medium	High	Medium	Medium	Low
Segmentation	Medium	High	Low	Low	High	High
Recommender	Medium	High	Medium	Low	Medium	High
Perception	Medium	High	Low	Low	Low	Low
Intelligent Automation	Medium	High	Low	Low	High	Medium
Anomaly Detection	Medium	High	Low	Medium	Medium	High
Content Generation	High	Low	Low	High	Low	Low
Chatbots	High	High	Low	Low	Medium	High

Zoumana Keita in Towards Data Science

AI Agents—From Concepts to Practical Implementation in Python

This will change the way you think about AI and its capabilities

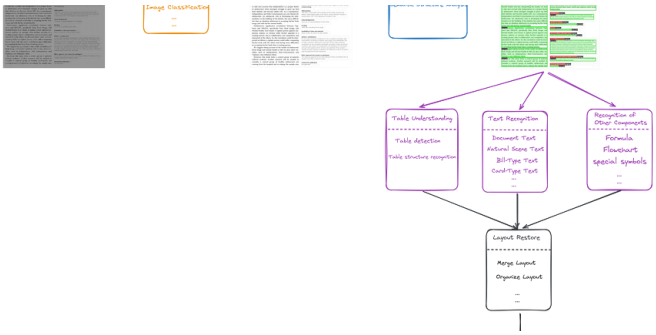
Aug 12 767 5

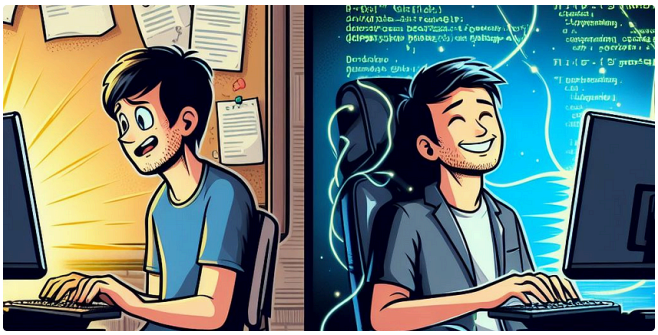
Christopher Tao in Towards AI

Do Not Use LLM or Generative AI For These Use Cases

Choose correct AI techniques for the right use case families

Aug 10 1.5K 16





Florian June in Generative AI

Demystifying PDF Parsing 01: Overview

Task Definition, Method Classification and Method Introduction to PDF Parsing

May 5 305

Abhay Parashar in The Pythoneers

17 Mindblowing Python Automation Scripts I Use Everyday

Scripts That Increased My Productivity and Performance

Jul 29 6.6K 56

See more recommendations