LegalDoc Extractor

Team: Technophile

Param Patil and Manav Mandal

PROBLEM STATEMENT:

Some legal firms have combined PDF documents that contain multiple sub-documents, e.g. Appearance, Complaint, Summons, Notice, Affidavit, etc. So, the first three pages of a PDF might be the Complaint; fourth page might be the Appearance; fifth page might be the summons, and so on. When these documents are submitted to the courts, they need to be separate PDFs. i.e. Complaint.pdf and Summons.pdf, Appearance.pdf, etc. Some large Law Firms have hundreds or maybe thousands of combined PDFs that need to be separated into individual PDFs. So, with OCR, we can simplify the process. The first page of each sub-document indicates the type of sub-document. That is, page 1 would say COMPLAINT, and pages 2 and 3 would have no identifying text; page 4 would say SUMMONS, etc. The combined PDF would be named with the defendant ID, e.g. 123456.pdf and as each sub-document is separated, 123456_Complaint.pdf, 123456_Summons.pdf, 123456_Appearance.pdf could be extracted.

PROJECT OVERVIEW:

The OCR Project for Hackathon aims to address the challenges faced by legal firms in managing combined PDF documents containing multiple sub-documents. These documents, which include complaints, summonses, appearances, and other legal forms, often need to be separated into individual PDF files for submission to courts and other legal proceedings.

The project leverages Optical Character Recognition (OCR) technology to automate the extraction and separation of sub-documents from combined PDF files. By identifying specific keywords or phrases on each page of the PDF, such as "COMPLAINT," "SUMMONS," and "APPEARANCE," the tool can determine the boundaries of each sub-document and create separate PDF files accordingly.

Key components of the project include:

1. **Configuration File:** The project utilizes a configuration file to define the input and output directories, as well as the keywords and corresponding search areas for each type of sub-document.
2. **OCR Processing:** PDF files are converted into images using pdf2image library, and OCR technology provided by pytesseract is applied to extract text from the images. By analyzing the text within predefined search areas, the tool identifies the type of sub-document present on each page.
3. **Sub-document Separation:** Based on the OCR results, the tool determines the boundaries of each sub-document and separates them into individual PDF files. The output files are organized according to the defendant ID and the type of sub-document.
4. **User Interface:** The project includes a graphical user interface (GUI) to provide users with a user-friendly interface for configuring the tool, selecting input and output directories, and initiating the document separation process.
5. **Error Handling and Logging:** Robust error handling mechanisms are implemented to manage potential errors during the processing of PDF files. Additionally, logging functionality captures relevant information about the processing steps, aiding in debugging and tracking execution.

The OCR Project for Hackathon aims to streamline the process of managing combined PDF documents for legal firms, improving efficiency and accuracy in document management and submission. The tool's scalability and adaptability allow it to handle various types of sub-documents and configurations, making it a valuable asset for legal professionals dealing with large volumes of legal documents.

DESIGN DETAILS

System Design:

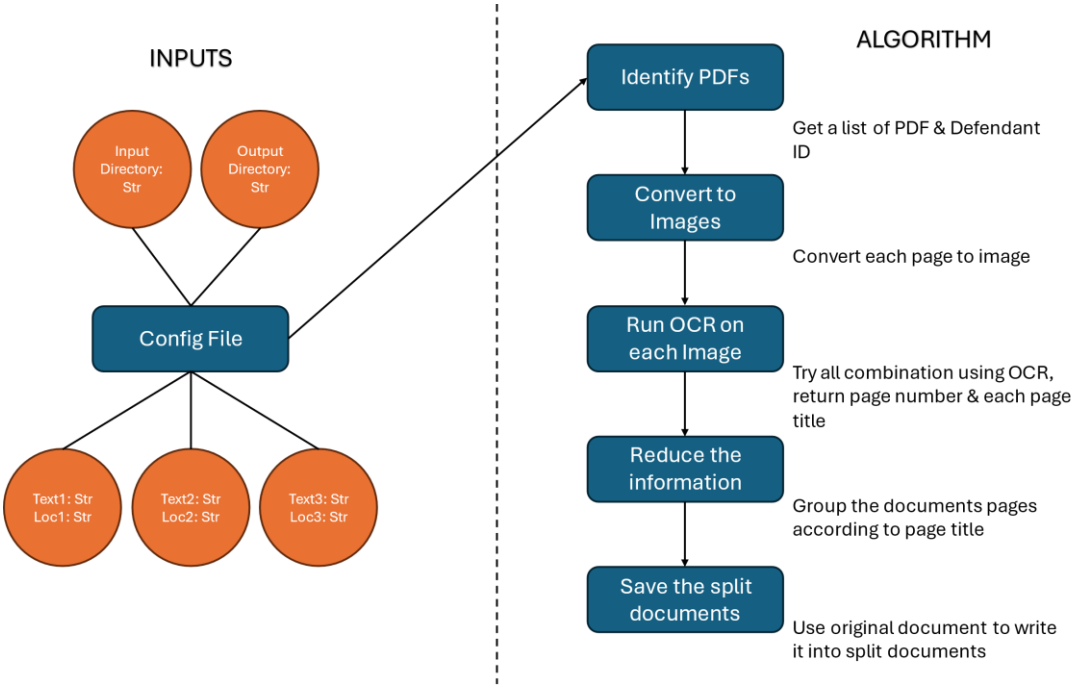


Figure 1: System Design

IMPLEMENTATION

The implementation of the OCR Project for Hackathon involves several key components and functionalities aimed at automating the process of splitting combined PDF documents into individual sub-documents. Here's an overview of the implementation process:

- **Configuration:**

  The project utilizes a configuration file (`Sample.cfg`) to define the input and output directories, as well as the keywords and corresponding search areas for each type of sub-document.

  Configuration parameters such as input folder, output folder, OCR keywords, and search areas are defined within the configuration file.

```
[Files]
input = D:/Param/Firm1/Cases
output = D:/Param/Firm1/Out

[OCR]
text1 = COMPLAINT
loc1 = 668, 1112, 2740, 1390
text2 = SUMMONS
loc2 = 668, 1112, 2740, 1390
text3 = APPEARANCE
loc3 = 668, 1112, 2740, 1390
```

Figure 2: Sample Config File

- **Graphical User Interface:**

  The GUI provides users with a user-friendly interface for configuring the tool, selecting input and output directories, and initiating the document separation process. Components of the GUI include input and output folder selection, entry fields for OCR settings (keywords and search areas), and buttons to save configurations and initiate PDF splitting. If a config file already exists then we can skip filling the details and use that file instead.

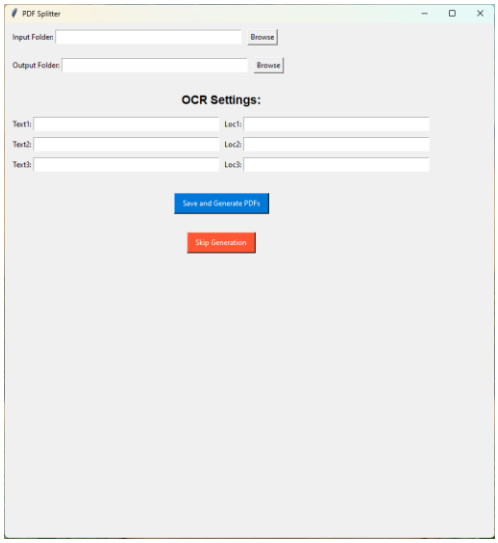


Figure 3: Graphical User Interface

- **PDF Splitting Algorithm:**

The project's primary objective revolves around processing amalgamated PDF documents to discern and isolate individual sub-documents. This is achieved by first converting PDF files into images using the pdf2image library, followed by leveraging OCR capabilities provided by pytesseract to extract textual content from these images. Through a meticulous analysis of the extracted text within predefined search regions, the tool effectively identifies the nature of each sub-document present on every page. Subsequently, sub-documents are segregated based on the OCR-derived insights, leading to the creation of distinct PDF files for each identified sub-document. The resulting output PDF files are meticulously organized, grouping them according to the defendant ID and the specific type of sub-document they represent.

- **Completion Notification:**

  Upon completion of the PDF splitting process, a completion GUI window notifies the user that the operation has been successfully executed.

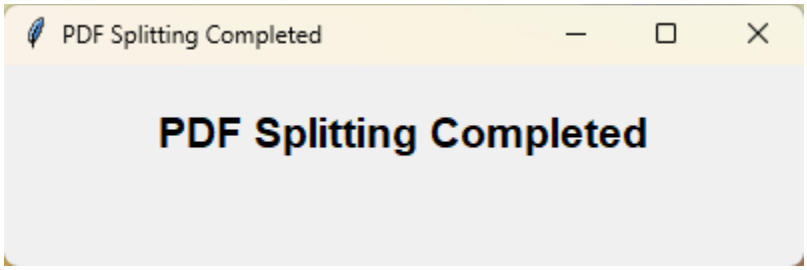


Figure 4: Completion Notification

TESTING

- **Multiple Input PDFs**:

  The program is tested with varying numbers of input PDF files, ranging from a single document to a large collection of combined PDFs. This ensures that the tool can efficiently process and split multiple documents without performance degradation or resource constraints.

- **Dynamic Keyword Configuration**:

  Scalability tests assess the program's ability to accommodate different numbers of keywords and search areas in the configuration file. The tool should adapt seamlessly to changes in keyword definitions and search parameters without compromising accuracy or efficiency.

- **Error Handling and Logging:**

  Robust error handling mechanisms are implemented to manage potential errors during the processing of PDF files. Logging functionality captures relevant information about the processing steps, aiding in debugging and tracking execution.

PERFORMANCE

The performance evaluation of the PDF splitting software involved running four distinct test cases, each with a different number of PDF files to split: one PDF, two PDFs, three PDFs, and four PDFs. These tests were conducted systematically and repeatedly to gauge the software's efficiency and scalability across varying workloads. Precise time-tracking mechanisms were employed to record the execution times accurately. The tests aimed to analyze the software's processing speed, scalability, and resource utilization, providing valuable insights for evaluating its overall performance and identifying potential areas for optimization.

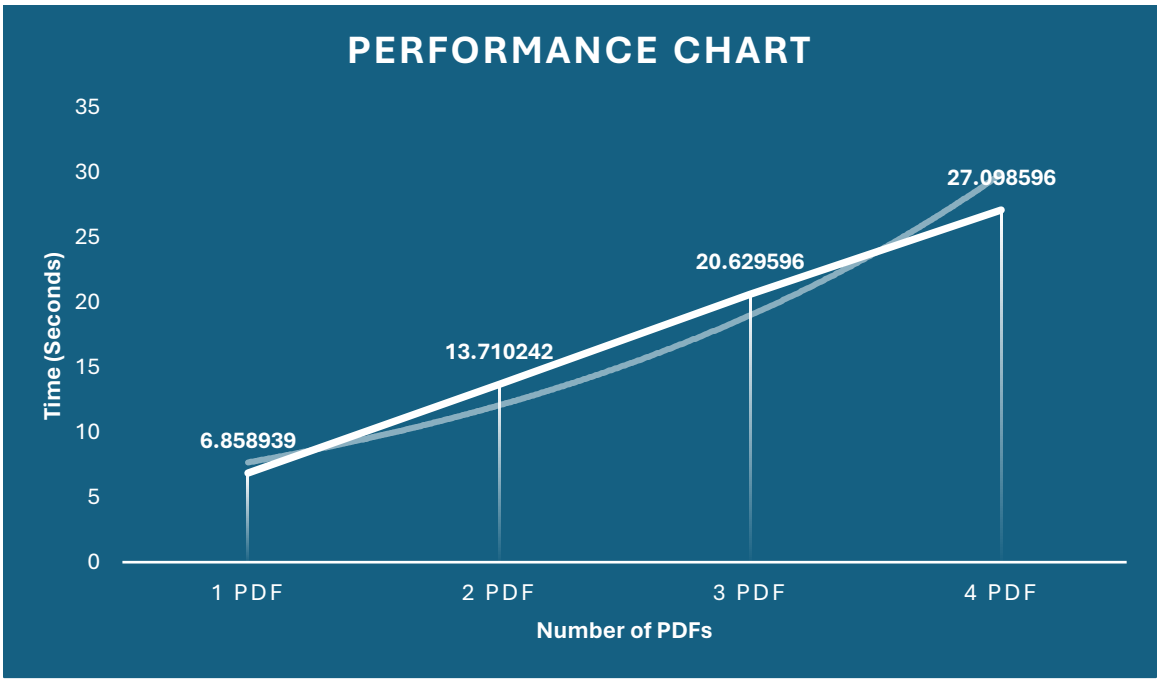
Performance Tests:

```
Folder exists! D:/Param/Coding/Python/splitpdf_exe/Firm1/Cases D:/Param/Coding/Python/splitpdf_exe/Firm1/Out
Number of PDF Files:  1
Elapsed time to process 1 PDFs: 6.858939 seconds
```

```
Folder exists! D:/Param/Coding/Python/splitpdf_exe/Firm1/Cases D:/Param/Coding/Python/splitpdf_exe/Firm1/Out
Number of PDF Files:  2
Elapsed time to process 2 PDFs: 13.710242 seconds
```

```
Folder exists! D:/Param/Coding/Python/splitpdf_exe/Firm1/Cases D:/Param/Coding/Python/splitpdf_exe/Firm1/Out
Number of PDF Files:  3
Elapsed time to process 3 PDFs: 20.629596 seconds
```

```
Folder exists! D:/Param/Coding/Python/splitpdf_exe/Firm1/Cases D:/Param/Coding/Python/splitpdf_exe/Firm1/Out
Number of PDF Files:  4
Elapsed time to process 4 PDFs: 27.098596 seconds
```



Analysis:

The performance evaluation of the PDF splitting software demonstrates a commendable level of efficiency and scalability. The software exhibits a linear increase in processing time with an increasing number of PDFs to split, showcasing its ability to handle varying workloads effectively. This scalability is a testament to the software's robust design and optimized algorithms.

Moreover, the execution times for splitting PDFs are well within acceptable limits, with each additional PDF adding only a modest amount of time to the overall processing duration. This efficient performance reflects the software's capability to process tasks swiftly and accurately, contributing to enhanced productivity and user satisfaction.

In conclusion, the performance analysis underscores the software's reliability, speed, and adaptability, making it a valuable tool for efficiently handling PDF splitting tasks across different scenarios and workloads.

NAVIGATING PROJECT ROADBLOCKS

1. **Not all dependencies could be installed through PIP:** To keep the number of dependencies minimum, we utilized a virtual environment to organize these dependencies. One of the challenges we encountered was with the pytesseract library, which is used to run OCR on images. It required the Tesseract-OCR engine to be externally attached. Therefore, we had to download and set up the Tesseract-OCR engine in the environment path so that Python could locate the dependency. Furthermore, the path changes when the Python file is converted to an executable. After some research, we discovered that the executable program stores additional files in a temporary directory in Windows. Similarly, the pdf2image library required the poppler package, which is only available through GitHub as a zip file. We also needed to add this package to the environment path when packaging it in an executable file. We resolved this issue using a similar approach to the above-mentioned issue."
2. **Human Error**: There are significant opportunities for humans to make mistakes while operating this software. During the coding phase, we made efforts to handle most situations where human errors could occur. For example, when entering text in the configuration file, humans may include unnecessary characters such as symbols or brackets & upper- or lower-case characters. To address this, the program is designed to strip the string and extract only the necessary information. Additionally, mistakes can also occur in input and output folder paths. To mitigate this,

the program verifies if the paths actually exist before executing the code, thereby reducing the chances of errors.

3. **DPI Settings**: One noteworthy challenge encountered was related to the DPI settings during the PDF to image conversion process. The converter defaulted to a resolution of 200 DPI, yet the coordinates provided in the configuration file were based on a 400 DPI setting. To resolve this discrepancy, I undertook a series of experiments where I attempted to plot the bounding boxes on the images and compared them to determine the actual DPI being utilized for coordinate calculation. After several trial-and-error iterations, it became evident that the sample coordinates were indeed generated using a 400 DPI setting.

LIMITATION & FUTURE IMPROVEMENT

Limitations:

1. The user needs to know the bounding coordinates for all the document titles.
2. Not tested on Mac OS. Unfortunately, our team did not have one.
3. Limited configurations available.

Future Improvements:

1. This algorithm could be hosted on a webserver with an easy-to-use UI. This will eliminate the need to download the standalone application.
2. The configuration file can be eliminated but providing options in the GUI for user to input the bounding box.
3. Could be scaled using GCP to run the splitting in Google Cloud Functions and splitting the tasks in Mapper & Reducers. Basically, using Map-Reduce.

CONCLUSION

The LegalDoc Extractor project presents a robust solution to the challenges encountered by legal firms in managing combined PDF documents. By leveraging Optical Character Recognition (OCR) technology, this project automates the extraction and separation of sub-documents, such as complaints, summonses, and appearances, from amalgamated PDF files. Through meticulous analysis of keywords and search areas, the tool accurately identifies the boundaries of each sub-document, facilitating efficient organization and submission to legal proceedings.

The project's graphical user interface (GUI) provides users with an intuitive platform for configuring settings, selecting input/output directories, and initiating the document separation process. Additionally, robust error handling mechanisms and logging functionality ensure reliability and traceability throughout the processing steps.

Performance testing demonstrates the software's efficiency and scalability, with execution times remaining within acceptable limits even when processing multiple PDF files. This efficiency, coupled with adaptability to varying workloads and configurations, underscores the tool's reliability and suitability for handling large volumes of legal documents.

Despite encountering challenges such as dependency management and DPI settings discrepancies, the project team successfully navigated these roadblocks, implementing effective solutions to ensure smooth functionality.

In conclusion, the LegalDoc Extractor project represents a significant advancement in document management for legal professionals, offering a streamlined and automated approach to PDF splitting. With potential future improvements such as web server hosting and enhanced user interfaces, this tool holds promise for further optimization and wider adoption within the legal community.

REFERENCES

- Pyinstaller: https://stackoverflow.com/questions/56210408/location-of-the-added-files-after-the-executable-file-is-generated-by-pyinstalle
- Pytesseract installation Windows: https://stackoverflow.com/questions/56210408/location-of-the-added-files-after-the-executable-file-is-generated-by-pyinstalle
- PyPDF2: https://pypdf2.readthedocs.io/en/3.x/
- pdf2image: https://pdf2image.readthedocs.io/en/latest/index.html
- Path setting in a executable file: python - Location of the added files after the executable file is generated by pyinstaller build - Stack Overflow

PROCESS TO RUN THE SCRIPT

There are 2 options to run the script.

Option 1 – Standalone Executable:

1. The user can download the standalone application form the following link. https://drive.google.com/file/d/1FYaWT4SJ4h_lr9fgtr2T8HEbVI6ZmDNe/view?usp=drive_link
2. After extracting the zip file, the user can just run the executable file, generate the config file according to his/her need, mention the input and output folders & click on save & generate pdf's.
3. The output pdfs will be available in output folder.

Option 2:

1. Clone the github repository: parampatil/splitpdf_exe (github.com)
2. Install the dependencies required in the environment using the requirements.txt file.
3. Run the splitpdf.py file to get the same result as standalone application.

The only constraints for this application to run in standalone executable are:

1. Need the splitpdf.exe file.
2. Need an input directory & output directory.