Structure-aware OCR

Sofia Santos (A89615)
Mestrado em Engenharia Informática
Universidade do Minho
Ano Letivo 2022/23

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Structure-aware OCR

Abstract

...

I. Introduction

Up until the end of the 20th century, if someone needed to convert a physical document into a digital format, the easiest methods to do so would be to either transcribe the document manually, which takes a lot of time and effort, or, alternatively, scan or take a photograph of the document. This second option is a lot easier and faster, but it doesn't allow us to directly edit or manipulate the document's contents.

Throughout the last couple of years, as the fields of computer vision and machine learning keep advancing, Optical Character Recognition, or OCR, has become a common solution for this kind of problem. Despite some version of OCR existing as early as 1914 [2], OCR tools only became popular and accessible from the early 2000s onward, mostly through web services like WebOCR [20]. Using an OCR tool, one is capable of creating a digital copy of a scanned physical document as a textual file, instead of as an image [21].

However, most OCR tools are limited to a textual output. In other words, if an image contains text with varying font sizes, indentation or colors, inputting this image into an OCR tool will generate a string of pure text without any of that ”extra” information.

Although this is perfectly acceptable for many use cases, there are scenarios where it would be useful for the converted text to maintain some of its original structure. For example, if scanning a newspaper page, one would want to keep each article as a separate file or section within a file.

The main goal of this report is to provide a general overview of existing OCR tools, their strengths and limitations, and to showcase a prototype tool capable of performing OCR while keeping the original text's structure as intact as possible.

II. State of the Art

A. *OCR Tools*

Most available OCR tools can be described as ”basic”, and a simple online search for the term ”online OCR” will reveal hundreds of websites that perform this type of OCR. Using these tools, a user can upload a file, usually an image, and the tool will process it and return the string of text it found on the image. These tools offer very little customization, as they are meant to be used by anyone, regardless of previous knowledge in character recognition software, with most of them only allowing users to select the language in which the text is written and if the output file should be a text file, a PDF file or a Microsoft Word file, for example [11][14].

One of these tools that stands out from the rest is Google Lens [16]. This tool, available on mobile devices, allows the user to use their smartphone's camera to capture an image containing text. After taking this picture, the user can interactively select and copy the text recognized in the picture. Google Lens also excels at recognizing handwritten text and other kinds of text that traditional tools struggle with [3], mainly due to Google's investment in Machine Learning algorithms.

Some tools also offer the possibility to create a special PDF file containing the original scanned document, but with an invisible text layer above the images [7]. This way, users can still see the original document but also select the text found in it, as if it were a regular PDF file. This method is a possible solution to structure-aware OCR, and is used by search engines in order to find text in PDF files composed only of images [6]. However, the text itself is not structured, it only appears to be, thus it is not the best solution for this problem.

Table II-A contains a list of some popular OCR tools [5] and the main features supported by each. This list only includes tools that can be used for free. Paid tools tend to offer most, if not all of these features, but the author did not find any additional feature offered by a paid tool that was not also part of a free tool. By default, all of these tools support textual output.

Tool	Recognize text in Portuguese	Auto-rotation	Table recognition	Restrict OCR to part of file/image	Create searchable PDF	Create DOCX file
OCRSpace [13]	X	X	X		X	
FreeOCR [10]						X
OnlineOCR [11]	X	X	X		X	X
Google Lens [16]	X	X		X		
Text Grab [19]				X		

It's worth noting that different tools have different use cases, and may therefore appear to have fewer features. For example, Google Lens uses a smartphone's camera to capture and detect text, unlike the other tools, and Text Grab allows the user to take a screenshot of their computer and immediately recognize text from the captured image, so it makes sense that these tools don't focus on features like creating searchable PDFs.

B. OCR Engines

Although the aforementioned tools perform optical character recognition, they are merely wrappers for OCR engines. An OCR engine is the software responsible for recognizing the characters in an image and converting them to text [1]. Applications like Text Grab use these engines to bring OCR functionality to their tools, and are meant to be more user-friendly and intuitive than purely using an engine.

One of the most widely used OCR engines is called Tesseract. It can be used directly via command line, through a 3rd party tool or by using an API written for a programming language [17]. Examples of Tesseract APIs include Python-tesseract [15] or Tesseract.js [18], which can be used by programmers to create applications with OCR functionalities.

Tesseract supports dozens of languages [12] and offers a wide assortment of options [17]. One of these is related to page segmentation, and affects how Tesseract detects text in an image. By default, Tesseract tries to divide the original picture into segments, which can be titles or columns of text, for example. Then, it performs OCR on each of those segments. If the output format is a textual format, this will mean that the final text will be split into segments. However, each one of these segments has the same font and size, making it impossible to distinguish between a title and a regular sentence, for example, without context. This default behavior can be changed, if one desires. For example, Tesseract can interpret the entire picture as a single word, and will try to read it as such.

Additional options include output formatting, which can be a PDF file, a text file or an hOCR file (HTML compatible file, with additional information about the original text's structure), for example, or an option to let Tesseract detect words from a user-provided list.

Since Tesseract is just an OCR engine with no image processing functions, many users feel the need to apply pre-processing effects to their images before inputting them in Tesseract, in order to improve the text detection [4]. These may include changes in contrast, brightness, size or even converting the image to black-and-white, to avoid issues with colored text/backgrounds.

III. Problem Definition

This paper proposes the creation of a program capable of performing Optical Character Recognition on a file, while maintaining its original structure. For this purpose, a prototype was developed, in order to showcase the benefits of this approach to OCR. Unlike already existing solutions that convert a file to a PDF file with invisible text, this tool creates a purely textual file. In order to have a structured textual file, this file uses Markdown [8], a markup language [9].

This tool was developed in the Python programming language and uses the Tesseract OCR engine. This decisions stems from the author's experience with Python, the robustness of Tesseract and its Python library, `pytesseract` and the ability to easily and rapidly create a command-line or graphical application with Python.

The application focuses primarily on detecting text from newspapers or magazine pages, essentially pages with one or more articles. In practice, it can scan any type of document, but it might not be able to preserve its structure.

The tool has a Graphical User Interface (GUI), developed using the Kivy framework, where a user can select a source image containing the article(s) they want to convert. Then, they can select each article's position within the image. This allows the tool to focus less on discerning between different articles and more on formatting each one independently. After this step, the tool performs OCR on each image section and display the results to the user, who can then save the detected text in a Markdown file.

The system follows an architecture similar to the one shown in Figure 1.

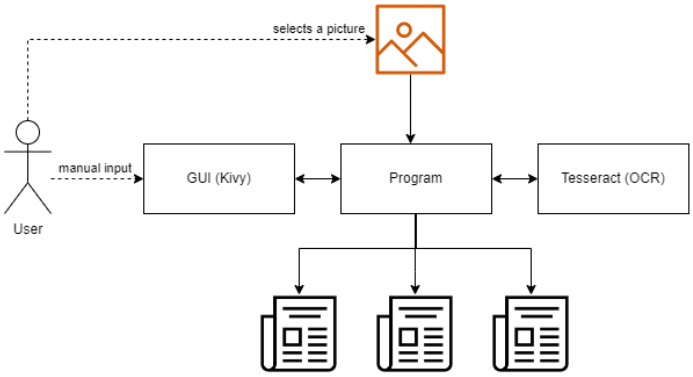


Fig. 1. System architecture.

The main program receives an image and produces a file containing the articles found in the picture. The OCR is performed by Tesseract and the GUI allows the user to interact with the program.

The program follows a series of steps to convert an image into articles. These steps, which have been already mentioned above, can be seen in Figure 2.

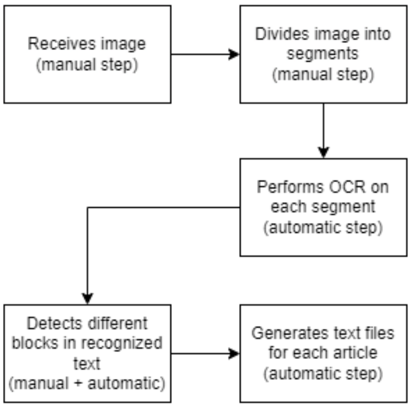

Receives image (manual step)
Divides image into segments (manual step)
Performs OCR on each segment (automatic step)
Detects different blocks in recognized text (manual + automatic)
Generates text files for each article (automatic step)

Fig. 2. Program steps.

Some steps, like performing OCR on an image, are fully automatic. In other words, they can be performed without user interaction. On the other hand, dividing an image into segments is a fully manual step, since automating this step would require an advanced algorithm, due to the wide variety of possible article arrangements in newspapers and magazines, and would likely require a long time to create something functional.

IV. Development

The first step in development consisted of creating a simple program capable of converting an image into an intermediate representation (IR). This IR holds information about the text and additional context about the OCR process, for example, the dimension of the bounding box where the text was located. The pytesseract module already has a method `image_to_data` which can convert an image into a data structure (e.g., TSV file, dictionary or `pandas` dataframe). However, this data structure contains more data than necessary, so it became necessary to simplify it and make it easier to navigate and access. Hence, this information is filtered and converted into instances of Python classes.

Four classes were created for this purpose: Article, Block, Paragraph and Line. The last three classes simply mirror the information returned by pytesseract, which splits text into pages, blocks, paragraphs, lines and words. Since this tool will only work with individual pages, the first category is not needed, and the fifth category, words, is represented as a list in the Line class, since there is no need to store extra information about each word. Besides a list of words, the Line class also contains information about its height. This information is useful because Tesseract might sometimes classify two lines as being part of different paragraphs, or two paragraphs as being part of different blocks, which is not always the case. If we assume that lines of the same paragraph and paragraphs of the same block should have text of the same size, by storing each line's height, we can compare the height of different lines and paragraphs, and infer if two lines/paragraphs should be part of the same paragraph/block. For this purpose, the Paraphaph class contains a `get_line_height` method, which returns the average height of all its lines. Similarly, the Block class contains an equal method, but one which returns the average height of its paragraphs' lines.

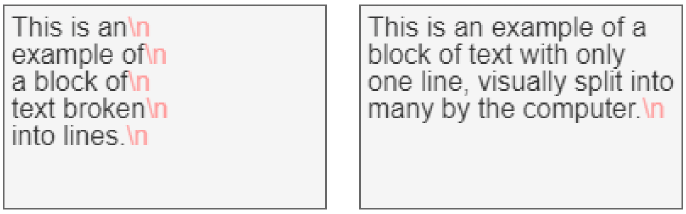
After this step was successfully complete, the next stage of development consisted of developing a simple user interface.

When opening the application, the interface contains a window where the user can select an image to be analyzed. After selecting an image, the next window of the application allows the user to select the articles present in the image by drawing rectangles over them. Optionally, users can select areas for the program to ignore. For example, if the original articles contain images with text, one could select the image as an area to be excluded, so that Tesseract won't detect the text in the image.

After selecting the articles, the tool divides the image into segments, based on the drawn rectangles, and feeds those segments to Tesseract, which proceeds to detect the text within those images. Then, the tool creates instances of the Article class, each containing the text from a different article. When creating these instances, the program perform an optimization step, where it tries to group together different blocks or paragraphs, based on how they end. For example, if a paragraph ends with a hyphen and the next one begins with a lowercase letter, it probably means that Tesseract failed to detect both lines as part of the same paragraph, and the program will try to automatically fix that mistake.

In addition, the tool also tries to detect an article's title. To do this, it analyses every block and finds the one with the biggest font size, which it then tags as the article's title. In the final file, this is represented with a pound sign (#) before the block, which is used in Markdown to define a heading. If every block has a similar font size, the tool doesn't do anything, to avoid mislabeling a block, although this can still happen if the block with the biggest font size is not the article's title.

Another feature offered by the application, which is not present in most OCR tools, is the possibility to include or exclude line breaks inside each paragraph. Typical OCR tools, when scanning a document, will preserve each line in the original image as a separate line in the final text. However, in the original document, the text is only split into multiple lines due to the limited size of the paper. On a computer screen, which is typically wider than a piece of paper, it might not make sense to preserve the original line breaks. Therefore, after scanning a picture, there's an option in the application to remove the original line breaks, and keep each paragraph as a single continuous line. This behavior mimics computer text editors, like Microsoft Word, where the line breaks are artificially created by the software in order to make the text fit in the screen, while the text remains stored as a single line. Figure 3 illustrates this difference, with the newline characters highlighted in red.



This is an\n
example of\n
a block of\n
text broken\n
into lines.\n

This is an example of a
block of text with only
one line, visually split into
many by the computer.\n

Fig. 3. Distinct options for line breaks in output file. The first option mimics the original image.

Bibliography

[1] *What is OCR (Optical Character Recognition)?* 1978. URL: https://aws.amazon.com/what-is/ocr/.

[2] Herbert F. Schantz. *The history of OCR, optical character recognition*. Recognition Technologies Users Association, 1982.

[3] James Vincent. *Google lens can now copy and paste handwritten notes to your computer*. May 2020. URL: https://www.theverge.com/2020/5/7/21250556/google-lens-copy-paste-handwritten-notes-computer-phone-ios-android.

[4] Kaan Kuguoglu. *How to use image preprocessing to improve the accuracy of Tesseract*. July 2021. URL: https://towardsdatascience.com/getting-started-with-tesseract-part-ii-f7f9a0899b3f.

[5] *12+ best free OCR software for windows [2022 updated list]*. Sept. 2022. URL: https://www.softwaretestinghelp.com/ocr-software-for-pc/. (accessed: 2022-10-19).

[6] *Google answers whether it's better to OCR text in pdfs or not*. Aug. 2022. URL: https://iloveseo.com/seo/google-answers-whether-its-better-to-ocr-text-in-pdfs-or-not/.

[7] Trey Harris. *Converting a scanned document into a compressed, searchable PDF with redactions*. Sept. 2022. URL: https://medium.com/@treyharris/converting-a-scanned-document-into-a-compressed-searchable-pdf-with-redactions-63f61c34fe4c.

[8] URL: https://www.markdownguide.org/.

[9] *An introduction to markup*. URL: https://port.sas.ac.uk/mod/book/view.php?id=568&amp;chapterid=336.

[10] *FreeOCR*. URL: http://www.paperfile.net/.

[11] *Image to text converter using OCR Online*. URL: https://www.onlineocr.net/.

[12] *Languages supported in different versions of Tesseract*. URL: https://tesseract-ocr.github.io/tessdoc/Data-Files-in-different-versions.html.

[13] *OCRSpace*. URL: https://ocr.space/.

[14] *Online OCR - free OCR PDF Document Scanner & converter*. URL: https://www.sodapdf.com/ocr-pdf/.

[15] *Pytesseract*. URL: https://pypi.org/project/pytesseract/.

[16] *Search what you see*. URL: https://lens.google/.

[17] *Tesseract user manual*. URL: https://tesseract-ocr.github.io/tessdoc/.

[18] *Tesseract.js: Pure javascript OCR for 100 languages!* URL: https://tesseract.projectnaptha.com/.

[19] TheJoeFin. *TheJoeFin/text-grab: Use OCR in Windows 10 quickly and easily with text grab. with optional background process and popups*. URL: https://github.com/TheJoeFin/Text-Grab.

[20] *WebOCR & OnlineOCR*. URL: http://www.expervision.com/ocr-software/webocr-onlineocr.

[21] *What is optical character recognition (OCR)?* URL: https://www.ibm.com/cloud/blog/optical-character-recognition.