Text Extraction from Images, PDF Files, and Speech Data

Penelope Prochnow

I. Introduction

The aim of this project is to create an algorithm that surpasses the current accuracy rate for the most widely adapted and refined image, PDF, and speech data text extraction implementations. Through this adaptation there were elements devised and mutated with alternative closely relating algorithms to create a product with increased accuracy. This report will highlight the trial and error of various methods conducted, which ultimately lead to the final algorithms.

II. PDF Interpreter

A. *Image to Text*

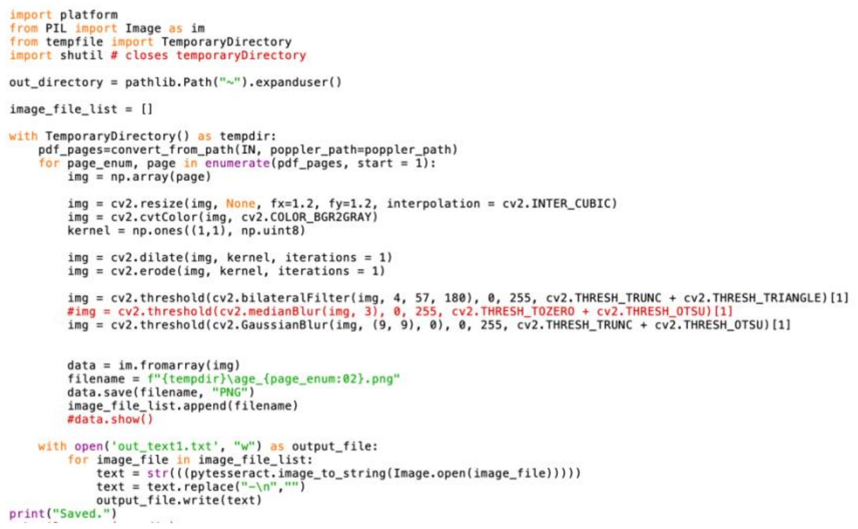
After installing the pre-established files included in the project description, there were minimal alterations taken to implement the image function. The image function scans images that contain text using pytesseract image to string extraction and translates to output text and has a text to speech capability using gTTS, a Google Speech package. The methods contained in this function aided the creation of the subsequent PDF interpretations containing images and the implementations of translating text to speech.

B. *Text PDFs*

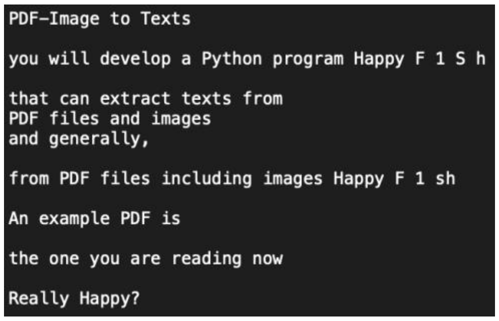
As a starting point for creating the PDF Interpreter implementation, there were two pre-established packages that resulted in the highest accuracy and created an elegant .txt file interpretation of plain text PDF files. These two packages were 'fitz' and 'pdfplumber.' While these packages created promising output for PDFs that contained only text, they were altogether inefficient and inadequate when testing with the three-page 'Programming-Project-2.pdf' file. The complications from these packages arose at the images on the third page of this PDF, not only did the text not scrape properly from the images but these packages are incomprehensible for PDFs containing many columns.

Initially, one approach that was taken to overcome this challenge was to locate the images and interpret them separately through an image extraction process. This approach proved to be laborious as the images contained in the third page appeared to be encoded in the document and became corrupt when transferring to JPEG image scanning methods. Due to these setbacks, the implementation of this approach came to a halt and another approach was adapted. While adapting the usage of the 'pdfplumber' package, I came across an approach to using OpenCV as a method of reading text from images. Although using OpenCV in adaptation with 'pdfplumber' was altogether disappointing, the method gave birth to another idea. The ulterior approach was to transform each page of the PDF to an image to utilize OCR capabilities. This idea is widely adapted and used across many interfaces. Through many sessions and days of trial and error, a mediocre algorithm was created. This algorithm would take a PDF file as input and immediately scan each page to create an image using pdf2image. The images were then processed through resizing, transforming to greyscale, thresholding through various gradient cv2 methods, structuring a kernel of each page to dilate and erode images, then each image was saved to a temporary directory. From the temporary directory, images were read back in and converted to text with Pytesseract's image to string function. From there text was created. The text output from the OCRed pages was altogether disappointing. This text could be altered and improved by adjusting the parameters of the various pre-processing techniques.

** Reference stackoverflow article used for processing



```python
import platform
from PIL import Image as im
from tempfile import TemporaryDirectory
import shutil # closes temporaryDirectory

out_directory = pathlib.Path("~").expanduser()

image_file_list = []

with TemporaryDirectory() as tempdir:
    pdf_pages=convert_from_path(IN, poppler_path=poppler_path)
    for page_enum, page in enumerate(pdf_pages, start = 1):
        img = np.array(page)

        img = cv2.resize(img, None, fx=1.2, fy=1.2, interpolation = cv2.INTER_CUBIC)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        kernel = np.ones((1,1), np.uint8)

        img = cv2.dilate(img, kernel, iterations = 1)
        img = cv2.erode(img, kernel, iterations = 1)

        img = cv2.threshold(cv2.bilateralFilter(img, 4, 57, 180), 0, 255, cv2.THRESH_TRUNC + cv2.THRESH_TRIANGLE)[1]
        #img = cv2.threshold(cv2.medianBlur(img, 3), 0, 255, cv2.THRESH_TOZERO + cv2.THRESH_OTSU)[1]
        img = cv2.threshold(cv2.GaussianBlur(img, (9, 9), 0), 0, 255, cv2.THRESH_TRUNC + cv2.THRESH_OTSU)[1]


        data = im.fromarray(img)
        filename = f"{tempdir}\age_{page_enum:02}.png"
        data.save(filename, "PNG")
        image_file_list.append(filename)
        #data.show()

    with open('out_text1.txt', "w") as output_file:
        for image_file in image_file_list:
            text = str(((pytesseract.image_to_string(Image.open(image_file)))))
            text = text.replace("-\n","")
            output_file.write(text)
print("Saved.")
```



```
PDF-Image to Texts

you will develop a Python program Happy F 1 S h

that can extract texts from
PDF files and images
and generally,

from PDF files including images Happy F 1 sh

An example PDF is

the one you are reading now

Really Happy?
```

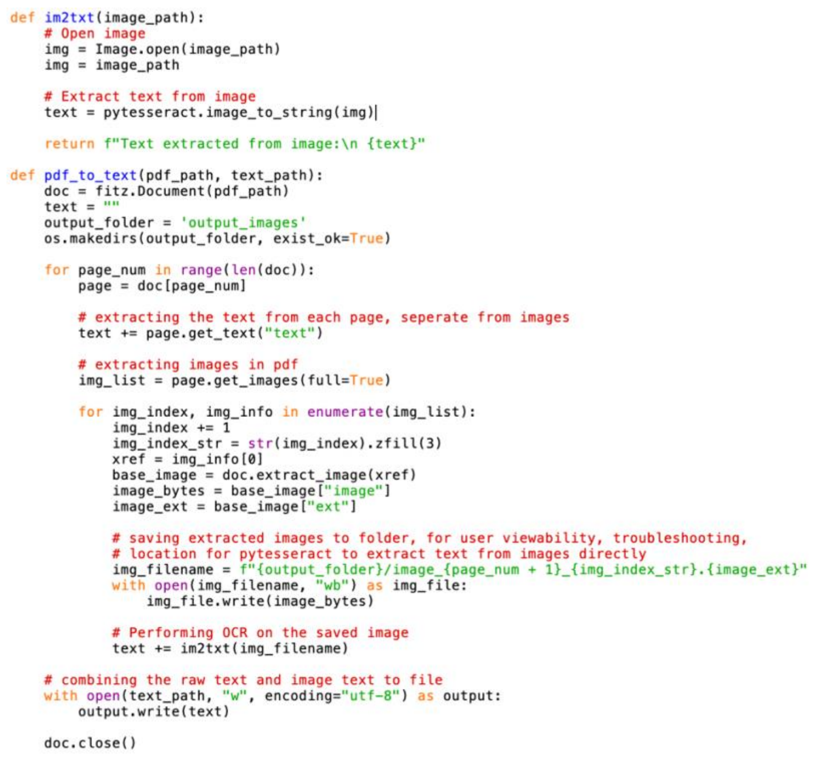
Figure 1. OCR implementations to translate PDF file through pdf2image

In the end, as shown in Figure 1, even the most fine-tuned pre-processing techniques resulted in a poor text translation. Ultimately, the most complex lines included the second "Happy Fish" and the "Really Happy?" lines. It would seem as one line got closer to being perfected the other line would stray further from a proper translation. This translation is a sound method for most applications and results in an improvement over basic pdf interpretation packages as all other elements of the pdf were properly translated and the text appears in proper spacing coordination with the original pdf file.

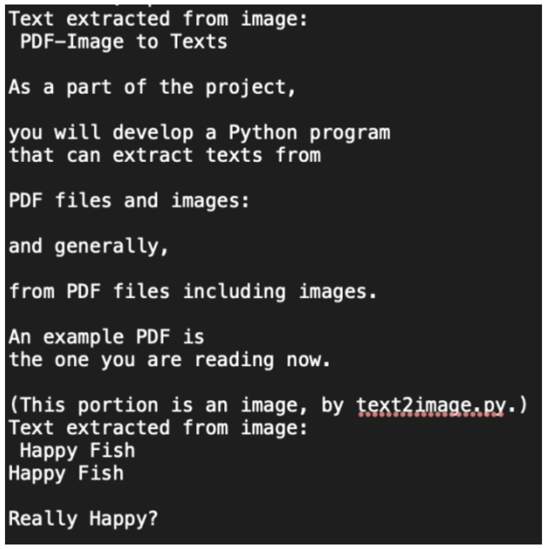
C. Adaptations taken for the 'Programming-Project-2.pdf' File

** Start with the roundabout final function used

While compiling the three aforementioned methods the results for this use case were not optimal. After reiterating through each of the attempts, troubleshooting the issues of each implementation was segmented into a finer revision. Troubleshooting the implementation of the 'fitz' package method led to an alternative discovery after having worked with and learning the basis of image processing in python through the trial and error shown in Figure 1. Segmenting the code into fractions that worked well, plain text interpretation, and segments that did not work well, interpretation of images, and thoroughly examining routes that could be taken to better each limitation led to a furthered development of image extraction. At which point, a development was made as it was discovered that in the PyMuPDF 'fitz' package there is a function to get images from pdf files. After getting the image information into a list, each image could then be extracted individually, and text could be taken from the images using the pytesseract image to string function as used in the image to text function. This final development is shown in Figure 2.



```
def im2txt(image_path):
    # Open image
    img = Image.open(image_path)
    img = image_path

    # Extract text from image
    text = pytesseract.image_to_string(img)

    return f"Text extracted from image:\n {text}"

def pdf_to_text(pdf_path, text_path):
    doc = fitz.Document(pdf_path)
    text = ""
    output_folder = 'output_images'
    os.makedirs(output_folder, exist_ok=True)

    for page_num in range(len(doc)):
        page = doc[page_num]

        # extracting the text from each page, seperate from images
        text += page.get_text("text")

        # extracting images in pdf
        img_list = page.get_images(full=True)

        for img_index, img_info in enumerate(img_list):
            img_index += 1
            img_index_str = str(img_index).zfill(3)
            xref = img_info[0]
            base_image = doc.extract_image(xref)
            image_bytes = base_image["image"]
            image_ext = base_image["ext"]

            # saving extracted images to folder, for user viewability, troubleshooting,
            # location for pytesseract to extract text from images directly
            img_filename = f"{output_folder}/image_{page_num + 1}_{img_index_str}.{image_ext}"
            with open(img_filename, "wb") as img_file:
                img_file.write(image_bytes)

            # Performing OCR on the saved image
            text += im2txt(img_filename)

    # combining the raw text and image text to file
    with open(text_path, "w", encoding="utf-8") as output:
        output.write(text)

    doc.close()
```



```
Text extracted from image:
 PDF-Image to Texts

As a part of the project,

you will develop a Python program
that can extract texts from

PDF files and images:

and generally,

from PDF files including images.

An example PDF is
the one you are reading now.

(This portion is an image, by text2image.py.)
Text extracted from image:
 Happy Fish
Happy Fish

Really Happy?
```

Figure 2. Optimized Implementation of PyMuPDF capabilities coupled with Image Extraction

As presented in the Figure 2 code extract, the images are individually saved to a “output_folder” directory, the purpose suffices as a troubleshooting method as well as a user clarity check to assure that each image was properly extract and they can view exactly which elements are appearing in the text extraction. Although this code is highly efficient to translate all elements of any pdf, it does not place the text in the corresponding image location from the pdf in the text file. This concept could be furthered and better adapted to any use case through PyMuPDF page.get_image_rects package. This package will obtain the data corresponding to the coordinate points of each image on each page. This implementation would have been adapted into this program in future adaptations with time allowance. However, as it currently stands, depending on user preference, the images can be explicitly listed as images in the pdf at the end of the pdf text extraction. As this is my preference, I have left the program to work in such way for now. Alternative tests are included below with original pdf images and output text.

III. Speech and .Wav Files

A. *Translation from .wav file input*

B. *Translation from microphone input*