Workflow for processing digitized documents at the United Nations

João Maria Prieto Dantas Vizoso

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. José Luís Brinquete Borbinha

September 2021

Acknowledgments

I want to thank my supervisor, Prof. José Borbinha for all the knowledge and help provided throughout the realization of this dissertation, as well as the Archives and Records Management Service of the United Nations for their collaboration and feedback given throughout the development of the final solution.
I would also like to thank my family and friends, especially my parents, João and Elsa, and my sister, Beatriz, for always being there for me, not only during this dissertation, but throughout my life.
Finally, a special thanks to Constança for the great support given in these last months.

Resumo

Este trabalho aborda o problema da digitalização de documentos sob a responsabilidade do Archives and Records Management Service das Nações Unidas. Estas colecções são únicas, reflectindo a história da organização desde 1945, o que motiva a sua digitalização, para que possam ser preservadas e mais facilmente acedidas. Até agora, mais de 10 milhões de imagens e mais de 5 TB de dados foram criados. Uma grande parte já está disponível online, mas ainda há um longo caminho a percorrer para digitalizar o que ainda está em papel e para isso o ideal seria ter um workflow automatizado para processar as imagens digitalizadas, produzindo conteúdo para um conveniente acesso digital. Assim sendo, esta tese visa construir esse workflow automatizado para processar imagens digitalizadas dos documentos, de forma a que possam ser publicadas e acedidas online.

Palavras-chave: Documento, PDF, OCR, Workflow, Arquivo, Imagem

Abstract

This work addresses the problem of the digitization of paper documents under the responsibility of the Archives and Records Management Service of the United Nations. Those collections are unique, reflecting the history of the organization since 1945, which motivates its digitization, to promote its easy access. So far, more than 10 million images and more than 5TB of data was created, a large part already available online, but there is still a long way ahead to either digitize what is still in paper and to have an automated workflow to process the digitized images and produce objects for convenient digital access. This thesis therefore aims to build an effective automated workflow to process those digitized images for optimal online publishing and search.

Keywords: Document, PDF, OCR, Workflow, Archive, Image

Contents

Acknowledgments . . . iii

Resumo . . . v

Abstract . . . vii

List of Figures . . . xiii

List of Acronyms . . . xv

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Objectives . . . 2

1.3 Structure of the documment . . . 2

2 Relevant file formats **3**

2.1 Tagged Image File Format (TIFF) . . . 3

2.2 Portable Document Format (PDF) . . . 5

2.3 JavaScript Object Notation (JSON) . . . 8

3 Optical Character Recognition **9**

3.1 OCR review . . . 9

3.2 OCR reference architecture . . . 10

3.3 Examples of existing software . . . 12

3.3.1 Tesseract . . . 12

3.3.2 OCRopus . . . 14

3.3.3 Adobe Acrobat Pro DC . . . 15

3.4 hOCR data schema . . . 15

4 Problem analysis and proposed solution **18**

4.1 Existing solution . . . 18

4.2 Requirements . . . 20

4.3 Proof-of-concept . . . 21

4.3.1 Technologies used . . . 22

4.3.2 Development . . . 22

4.3.3 Execution . . . 23

4.4 Final version . . . 24
4.4.1 Development . . . 25
4.4.2 Execution . . . 27
4.4.3 Deployment . . . 35

5 Demonstration and assessment 37
5.1 Demonstration . . . 37
5.1.1 Split a multi-page TIFF image . . . 37
5.1.2 Pre-process an image - Simple . . . 39
5.1.3 Pre-process an image - Additional . . . 39
5.1.4 Segment an image . . . 40
5.1.5 Perform second OCR run on the segmented image . . . 42
5.1.6 Compare results from OCR . . . 42
5.1.7 Merge results . . . 44
5.1.8 Continue from interruption . . . 45
5.1.9 Forced start . . . 45
5.1.10 Skip comparison . . . 46
5.2 Assessment . . . 46
5.2.1 Real-world environment . . . 47
5.2.2 Development environment . . . 48

6 Conclusions 54
6.1 Main challenges and lessons learned . . . 54
6.2 Discussion of the results . . . 55
6.3 Future Work . . . 56

Bibliography 57

A Example of hOCR file 59

B Available parameters 61

C Individual results of efficiency assessment 63

D Individual results of effectiveness assessment 68

List of Tables

3.1 Different OCR system's architectures, according to Ray Smith [11] 11

5.1 Files used in the assessment 49
5.2 Total processing time using three different parametrizations 50
5.3 CER (%) and WER (%) of the three experiments 53

List of Figures

2.1 General structure of a TIFF file [5] 4
2.2 Example of general structure of a PDF file, extracted from PDFXplorer software 6
2.3 PDF document structure [6] 7
2.4 Example of JSON objects and their visualization 8

3.1 Example of binarization 11
3.2 Example of deskewing 12
3.3 Main steps of an OCR reference architecture 12
3.4 Tesseract main steps 13
3.5 OCRopus main steps 14
3.6 Example of bounding box propery 16
3.7 Example of baseline property 17

4.1 Existing solution at ARMS 19
4.2 Resizing process 19
4.3 Renaming process 20
4.4 Proof-of-concept directory structure 22
4.5 Example of a JSON entry 23
4.6 Final version directory structure 26
4.7 Available segmentation methods on tesserocr 27
4.8 Example of a JSON entry 28
4.9 Example of a JSON entry 28
4.10 Prologue 29
4.11 Splitting process 30
4.12 Noise removal 31
4.13 Before dilation (left) and after (right) 31
4.14 Pre-processing process 32
4.15 Segmentation process 33
4.16 Example of an entry on *regions.JSON* file 33
4.17 OCR process 34
4.18 Comparison process 35

4.19 Merging process 36

5.1 Splitting task 38
5.2 Result of splitting task 38
5.3 Pre-processing task 39
5.4 Result of pre-processing task 39
5.5 Additional pre-processing task 40
5.6 Result of additional pre-processing task 40
5.7 Segmentation task 41
5.8 First result of segmentation task 41
5.9 Second result of segmentation task 41
5.10 Third result of segmentation task 42
5.11 Second OCR task 42
5.12 Result of second OCR task 43
5.13 Result from first OCR task 43
5.14 Result from second OCR task 44
5.15 Compare results from OCR task 44
5.16 Result of compare results from OCR task 44
5.17 Merging task 45
5.18 Result of merging task 45
5.19 Continue from interruption task 46
5.20 Forced start task 46
5.21 Skip comparison task 47
5.22 Average processing time using three different parametrizations 50
5.23 Image 1 used in the assessment 51
5.24 Image 2 used in the assessment 51
5.25 Image 3 used in the assessment 52

List of Acronyms

API	Application Programming Interface
ARMS	Archives and Records Management Service
CER	Character Error Rate
DPI	Dots Per Inch
GB	Gigabyte
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HMM	Hidden Markov Model
HTML	HyperText Markup Language
IFD	Image File Directory
IUPR	Image Understanding and Pattern Recognition
JSON	JavaScript Object Notation
LSTM	Long short-term memory
OCR	Optical Character Recognition
PDF	Portable Document Format
RAM	Random Access Memory
RNN	Recurrent neural network
TB	Terabyte
TIFF	Tagged Image File Format
TSV	Tab-separated values
UN	United Nations
USA	United States of America
WER	Word Error Rate

Chapter 1

Introduction

In the spring of 1945, it was held in San Francisco, USA, the United Nations Conference on International Organization. As a result of this convention, the Charter of the United Nations [1] was created, allowing the foundation of the organization known today as the United Nations (UN)[1]. Apart from this important document, many more documents were produced during that convention, making it necessary to create a documents' service unit, which primary and mostly exclusive job was to maintain custody of those documents.

In the present date, the Archives and Records Management Service (ARMS)[2] of the United Nations is the result of that service that was initiated 75 years ago, being responsible for the large number of documents produced during these years of United Nations existence [2]. With the rise of new technologies and globalization, it became almost mandatory to find a better way to preserve these documents (all of them are unique and some of the documents are very old and therefore very fragile).

As a solution, ARMS started to digitize and process some of the documents [3]. Most of these documents are classified for security reasons. However, some of them are declassified and made available to the general public from time to time. When that happens, ARMS uploads those documents into their public database, which is available to everyone with Internet access. So far, more than 10 million images and more than 5 TB of information have been made available, but the process to do this still takes a considerable amount of time and resources (which can be allocated to different tasks), and there are many more documents ready to be digitized, processed and uploaded.

1.1 Motivation

As mentioned above, the workflow currently being used by ARMS is not fully automated, taking up more time and resources than desired.

There are steps of the workflow that will always have to be handled by humans, such as handling the physical documents and digitize them. Nonetheless, all the document processing, for it to be ready to be uploaded in the public database can be performed automatically. Although the existing workflow already

[1] https://www.un.org/
[2] https://archives.un.org/

uses software to perform some of these steps, there are intermediate steps that still have to be done by a staff member. In addition, this software that is used for text recognition, does not always provide excellent results.
It is, therefore, necessary to build a solution that allows the work of processing digitized documents to be more efficient (consuming less time and resources) and more effective (producing better results and providing a better experience to the users that will have to deal with it).

1.2 Objectives

With the collaboration with ARMS, the main objective of this project is to propose a solution of an automated workflow to process the digitized images, taking into account the challenges presented above. The final solution has to be as automated as possible, consuming the least resources and providing the best result: a legible document similar to the original one, which can be uploaded to the digital archive of the ARMS and accessed by people all around the world.
One of the main focus of this dissertation is the Optical Character Recognition (OCR), which is the step of the workflow that is lacking a better solution. Other tasks, like PDF compression and improvement, or image pre-processing, although no less important, will have to remain for future improvements that may be made to this workflow. It is important to mention that the purpose was not to create a new OCR solution but to make use of the available ones, together with other tools, to come up with the best results possible.

1.3 Structure of the documment

This document is organized into six different chapters. Chapter 1 introduces this dissertation, with its motivation and primary objectives. Chapter 2 covers the basics of the file formats relevant to this thesis, namely the PDF, TIFF, and JSON. Chapter 3 covers some existing solutions for Optical Character Recognition (OCR) and the main steps of their functioning. Chapter 4 describes the problem tackled in this dissertation, presenting the existing solution used at ARMS and also explains what the proposed solution for the presented problem is. Chapter 5 is where the demonstration and evaluation of the proposed solution are made. Finally, Chapter 6 presents the main challenges while developing this same solution, as well as the conclusions that were reached.

Chapter 2

Relevant file formats

In this dissertation, it is assumed that the documents to be processed by the workflow are already scanned and in digital form. In ARMS, the format chosen to save the files is TIFF. Therefore, the input format of the workflow will be a TIFF image, and the output format will be a PDF file. Both these formats will be covered in this chapter. In addition, the JSON format that is relevant to the scope of this dissertation will also be mentioned.

2.1 Tagged Image File Format (TIFF)

Created in 1986 by Aldus Corporation as a standard method for storing black and white images originated by scanners, the Tagged Image File Format (TIFF) is one of the most used image file formats used nowadays, especially by graphic artists, photographers, or the publishing industry.
As its name implies, “Tagged” refers to the format’s file structure since tags are used to carry important information (such as size, resolution, applied image compression, etc.) to the program displaying the file. There are 70 different tag types, which allows a great level of flexibility when reading this type of file [4]. The ability to support a full range of image sizes, resolutions, and colors, the use of lossless compression techniques (allowing the files to maintain their resolution without loss of detail), and the fact that each file can contain several images are the biggest strengths of the TIFF and the reasons why it is still widely used. When it comes to weaknesses, the large file size (resulting from the use of lossless compression and the amounts of tags needed to transport the image data) can prevent their use when choosing what type of format to use.
In the specific context of this project, the Tagged Image File Format is the format used when the image is digitized and before its processing. When scanning the documents, ARMS uses a resolution of 400DPI, producing files with a larger size but ensuring a very good level of detail of the images, which is relevant when performing the OCR on the document.
About its structure, a TIFF file is composed by an header, an Image File Directory (IFD) and directory entries [5].
The header of a TIFF image is 8-byte long and points to an IFD. The image file directory is composed

by information about the image, and pointers to the actual image data. The information present in the header includes: the byte order used within the file, a number chosen to identify the file as a TIFF file (in this case is the number 42) and the offset of the first IFD.
The image file directories can be placed anywhere after the file header, and the reader just has to follow where the pointers point. These are composed by a 2-byte count of the number of directory entries, followed by a 12-byte field entries sequence and a 4-byte offset of the next IFD (if exists). Each TIFF file must have at least one IFD and each IFD must contain at least one entry.
Finally, the directory entries, as stated above, are 12-byte long divided in 2-bytes for the tag that identifies the field, 2-bytes to identify the Type field, 4-bytes to idenfity the number of values present and the last 4-bytes are for the Value Offset, which is the file offset of the Value for the field. In the context of a TIFF file, a field is a logical entity consisting of a TIFF tag and its value.
In terms of the file structure, a multi-page TIFF file is composed by more than one Image File Directory, with each IFD defining a subfile.
In Figure 2.1, is possible to observe the TIFF file structure, as explained above.

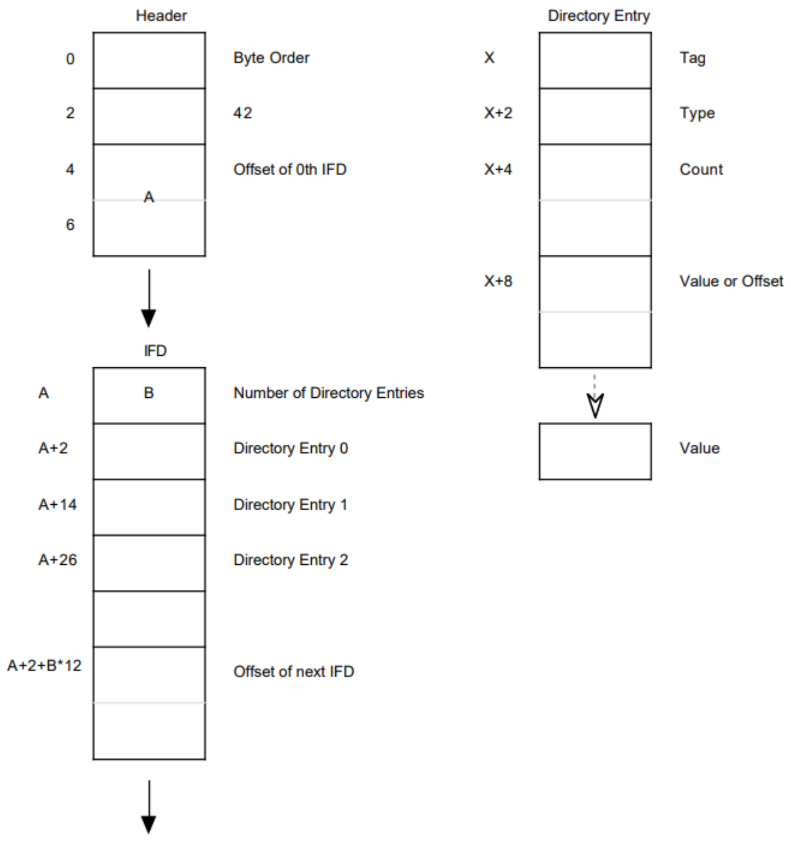


Figure 2.1: General structure of a TIFF file [5]

2.2 Portable Document Format (PDF)

After the invention of the PostScript language in the ’80s, the company Adobe Systems presented the first version of the Portable Document Format (PDF) in 1993. This file format was created to fulfill the need for a graphic visual interface to the PostScript, which was only a standard page description language that was sent to the printing devices to be converted into printed pages. The appearance of this powerful format represented a change in the way documents were exchanged, being standardized in 2008 as ISO 32000.

In this dissertation’s context, the Portable Document Format is the desired format for the documents that result from the automated workflow.

In terms of the PDF file structure, it can be considered that PDF is divided into four different components [6]: Objects, File Structure, Document Structure, and Content Stream.

The ‘Objects’ component contains the set of object types that PDF uses to represent objects. The type of objects included in PDF are Boolean values, numbers (real and integers), strings, names, arrays, dictionaries, streams, and the null object. Objects that need to be referred by others are called “indirect objects” and have to be labeled with a unique object identifier. This identifier consists of a positive integer and a non-negative integer. For example, the indirect object Project can be defined as:

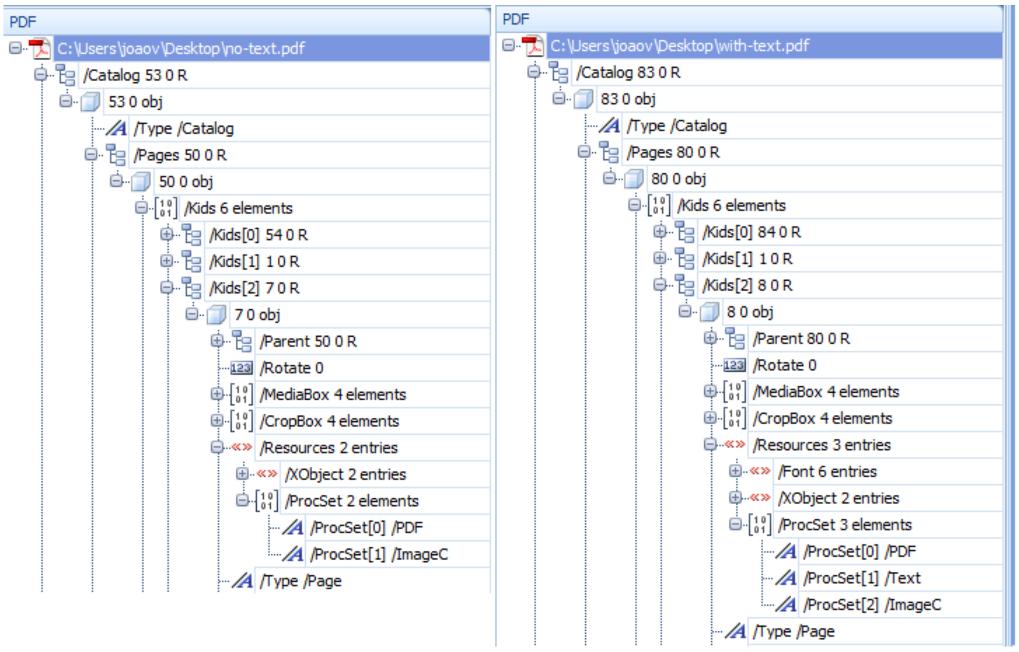
```
1 0 obj
    (Project)
endobj
```

When this object is referred by another object, the reference should be “1 0 R”, where ‘R’ is a keyword for referencing.

Within the many existing objects, there is one that is worth referring to in the context of this project, which is the Optional Content Group, as known as, Layer. These objects consist of sections of content that can be displayed or hidden in the document. The existence of these layers allows the distinction between three types of PDF files: digitally created PDF files where both the text characters and the meta-information have an electronic character designation; scanned PDF files that are image-only and the searchable PDF files that often result from the previous ones after an OCR process is performed, where an extra layer of text is added to the existing image layer. There is software capable of inspecting a PDF file content, providing an overview of its general structure, like PDFXplorer [1]. Figure 2.2, for example, shows the difference between the structure of a only-image PDF file page (left) and the structure of the same PDF file page with an additional text layer (right). As it is possible to observe the object “8 0 obj” on the right, which is similar to the object “7 0 obj” on the left, contains one more entry, referring to the text contained in that specific page.

The way the objects are stored, accessed, and updated in the document is defined by the ‘File Structure’. The file structure is composed of four different elements: a header, identifying the PDF version; a body, where the objects are contained; a cross-reference table to store the information regarding the indirect objects and a trailer that provides the location of the cross-reference table and of other special objects. In

[1] https://www.o2sol.com/pdfxplorer/overview.htm



```
PDF
C:\Users\joaov\Desktop\no-text.pdf
  /Catalog 53 0 R
    53 0 obj
      /Type /Catalog
      /Pages 50 0 R
        50 0 obj
          /Kids 6 elements
            /Kids[0] 54 0 R
            /Kids[1] 1 0 R
            /Kids[2] 7 0 R
              7 0 obj
                /Parent 50 0 R
                /Rotate 0
                /MediaBox 4 elements
                /CropBox 4 elements
                /Resources 2 entries
                  /XObject 2 entries
                  /ProcSet 2 elements
                    /ProcSet[0] /PDF
                    /ProcSet[1] /ImageC
                /Type /Page
```

```
PDF
C:\Users\joaov\Desktop\with-text.pdf
  /Catalog 83 0 R
    83 0 obj
      /Type /Catalog
      /Pages 80 0 R
        80 0 obj
          /Kids 6 elements
            /Kids[0] 84 0 R
            /Kids[1] 1 0 R
            /Kids[2] 8 0 R
              8 0 obj
                /Parent 80 0 R
                /Rotate 0
                /MediaBox 4 elements
                /CropBox 4 elements
                /Resources 3 entries
                  /Font 6 entries
                  /XObject 2 entries
                  /ProcSet 3 elements
                    /ProcSet[0] /PDF
                    /ProcSet[1] /Text
                    /ProcSet[2] /ImageC
                /Type /Page
```

Figure 2.2: Example of general structure of a PDF file, extracted from PDFXplorer software

the case of the PDF file being updated, instead of re-writing the document and changing the existing file structure, the changes are appended to the end of the file, creating an additional smaller file structure.
The third component is the ‘Document Structure’, and its purpose is to determine how the object types can be used to represent the several components of a PDF document. A PDF document can be considered as a hierarchy of objects that are contained in the section ‘body’ of the File Structure (Figure 2.3). The root of this hierarchy is the object “catalog”, which is a dictionary, as most of the other objects included in this hierarchy. The relationships between parent, child, and siblings of the hierarchy are examples of the indirect references that were mentioned before.
Some of the ‘sons’ of the object ‘catalog’ include the Page Tree, which defines the order of the presentation of the pages within the document, the Outline Hierarchy, which defines the hierarchy between sections and subsections, or the Article Threads, that allows the representation of logically related items that are not physically sequential.
Finally, the last component, ‘Content Stream’, contains all the instructions needed for the graphical part of the pages. This component is represented in the same way as other PDF objects, however, its access is done differently. Each page of the document must have one or more content streams associated.

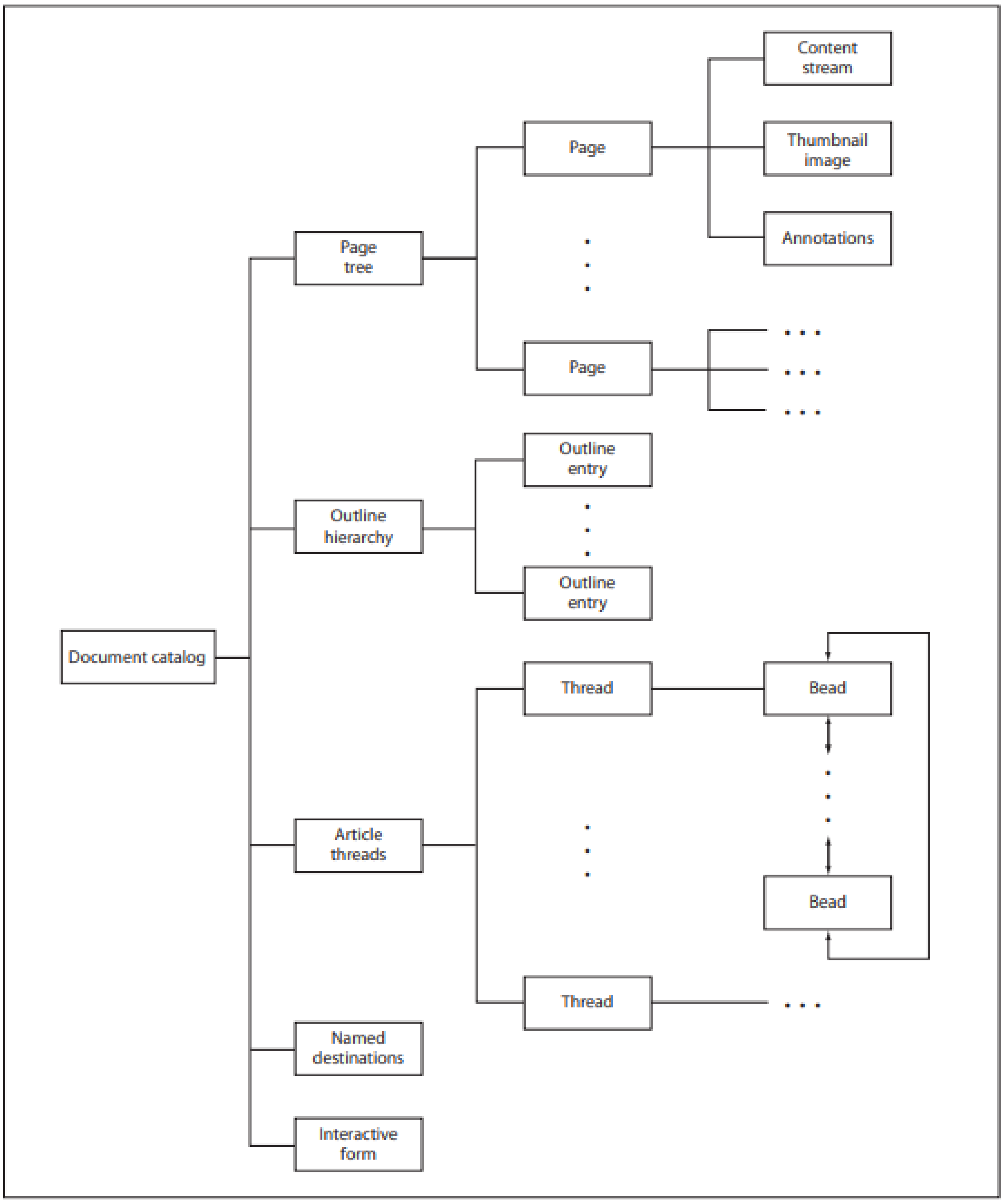


Figure 2.3: PDF document structure [6]

2.3 JavaScript Object Notation (JSON)

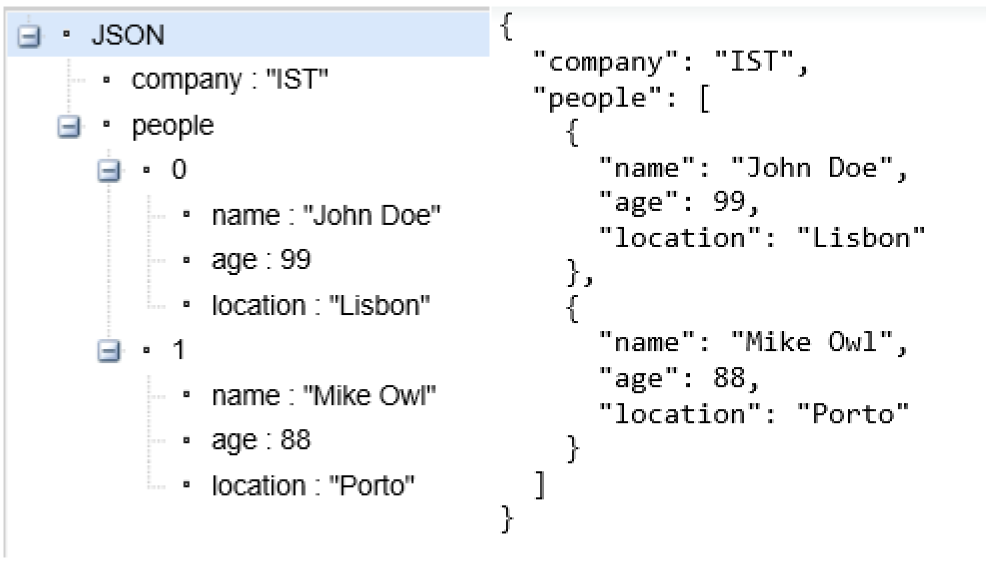
JSON, which stands for JavaScript Object Notation, is a text format for storing and transporting data. Its syntax is derived from the Javascript object notation, however JSON format is text only. It is built on two structures: a collection of name-value pairs and an ordered list of values. Both are universal and programming language-independent data structures, so the code for reading and generating JSON exists in most of the programming languages, and the data can be easily exchanged between platforms. JSON can be useful as a mean of data exchange used by APIs, producing files for configuring systems, or a way of keeping persistent data.
As it was mentioned before, JSON is composed of name-value pairs, being a name, a string enclosed in quotation marks, while a value can either be a string, a number, a Boolean expression, an array, or even another object. The name-value pair has to follow a specific syntax, with the name being followed by a colon, followed by the value. Each name-value pair is separated by a comma. An example of a name-value pair would be:

“name” : ‘John Doe’

A JSON Object Literal is an unordered group of name-value pairs and is surrounded by curly braces ({}). For example:

{“name”: ‘John Doe’ , “age”:99}

In Figure 2.4, it is possible to observe an example of some JSON objects in a file (on the right) and how the same data is showed in a JSON viewer (on the left).



```
JSON
    company : "IST"
    people
        0
            name : "John Doe"
            age : 99
            location : "Lisbon"
        1
            name : "Mike Owl"
            age : 88
            location : "Porto"
```

```
{
  "company": "IST",
  "people": [
    {
      "name": "John Doe",
      "age": 99,
      "location": "Lisbon"
    },
    {
      "name": "Mike Owl",
      "age": 88,
      "location": "Porto"
    }
  ]
}
```

Figure 2.4: Example of JSON objects and their visualization

Chapter 3

Optical Character Recognition

One of the fundamental parts of this dissertation is to convert scanned images into searchable documents and to do so, there is a very useful technology called Optical Character Recognition (OCR). As its name implies, this solution allows recognizing characters contained in a scanned document or image, turning it into text that can be manipulated by a machine. This chapter briefly describes the core concepts of an OCR system, its different phases and enumerates some of the existing software capable of performing this type of job.

3.1 OCR review

As mentioned above, an OCR system's main problem is the optical recognition of processed characters by a machine.

The first attempts to create an optical character recognition system date back to 1912, with the invention of the optophone [7]. This device's purpose was to help the blind by scanning text present in a document and producing different sounds according to the characters being read. Thirty-nine years later, in 1951, Gismo, a machine that was able to convert printed messages into machine language to be processed by a computer, was invented, and it is still considered the first "modern" optical character recognition machine. In 1954, Reader's Digest[1] magazine was the first company to have this kind of machine installed to convert typewritten sales reports into punched cards to be used as an input for the computers. After Gismo, other machines were created, and even if they were available for the public in general, their high cost prevented a significant number of sales.

At the end of the 20th century, with the improvement of hardware solutions (and its decreasing cost), constructing machines to perform OCR was no longer useful, so software packages that could execute the same job started to become available. This shift in the way OCR systems were produced resulted in cheaper solutions and, at the same time, in an increase of systems sold [8]. As of now, there are plenty of solutions available in the market, either they are paid or open-sourced.

Both handwritten and printed characters can be recognized by a typical OCR system, with the perfor-

[1] https://www.rd.com/

mance of their recognition being directly linked with the quality of the input documents (the better the input, the better are the results) and the quality of the software being used (since each software uses its own way to recognize characters).

Some of the typical problems an OCR system can face are variations in the fonts being used, deformations on the document being used as an input, or images and graphics mixed with the text. The usage of different fonts was partially solved with the creation of two standardize fonts, easing the OCR procedure: OCR-A (American version) and OCR-B (European Version). The characteristics of these two types of fonts include the same thickness and width and the distinct shape of every character [9]. While the usage of these fonts, which are typically present in credit cards and bar codes, can improve the accuracy of the optical recognition, it still does not solve the recognition of handwritten characters and other types of fonts that were used in older documents.

Although important steps were made in recent years, there is still no perfect system, so there is always room for improvement. For example, handwriting, historical fonts, and the non-Latin alphabet are the focus of a great variety of studies being done that aim to improve the recognition of these types of characters. In addition, some topics like deep learning and voting systems can also play an important role in implementing new and better OCR solutions [10].

3.2 OCR reference architecture

Since its scanning, an image suffers different types of processing during an OCR workflow until it reaches the final result. Depending on several factors, such as the input/output type or the software being used, an OCR architecture may differ from system to system.

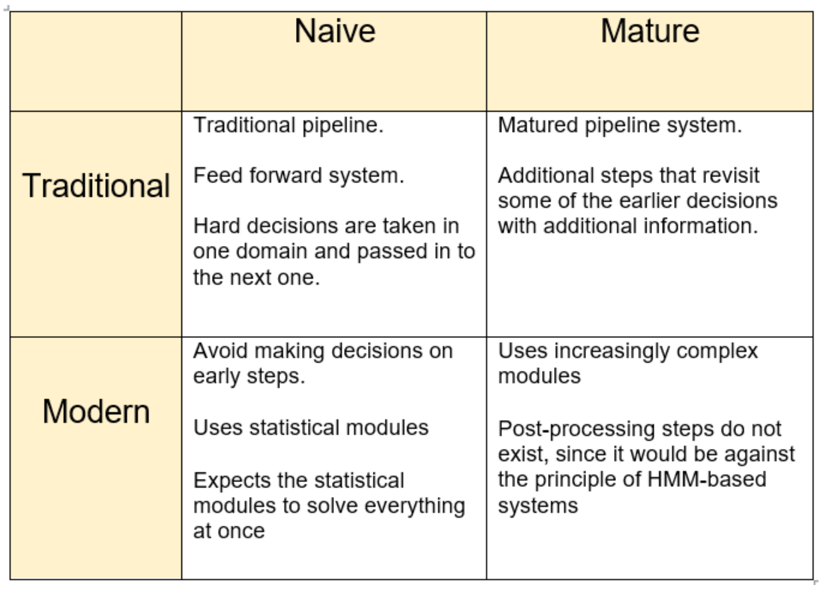
Accordingly to Ray Smith, the creator of Tesseract (one of the OCR software mentioned in the ”Examples of existing solutions” section), the existing architectures of OCR systems can be divided into four different categories [11] : Traditional-naive, Traditional-mature, Modern-naive and Modern-mature. (Figure 3.1)

The traditional-naive systems are the ones composed by a traditional pipeline that starts with a series of steps that make hard decisions on one domain and passes the results to the next part of the pipeline.

The traditional-mature are an improvement of the previous ones, where the pipelined systems have additional steps that revisit some of the earlier decisions that were taken, with additional information obtained in other parts of the pipeline. Some examples of this are the adaptive character classification or the use of document dictionaries.

In other hand, the modern-naive OCR systems, try to avoid making decisions at an early stage and push all the hard problems into a monolithic statistical module, such as a Hidden Markov Model (HMM). The system then expects this module to solve everything at once.

Finally, the modern-mature systems are characterized by increasingly complex models that consider all the printed information structure, since using post-processing modules to revisit earlier decisions would be against the main idea of using HMM-based system.



	Naive	Mature
Traditional	Traditional pipeline. Feed forward system. Hard decisions are taken in one domain and passed in to the next one.	Matured pipeline system. Additional steps that revisit some of the earlier decisions with additional information.
Modern	Avoid making decisions on early steps. Uses statistical modules Expects the statistical modules to solve everything at once	Uses increasingly complex modules Post-processing steps do not exist, since it would be against the principle of HMM-based systems

Table 3.1: Different OCR system's architectures, according to Ray Smith [11]

The architecture used as a reference for this dissertation can be split into four main steps (Figure 3.3), and each one can be divided into several other sub-steps.

The first main step is called Pre-Processing. Considering that digitized images can present a great range of different layouts, fonts, and colors, there is a need to prepare them to obtain a better result when performing the OCR. Some of the most common methods that are used in the pre-processing of the images by the state-of-art software are binarization (Figure 3.1), in which the main goal is to convert the source image into a binary one (only black and white pixels); deskewing (Figure 3.2), by adjusting the rotation of the document, making sure it is straight or simply cropping or rescaling the images[12].

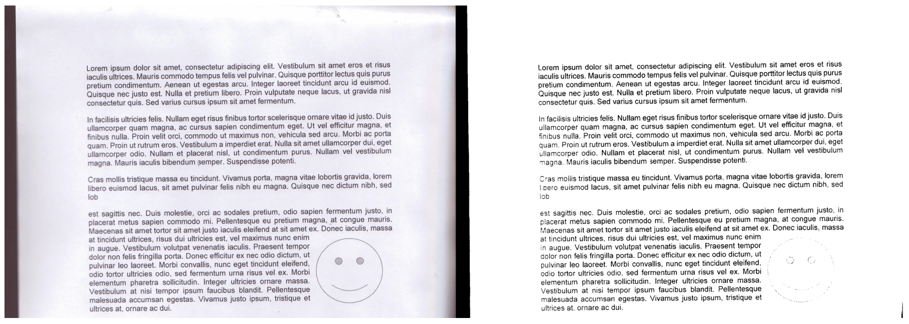
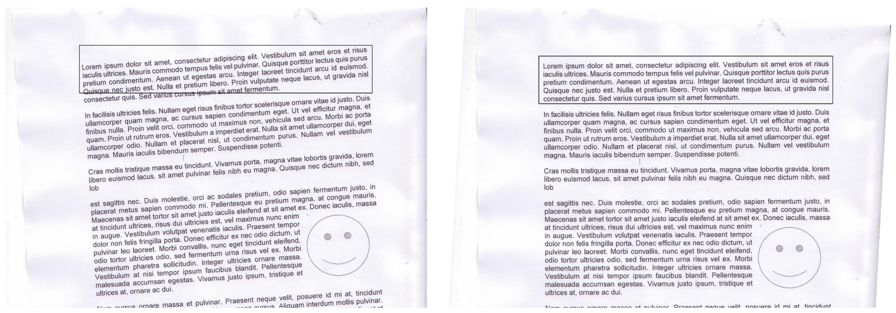


Figure 3.1: Example of binarization

The second step is segmentation. Depending on the software used and the user requirements, different types of segmentation can be performed. Usually, there are three types of segmentation: page segmentation, line segmentation, and word segmentation. The first one aims to distinguish between areas of the image that contain text from those that do not. After that, line segmentation splits those found regions into lines of text. Finally, the lines are segmented, resulting in groups of words [12].



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Vestibulum sit amet eros et risus iaculis ultrices. Mauris commodo tempus felis vel pulvinar. Quisque porttitor lectus quis purus pretium condimentum. Aenean ut egestas arcu. Integer laoreet tincidunt arcu id euismod. Quisque nec justo est. Nulla et pretium libero. Proin vulputate neque lacus, ut gravida nisl consectetur quis. Sed varius cursus ipsum sit amet fermentum.

In facilisis ultricies felis. Nullam eget risus finibus tortor scelerisque ornare vitae id justo. Duis ullamcorper quam magna, ac cursus sapien condimentum eget. Ut vel efficitur magna, et finibus nulla. Proin velit orci, commodo ut maximus non, vehicula sed arcu. Morbi ac porta quam. Proin ut rutrum eros. Vestibulum a imperdiet erat. Nulla sit amet ullamcorper dui, eget ullamcorper odio. Nullam et placerat nisl, ut condimentum purus. Nullam vel vestibulum magna. Mauris iaculis bibendum semper. Suspendisse potenti.

Cras mollis tristique massa eu tincidunt. Vivamus porta, magna vitae lobortis gravida, lorem libero euismod lacus, sit amet pulvinar felis nibh eu magna. Quisque nec dictum nibh, sed lob

est sagittis nec. Duis molestie, orci ac sodales pretium, odio sapien fermentum justo, in placerat metus sapien commodo mi. Pellentesque eu pretium magna, at congue mauris. Maecenas sit amet tortor sit amet justo iaculis eleifend at sit amet ex. Donec iaculis, massa at tincidunt ultrices, risus dui ultricies est, vel maximus nunc enim in augue. Vestibulum volutpat venenatis iaculis. Praesent tempor dolor non felis fringilla porta. Donec efficitur ex nec odio dictum, ut pulvinar leo laoreet. Morbi convallis, nunc eget tincidunt eleifend, odio tortor ultricies odio, sed fermentum urna risus vel ex. Morbi elementum pharetra sollicitudin. Integer ultricies ornare massa. Vestibulum at nisi tempor ipsum faucibus blandit. Pellentesque malesuada accumsan egestas. Vivamus justo ipsum, tristique et ultrices at, ornare ac dui.

Figure 3.2: Example of deskewing

The third step is the recognition itself. During this step, segmented words are recognized by a trained model, resulting in a textual representation of the original image. The method used to recognize words differs from the OCR engine (subsection ”Examples of existing software” of this chapter describes the operation of three different OCR solutions) [12].
And finally, the fifth step is post-processing. After the words are recognized, the output is ready. However, some sub-steps can be applied during this phase, such as using dictionaries, which will prevent the output from containing non-existing words (sometimes it can be a problem since more technical or made-up words are not present in the dictionary). Considering that the output is normally given in plain text, the post-processing step can also include converting or assembling the final result into more complex formats (such as PDF, for example) [12].

$$\begin{tikzcd}
\boxed{\text{Pre-processing}} \arrow[r] & \boxed{\text{Segmentation}} \arrow[r] & \boxed{\text{OCR}} \arrow[r] & \boxed{\text{Post-processing}}
\end{tikzcd}$$

Figure 3.3: Main steps of an OCR reference architecture

3.3 Examples of existing software

Throughout the Web, several different tools allow users to perform OCR. Not every software available provides the same result accuracy, and some of the solutions available are paid. However, there are already some free and open-source tools that allow users to convert their images or scanned documents into text with no cost, still achieving good results. This subsection describes two open-source tools (Tesseract and OCRopus) and a paid software (Adobe Acrobat Pro DC) commonly used to perform OCR.

3.3.1 Tesseract

Created by Hewett-Packard in 1985, this OCR system is currently being developed by Google. The package available in open-source is composed of an OCR engine (libtesseract) and a command-line program (tesseract), not including a GUI application. According to Tesseract's documentation, this tool can recognize more than 100 languages and be trained to recognize even more [13]. Its latest stable version is 4.1.1, which was released on December 26th of 2019. Compared with the previous version, a new neural net (LSTM) based OCR engine was added [14]. Even though the introduction of this new engine can result in higher accuracy, it also requires more computing power and trained data. Due to this fact, Tesseract developers still allow users to use the engine developed in the previous version. To recognize the words in an image, the Tesseract engine (version 3) conducts a step-by-step process [15] (Figure 3.4) that works as follows:

1. It makes use of its built-in routines for pre-processing the images.

2. Then, a connected component analysis is performed, storing the outlines of the components found. Those components are gathered into Blobs which are then organized into text lines.

3. The engine then analyses the lines to find fixed-pitch text.

4. If the text found was fixed-pitch, the lines are split into words, and these are split into characters right away. However, if the spacing between characters is non-fixed-pitched (or proportional), the text is only separated into words, making use of definite spaces and fuzzy spaces. For example, this can occur in italic words, where the spacing between words and characters is not always the same.

5. After the word and line finding is done, the recognition is done in a two-pass process. During the first pass, a static classifier is used. The successfully recognized words are passed into an adaptive classifier, which will use that data to recognize the words that are left more accurately. The second pass is used for the cases when the adaptive classifier finds useful information too late. By running the second pass over the page, it is possible to recognize words at the top of the document that were not recognized well enough in the first pass.

6. In the final step, some fixing is performed. This happens for the cases where some decisions are fuzzy or need some correction from an early step, like x-heights, incorrect spacing or words that need multi-word context to resolve.

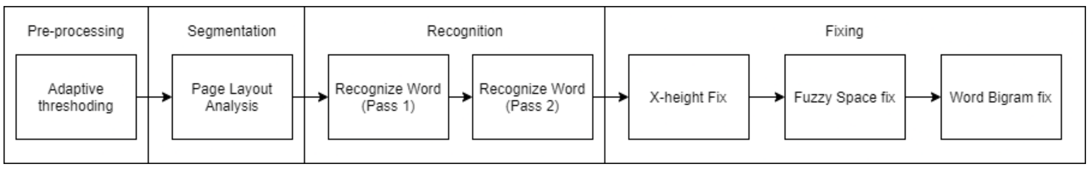


Figure 3.4: Tesseract main steps

The possible outcome formats include plain text, hOCR, PDF (searchable or not), or TSV.

3.3.2 OCRopus

More than just an OCR system, OCRopus is a set of document analysis tools that also includes a functional OCR engine. Its development started in 2007 as a Google-sponsored project in collaboration with the Image Understanding and Pattern Recognition (IUPR) research group, headed by Professor Thomas Breuel. The first version released under an Apache license was based on three existing components: a high-performance handwriting recognizer deployed by the US Census Bureau, the Tesseract OCR engine, and software for layout analysis. After some iterations of the project, Tesseract was no longer the default text recognition tool used because of its inability to cope with the statistical natural language models used by OCRopus, being replaced by a model using Recurrent neural network (RNN). This is a modular system built with several different scripts: it is necessary to run separate commands to perform the different steps of the process (Figure 3.5): pre-processing, segmentation, recognition and post-processing.[16].

1. During the first step, a pre-processing and cleanup of the original image occurs.
2. The segmentation is the second step, and it is done primarily by finding individual letters and calculate their x-height, creating a "scale". After this, the system uses these calculations to try to find the lines contained in the text. First, by removing the components which have a big difference compared to the "scale" and then by detecting the top and bottom edges of the remaining ones. Whatever is in between that top and bottom is considered a line.
3. After the segmentation is done, the third step is character recognition. As mentioned before, OCRopus uses an LSTM Recurrent Neural Network to predict which letters are contained in the image. This implementation uses columns of pixels as input and scores for each possible letter as output. When recognizing a character, the model assumes that the one that has a higher score corresponds to the correct letter.
4. Results can be improved if new training models are included, which OCRopus developers encourage users to do, providing tools to do so. The creation of this new training models, together with the assembling of the information into the final hOCR output, can be considered as the forth main step of the process.

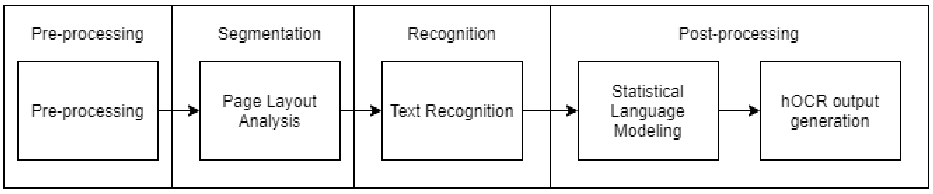


Figure 3.5: OCRopus main steps

According to the OCRopus documentation, the default parameters and settings consider that the image being used has 300DPI, is black and white, and contains standard font size. If the input differs from these settings, the recognition quality can be affected.

The latest version of this software was released as Ocropus3 in May of 2018. Compared with the initial version, this new one enables new deep network structures usage and deep learning techniques, and GPU support. These new additions can improve the results, providing faster training and, therefore, faster predictions.
The output format of OCRopus is hOCR.

3.3.3 Adobe Acrobat Pro DC

Adobe Acrobat Pro DC is part of the Adobe Acrobat DC software family, together with Adobe Acrobat DC Standard and Adobe Acrobat Reader DC. It was firstly released in 2015 and compared with the previous Adobe versions, this one includes new features, such as a new interface or cloud storage support.
In addition to standard tasks like view, create or modify PDF files, this paid version of Adobe PDF editor, also allows the transformation of scanned files into editable PDF documents. With an easy user interface, Acrobat can recognize text in any PDF or image file, individually or in a group of multiple files, with a couple of clicks. After the OCR process is finished, by default, the recognized text is saved inside the original file and in case the original file was an image, a new PDF file is created with both the image and the text layer.
Since Adobe Acrobat Pro DC also allows editing the content in a PDF file, if the recognized text is not completely correct, it is possible to manually correct it, however it is a time-consuming and laborious process.
Adobe Acrobat Pro DC is a commercial solution, so the specific way in which the optical character recognition is performed, is not disclosed.

3.4 hOCR data schema

The hOCR microformat [17] was originally developed by Thomas Breuel with the goal of creating a representation that would make it easier to store, share, process, and display OCR results while reusing as much existing technology as possible. Thus, it was built on top of HTML but with specific features that better represent document layout analysis.
Using the DIV and SPAN tags that do not have any specific meaning in HTML, it is possible to associate style and other information with text regions inside an HTML document. These tags allow some standard attributes, namely the class, style, and title attributes. The class attribute is used to identify the particular class or application to which the tag belongs (in this case, the class can either be ocr or ocrx). The style attribute is used for associating style information with its regions of text (for example, typographic and layout information). Finally, the title attribute can contain arbitrary text, and it is used to encode tag-specific information. In the specific case of the hOCR format, the DIV and SPAN tags are called elements, and the information contained in their title attribute is called properties.

The hOCR elements can be divided into five categories [18]: Typesetting, Float, Logical, Inline, and

OCR engine-specific elements.

- Typesetting elements are those that describe the areas of a page that can be grouped. As in other typesetting systems (like Microsoft Word or LibreOffice), each page in this model is divided into a number of areas. Some examples of typesetting elements are *ocr_page*, *ocr_columns*, *ocr_carea*, or *ocr_line*.
- Float elements can be described as those outside the regular reading order, like images, tables, or page numbers. Some of these elements are *ocr_float*, *ocr_image*, or *ocr_page_no*.
- Logical elements, as the name implies, are used to structure the hOCR document logically. Some examples are *ocr_document*, *ocr_title*, *ocr_author*, and *ocr_abstract*.
- Inline elements are composed of elements that should flow like text. Some examples are unrecognized characters and words (*ocr_glyph*) or mathematical and chemical formulas (*ocr_math* and *ocr_chem*).
- Finally, the OCR engine-specific elements are the elements that are not transposable between different OCR engines. These elements are differentiated by the use of the prefix ocrx. Some examples are *ocrx*, *ocrx_line* or *ocrx_word*.

When it comes to the hOCR properties, they can provide useful information such as:

- Physical page number (*ppageno*)
- Logical page number (*lpageno*)
- OCR-engine specific font name (*x_font*)
- OCR-engine specific font size (*x_fsize*)
- Bounding boxes (*bbox*), which are rectangular boxes around the element with their coordinates [18]. The bounding box is represented in the blue rectangle of Figure 3.6. The numbers used for its representation are the coordinates of the upper left corner of the element (10,20) and the lower right corner coordinates (160,30). In this example, the property is "bbox 10 20 160 30".

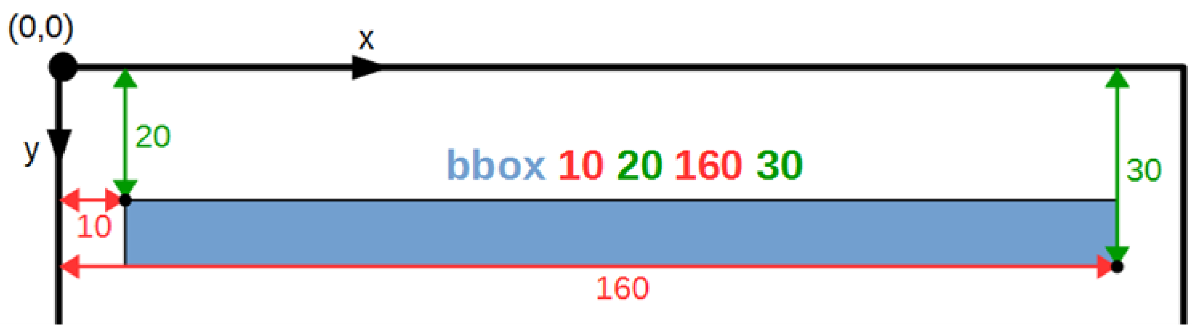


Figure 3.6: Example of bounding box propery

- Baseline (*baseline*). As shown in Figure 3.7, the baseline is represented by two numbers, the first being the slope (0.86°) and the second the constant term of an equation, which considers the relationship between the baseline and the lower-left corner of the bounding box [18]. The baseline is crossing the y-axis in the -18 coordinate and the slope angle is arctan(0.015)=0.86°, so the value of the baseline property would be "baseline 0.015 -18".

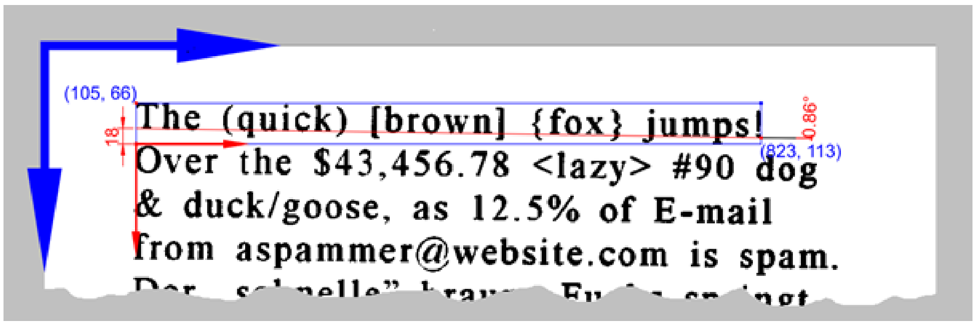

(105, 66)
18
0.86°
(823, 113)
The (quick) [brown] {fox} jumps!
Over the $43,456.78 <lazy> #90 dog
& duck/goose, as 12.5% of E-mail
from aspammer@website.com is spam.

Figure 3.7: Example of baseline property

An example of a hOCR file can be seen at Appendix A.

Chapter 4

Problem analysis and proposed solution

This chapter, is divided in two parts. In the first part, the problem presented in this dissertation will be analyzed and discussed to justify the decisions taken during the final solution design. This part then is divided into two different sections. The first one is an overview of the existing workflow currently being applied at ARMS to digitize their documents, and contextualize the problems that need to be tackled. The second section describes which are the requirements for the implementation of a functional workflow that could solve those problems.
The second part of this chapter aims to present the proposed solution for the problem presented above. The design and development of a final solution can be divided into two different steps: first, a proof-of-concept was designed and produced, taking into account the main steps that have to be performed in such workflow. This simple workflow should be developed to lay the foundations on what the final solution could be. Secondly, a completer workflow was designed and developed based on the feedback and needs of the stakeholders. Therefore, this part of the chapter is divided into two different sections. The first section contains an overview of the proof-of-concept developed before the final version. In contrast, the second section describes the final version, with a detailed description of each step of the process.

4.1 Existing solution

Currently, the Archives and Records Management Section of the United Nations have their own workflow to digitize and process documents. The existing workflow can be divided in four different tasks, as can be seen on Figure 4.1.

This workflow is performed by a member of the staff. This member has the support of an IT team in case there is an issue with the software or any of the IT hardware.
It starts with the digitization of the documents. However, since this step is out of the scope of this

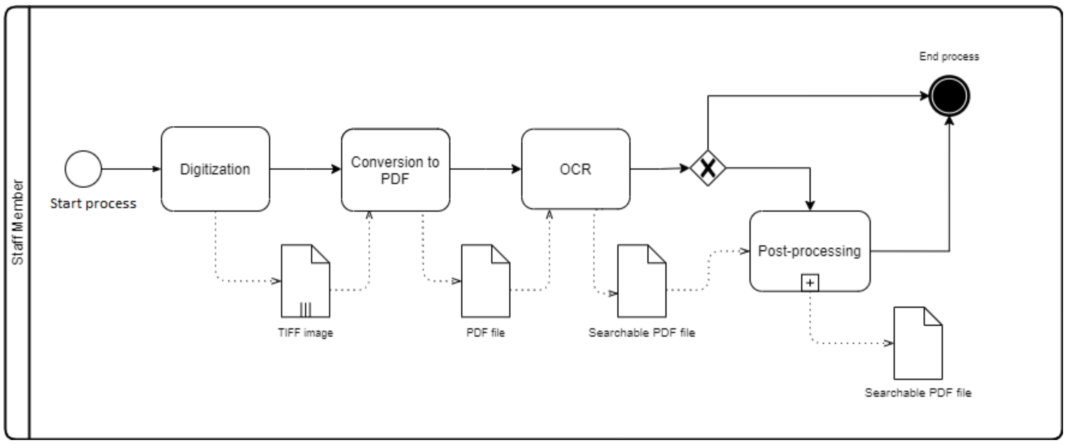


Figure 4.1: Existing solution at ARMS

dissertation, it will not be included in the proposed solution, and it will be assumed that the documents are already digitized. After the digitization, the documents that were scanned are saved as TIFF images. After that, the TIFF images are converted to a PDF file with defined settings. This is done so that the OCR can be performed in the next task. Both of these tasks are possible by using the Adobe Acrobat Pro DC software. At the end of this task, the result is a searchable PDF file.

The last step of the workflow is post-processing, it is optional and depends on two conditions that can be divided into two sub-processes, as can be seen on Figure 4.2 and Figure 4.3. The first is the final file size: if the file is too large, it is compressed or separated. Otherwise, the file is saved. The second condition is the naming convention used: if it is the desired one, the file is saved. If not, it has to be renamed before that happens.

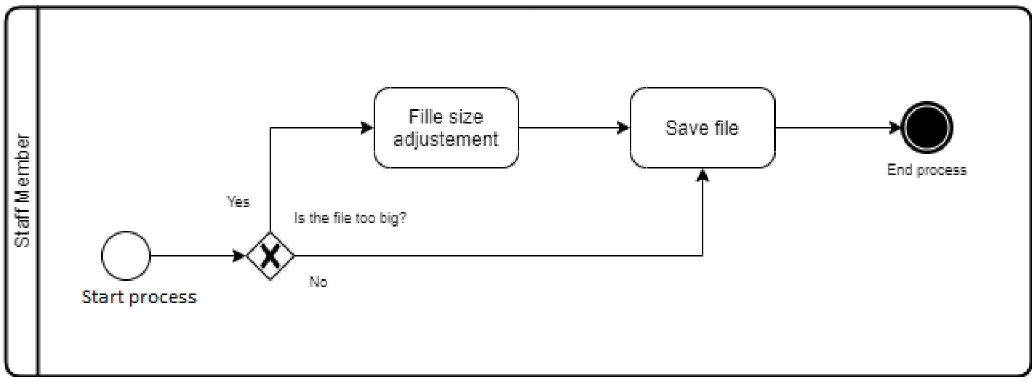


Figure 4.2: Resizing process

The majority of the problems that occur during the processing of a document are between its digitization and its final state, mainly during the OCR task. This happens because the software used, despite offering that functionality, is not perfect for text recognition on documents. Although it can recognize the regions containing text (i.e., the segmentation seems reasonable), the recognized text is often just “garbage” characters. It also does not offer much flexibility on parameters available since the only settings that can

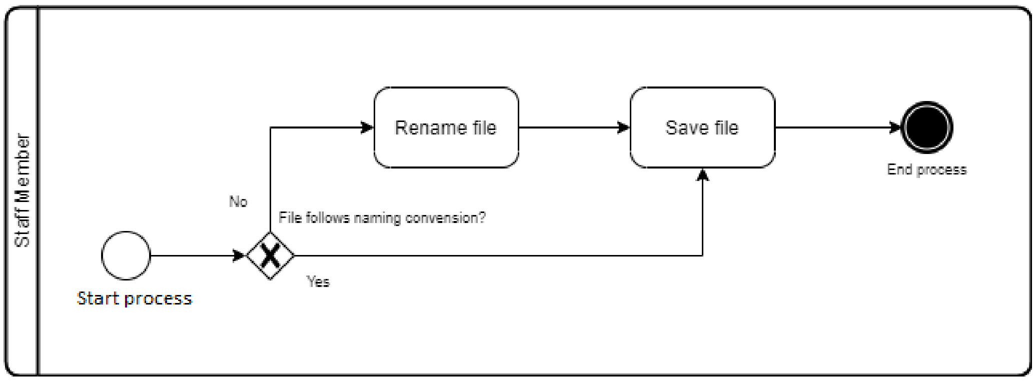


Figure 4.3: Renaming process

be pre-defined are the language in which the text is written and the format in which the final file should be saved. In addition to that, it has some other points of failure, such as crashing during the processing of a document without providing additional information on what caused it. This issue may not be very relevant for the regular user, but this lack of information on the current state of the process makes it difficult to bypass any of the problems that may arise.
Another problem that the existing workflow has, is its low efficiency since the intermediary steps between each task (for example, place the TIFF image to be converted to PDF or prepare the document to be OCR'ed) have to be performed by a member of the staff manually.
Ideally, the automated workflow should work as a black box. The user (in this case, the staff member) should only be required to place the digitized document in the workflow, select the desired parameters and start the workflow. While the documents are being processed, the user could perform other unrelated tasks without worrying about intermediary steps. In the end, a similar file to the original one is ready, although this is a PDF file and contains the recognized text.

4.2 Requirements

After presenting the existing solution, the focus of this dissertation is to improve it. Nevertheless, at the same time, this one has to be considered the minimum acceptable solution possible, so whatever the proposed solution will be, it has to provide the same or better results than the existing one.
Some of the requirements for a new solution are:

- The OCR accuracy should be improved, and new specific features may be added for the specific needs of ARMS.
- The user interface should be easy to use so that a user with basic IT knowledge can perform the tasks needed to use the workflow. A user guide explaining how to install and use the workflow should also be prepared.
- The computers in ARMS have Windows 10 installed, so the proposed solution must be able to run in this operating system. Also, it should only use free-to-use and open-source software.

- The workflow should be modular enough in case new functionalities or parameters have to be added. It also has to keep track of the state of the process so that it can start from where it stopped in case of interruption.

Scanning is outside the scope of this dissertation, so the focus is on improving the steps between scanning and the final result. The goal of the workflow is that given an image or a set of TIFF images, these after being processed by the workflow, with the definition of certain parameters by the user and without intervention in the intermediate steps, result in a PDF document with a text layer, so that it is searchable. This final file should also have the ideal size to be uploaded to the public database and be accessed/downloaded by the general public.

Building a workflow capable of processing all of the documents handled by ARMS with great accuracy, is a difficult task. These documents are very inhomogeneous, containing different sizes, different text structures, and different fonts. Some of the scanned documents are very old, and therefore barely legible, some of them even difficult to recognize by the human eye. However, it is important that the workflow is able to process the majority of the documents.

These requirements were collected during video meetings with the ARMS staff, that occurred every two weeks during the development of this dissertation. Besides gathering the requirements, these meetings would also serve to update the progress.

4.3 Proof-of-concept

By analyzing the problem presented in the previous section, it was clear that the presented solution would have to follow the following general steps:

1. Split the multiple-image TIFF files
2. Process each page, applying OCR
3. Merge all pages back into a single PDF

The first step is needed because after the digitization, the scanned images are stored in TIFF files, the majority of them containing more than one image. Since every image needs to be processed independently, they have to be split first. The second step is applying the optical character recognition to every single image, recognizing the text in each of them. The result of this step could already be one PDF file for each TIFF image, containing the original image and a layer with the recognized text. Finally, the last step would be merging all the separated pages back into a single file.

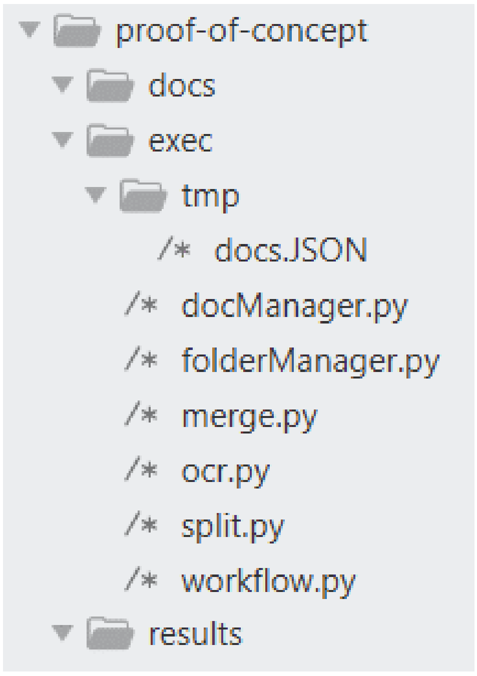
It was then necessary to create a first version of the workflow to be able to determine what kind of adjustments could be made afterwards, better fitting the workflow to the needs of the stakeholders. For that reason, a proof-of-concept was developed.

4.3.1 Technologies used

The scripts were written in Python (version 3), and all the libraries and packages used would have to be compatible with this coding language. This was the chosen programming language due to its simplicity and vast library support. The developing environment was a Linux virtual machine, running Ubuntu 16, with 6GB of RAM and 20GB of disk space.
For the OCR processing, after testing some of the existing solutions, Tesseract was the chosen one. This open-source software provides good results when recognizing text, it is easy to integrate with Python, and it is well documented.

4.3.2 Development

The approach chosen to build this proof-of-concept was the creation of scripts for each part of the process. These scripts were then coordinated by another main script that would make the connection between them. This way, modularity can be assured, and it would make it easier in case new steps or features needed to be added.
For the purpose of developing this proof-of-concept, six different scripts were created: one for each step of the process (*split.py*, *ocr.py*, and *merge.py*), two auxiliary scripts to keep track of the state of the process, and manage the creation/deletion of folders/files (*docManager.py* and *folderManager.py*) and one main script to manage all the other scripts (*workflow.py*). In addition, a JSON file called *docs.JSON* was also created to keep data persistency related to the state of the workflow.
In addition to the scripts, the workflow also followed a predefined directory structure, as seen in Figure 4.4.



```
▼ proof-of-concept
  ▼ docs
  ▼ exec
    ▼ tmp
        /* docs.JSON
      /* docManager.py
      /* folderManager.py
      /* merge.py
      /* ocr.py
      /* split.py
      /* workflow.py
  ▼ results
```

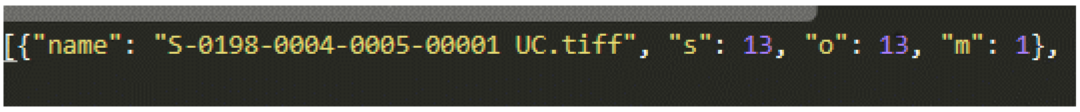
Figure 4.4: Proof-of-concept directory structure

4.3.3 Execution

Following the predefined directory structure created for this workflow, the docs folder was where the documents to be processed should be placed. After the documents were placed there, the user only had to run the following command on the Command Line:

```
$ python3 workflow.py
```

When executing this command, the workflow execution would start.
First, the docManager searches the docs folder looking for files with the ".tif" or ".tiff" extension. Even though they both refer to the Tagged Image File Format, some workflow steps could not deal with the ".tif" version, so if the document had this extension, the script would change it to ".tiff".
After that, the *docs.JSON* file is updated with a new entry related to the document being processed. An entry, similar to the one showed in Figure 4.5, was composed by the name of the original file, the number of pages that were left to split, the number of pages that were left to OCR and a 1 (not merged) or a 0 (merged) to keep track of the merging process. If an entry related to that specific document already existed in the JSON file, the docManager would gather the execution state (i.e., where the workflow stopped in the previous processing of this document) and continue from there, since these fields are updated every time the workflow progresses.



```
[{"name": "S-0198-0004-0005-00001 UC.tiff", "s": 13, "o": 13, "m": 1},
```

Figure 4.5: Example of a JSON entry

Additionally, a folder with the same name as the original document is created in the tmp folder. All the temporary files are kept in that new folder, such as the split pages and the OCR'ed PDF pages.
Considering that a new document is being processed, the first step is the splitting. The *split.py*script performs this step. This script uses the Python Image Library (PIL) to handle the multi-image TIFF files, specifically the Image module. Using the *seek()* and *save()* functions, it was possible to split the TIFF multi-image files into separated ones containing only one TIFF image per file. Next, these files are saved, following the naming convention *"page_" + "#number of the page"* in the tmp folder.
The next step would be the OCR. For that purpose, the *ocr.py* was created. This script used the pytesseract library. This library is a Python wrapper for the Tesseract engine that can be used as a standalone invocation script since it can read all image types supported by the PIL library, including TIFF. For its usage, the user is required to install Tesseract in the host computer. One of the available functions of this library is the function *image_to_pdf_or_hocr()*, that given an image and an extension (has to be PDF of hOCR), produces a searchable PDF, invoking Tesseract to recognize text. Since this is the desired outcome, this is the function used in this script, applied to every split image on the previous step. The result is one searchable PDF file for each TIFF image, all stored in the tmp folder. The naming convention for each saved file is *"#number_of_the_page.pdf"*.

Finally, the last step is merging all the single PDF files that resulted from the OCR processing. This step is performed by the *merge.py* script. Using the module PDFFileMerger of the PyPDF2 library, it was possible to gather all the single PDF files and merge them back into a multi-page searchable PDF file. The naming convention for this file was applying the same name as the original document.
In the end, all the temporary files are deleted, and the workflow is completed.

4.4 Final version

Even though the first version of the workflow was functional and delivering what it was asked for, a legible document similar to the original one, which can be uploaded to the digital archive of the ARMS, it was still too simplistic. It did not have any major improvements compared to the existing solution, except for a significant improvement in the recognized text, which confirmed the idea that Tesseract software would provide satisfactory results, and also reduction on the amount of user interaction needed.
For that reason, a new version started to be designed and developed, having the first version as a starting point. In the end, this final version is the proposed solution that this dissertation aims to contribute with.
After analyzing the flaws and the room for improvement on the proof-of-concept, it was decided that the proposed solution would have to have the following new characteristics:

- Additional pre-processing on the split pages;
- More flexibility on the OCR process;
- Change the way the PDF file was generated;
- Handle concurrent calls better.

An additional pre-processing was necessary because Tesseract software (and most of the OCR software) performs better if the image has better quality. As stated in Tesseract documentation:

"Tesseract does various image processing operations internally (using the Leptonica library) before doing the actual OCR. It generally does a very good job of this, but there will inevitably be cases where it isn't good enough, which can result in a significant reduction in accuracy." [13]

It is possible to see the result of the internal image processing of Tesseract, so an idea would be presenting that result to the user, and in case the image seems problematic, additional techniques could be applied.
More flexibility on the OCR process should also be provided. In the proof-of-concept, the Tesseract software was running with the default settings, not taking advantage of its full potential. Adding the option to choose the language in which the text is written or choose the page segmentation mode are two options that can be included.
The way PDF files were generated should also suffer a change. In the way it was done before, the result of the OCR processing were already single-paged PDF files, and the merge process would only have to

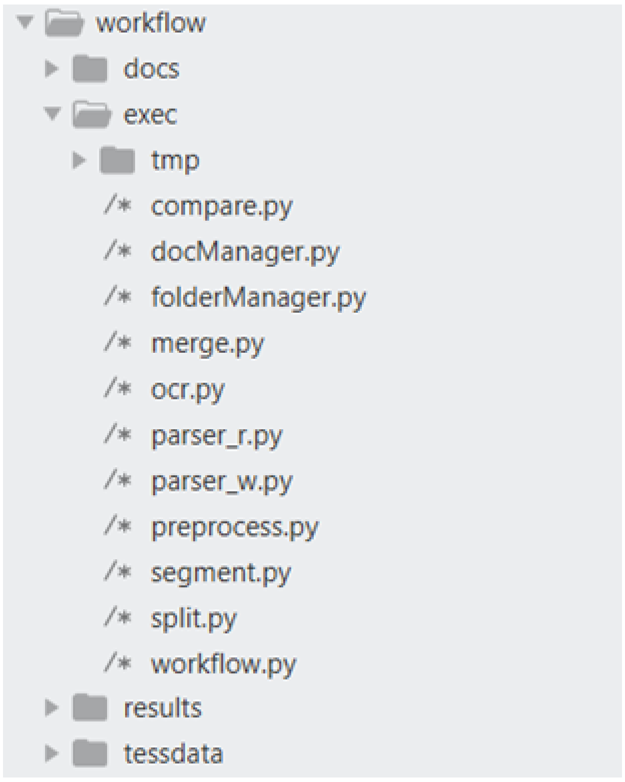
merge all those pages back into a single multi-page PDF file. This led to some of the final results being poor, and it did not allow much flexibility in the way the PDF files were generated.
Finally, the way the workflow kept the state of execution should also be improved, since it could lead to errors. Using only one JSON file to keep the data related to the state of execution could lead to concurrent calls on the file, leading to misunderstandings between writing and reading values. The new solution should either create at least one JSON file for each one of the files that were processed or find another way of keeping data persistency.
In the end, after analyzing these issues, the proposed design has the following steps:

1. Split the multi-image TIFFs
2. Apply pre-processing to each image
3. Perform a first OCR run
4. Segment the images in order to find regions of text
5. Perform a second OCR run on the segmented images
6. Compare the results of both runs and chose the best ones
7. Generate a single PDF file

The first step is the same as in the first version since every image needs to be processed independently. The second step is a new feature, and its focus is to improve the quality of the images so that the OCR process can produce better results. Steps 3 to 5 are an attempt to improve the accuracy of the OCR processing. The first OCR run is performed with the default settings, while the second run is performed on segmented parts of the text, using specific parametrization. The goal is to create an additional "opinion" of the software on what text should be recognized in those images. The sixth step is creating some form of comparison to decide on which opinion is the right one. Finally, the last step merges the results of the previous steps into a single searchable PDF file.

4.4.1 Development

The approach chosen to build this version was the same as the proof-of-concept, with the creation of scripts for each part of the process. These scripts were then coordinated by another main script, that would make the connection between them. Compared with the previous version, instead of six, eleven different scripts were created: one for each step of the process (*split.py*, *preprocess.py*, *segment.py*, *ocr.py*, *compare.py* and *merge.py*), four auxiliary scripts to keep track of the state of the process, manage the creation/deletion of folders/files and act as parsers (*docManager.py*, *folderManager.py*, *parser_r.py* and *parser_w.py*) and one main script to manage all the other scripts (*workflow.py*).
In addition to the scripts, and like the previous version, this workflow also followed a predefined directory structure, as seen in Figure 4.6.



```
▼ workflow
  ▶ docs
  ▼ exec
    ▶ tmp
      /* compare.py
      /* docManager.py
      /* folderManager.py
      /* merge.py
      /* ocr.py
      /* parser_r.py
      /* parser_w.py
      /* preprocess.py
      /* segment.py
      /* split.py
      /* workflow.py
  ▶ results
  ▶ tessdata
```

Figure 4.6: Final version directory structure

Compared to the previous version, one more folder was created, called tessdata. Trained data for specific languages being processed by the OCR, other than English should be placed here after being retrieved from the Tesseract repository.

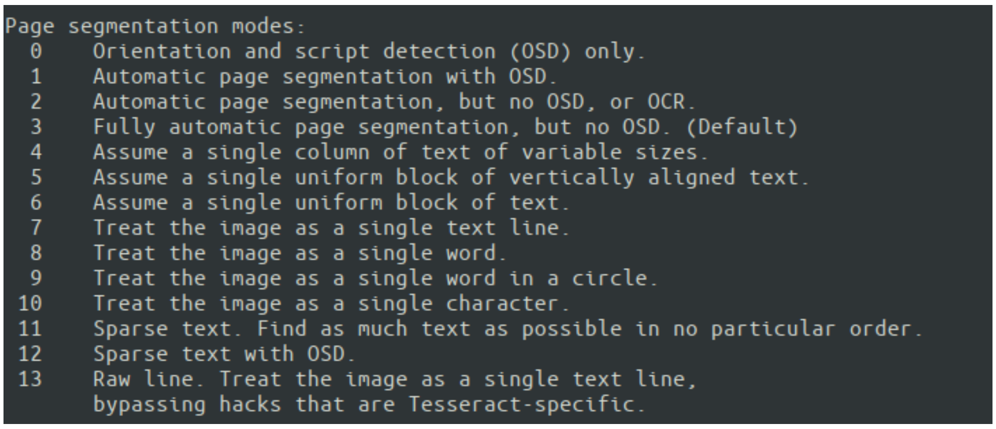
User interface

One of the goals when designing this workflow was to create a user interface. This interface should be easy to use and should not require any extraordinary IT knowledge to complete the tasks needed to run the workflow. However, during the design of this solution, cooperation with another master's dissertation emerged, with the intention of assisting in this project and giving their contribution in other fields of the workflow. Therefore, design and development decisions had to be made together with the others involved. One example of this collaboration is the creation of a user interface, which was no longer part of the scope of this dissertation. Therefore, the interaction with this version of the workflow is done through the command line.

tesserocr

In the first version of the workflow, the chosen tool to use Tesseract was the python library pytesseract. This library is just a wrapper of the tesseract-ocr command-line interface, which means that every time the function to recognize the text is called, it loads the model and processes the image. This library was not fast enough and did not allow much flexibility when choosing specific parameters.
After some research, another Python tool that dealt with the Tesseract was discovered. The name of

this tool is tesserocr, and it also consists of a Python wrapper, but this one is a wrapper around the Tesseract C++ API. By interfacing directly with the API, there is more flexibility, and other advanced features can be used. The trade-off is understanding its behavior since using it is more complex than the previous solution. Some features allowed with this tool are the introduction of new trained data for specific languages (more than 100 options available), the access to information and decisions being made by the OCR engine, or the definition of the page segmentation method to be used. The options available to segment the pages can be seen on Figure 4.7.



```
Page segmentation modes:
  0    Orientation and script detection (OSD) only.
  1    Automatic page segmentation with OSD.
  2    Automatic page segmentation, but no OSD, or OCR.
  3    Fully automatic page segmentation, but no OSD. (Default)
  4    Assume a single column of text of variable sizes.
  5    Assume a single uniform block of vertically aligned text.
  6    Assume a single uniform block of text.
  7    Treat the image as a single text line.
  8    Treat the image as a single word.
  9    Treat the image as a single word in a circle.
 10    Treat the image as a single character.
 11    Sparse text. Find as much text as possible in no particular order.
 12    Sparse text with OSD.
 13    Raw line. Treat the image as a single text line,
       bypassing hacks that are Tesseract-specific.
```

Figure 4.7: Available segmentation methods on tesserocr

4.4.2 Execution

As mentioned above, the execution of the final version is slightly different from the first version. Whilst the docs folder still exists, it is not mandatory to place the documents there. To run the workflow, the following command should be used in the Command Line:

```
$ python3 workflow.py path_to_file
```

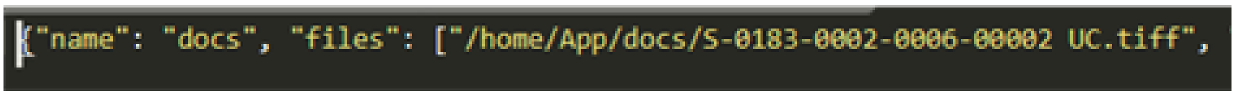
Another feature that was added to this version was the introduction of parametrization, being possible to define some parameters when running the workflow. In this particular version, there are seven different parameters that could be defined, but more can be added in the future. The available parameters can be seen in Appendix B.

When executing the previous command with or without any of the parameters, the workflow execution starts.

Prologue

The workflow starts its execution by creating a process (Figure 4.10). This process can correspond to the execution of a single file or the execution of several files in a specific folder, in case the –folder parameter was used.

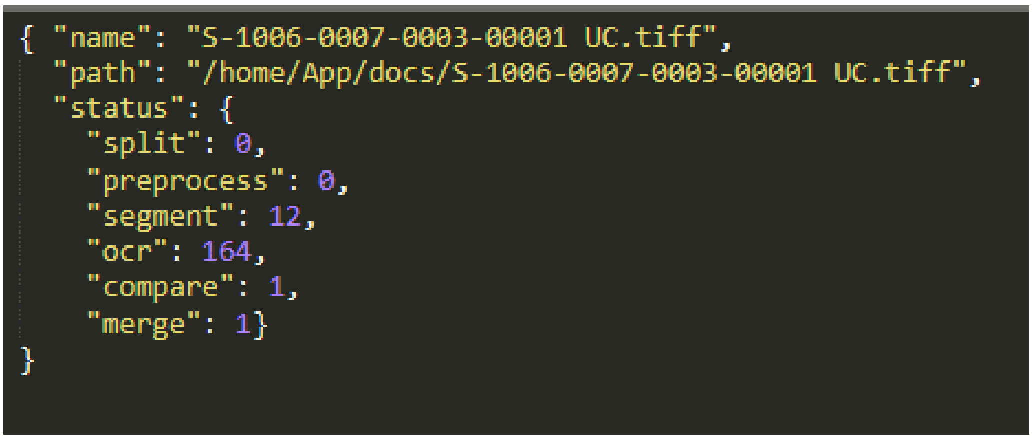
The creation of a process is characterized by the creation of a JSON file in the tmp folder, as well as the creation of a folder in the tmp and results folders, with the name of the original file (or the name of the original folder), which is also the name of the process by default. The JSON file contains only one entry with two different fields: the name of the process and the already processed files included in this process (which is empty at the beginning of the execution). An example of an entry can be seen in Figure 4.8.



```
{"name": "docs", "files": ["/home/App/docs/S-0183-0002-0006-00002 UC.tiff",
```

Figure 4.8: Example of a JSON entry

The purpose of this file is to keep the state of the workflow in case there is an interruption and to prevent overwriting in case the workflow is used again with the same files. Every time the workflow is started, it checks if a process with the same name already exists. If the answer is yes, it continues the process from where it stopped unless the –force parameter is used. In that case, all the progress until then is deleted, and the workflow starts from the beginning.
To keep track of the state of a specific file, another JSON file is created inside the corresponding folder of each process. There is one JSON file for each file in that process. This data file contains only one entry with the following fields: name of the file, path to the original file, and status. The status field contains six other fields corresponding to each step of the process, namely split, preprocess, segment, ocr, compare and merge. Figure 4.9 shows an example of an entry of the JSON file.



```
{ "name": "S-1006-0007-0003-00001 UC.tiff",
  "path": "/home/App/docs/S-1006-0007-0003-00001 UC.tiff",
  "status": {
    "split": 0,
    "preprocess": 0,
    "segment": 12,
    "ocr": 164,
    "compare": 1,
    "merge": 1}
}
```

Figure 4.9: Example of a JSON entry

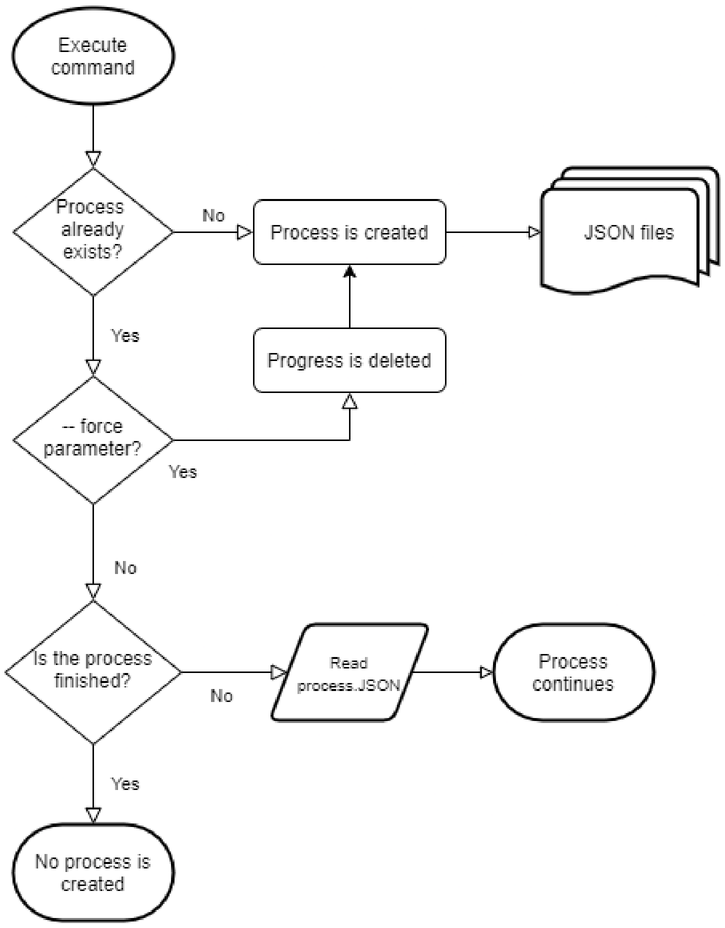


Figure 4.10: Prologue

Splitting

Like in the first version of the workflow, after the prologue, the first step is splitting the pages in case there is a multi-page TIFF image (Figure 4.11). This splitting is performed by the *split.py* script. The way it is done is also similar to the one in the proof-of-concept. The script uses the Python Image Library (PIL) to handle the multi-image TIFF files, specifically the Image module. With the *seek()* and *save()* functions, it is possible to split the TIFF multi-paged files into separated ones containing only one TIFF image per file. These files are saved, following the naming convention *"page_" + "#number of the page"* in the tmp folder.

Input: multi-paged TIFF image with n pages
Output: n TIFF images , with n= number of pages

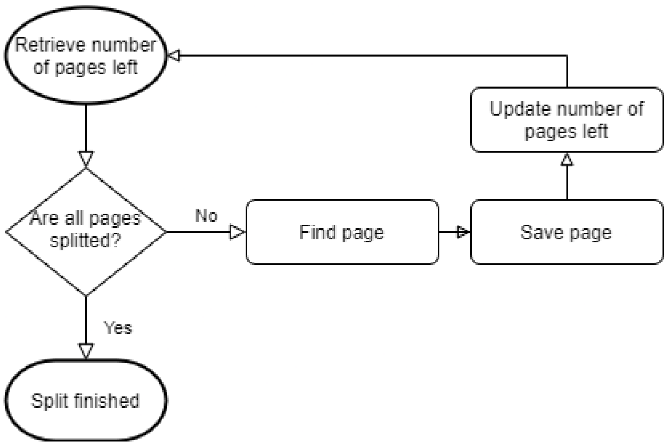


Figure 4.11: Splitting process

Pre-processing

One of the steps that was added to this version was pre-processing (Figure 4.14). In this step, performed by the *preprocess.py* script, additional pre-process techniques are applied to the split TIFF images. There are two options when running this script. In the default option, the thresholded images produced by the internal process of the Tesseract API are saved on the tmp folder. The other option occurs when the –prep parameter is used. When selecting this option, four additional techniques are applied, using the cv2 library, in the following order:

- Grayscale image: using the *cvtColor()* function, the color of the image is converted from RGB to grayscale;
- Noise removal: using the *medianBlur()* function that removes the noise in the image by applying a slight blur;
- Thresholding (additional): like the default option, this technique applies additional thresholding, using the function *threshold()*;

- Dilation: by using the *dilation()* function, it is possible to add pixels to the boundaries of objects in an image. This technique allows improving the quality of the text contained in the images

Examples of the performance of these techniques can be observed in Figure 4.12 and Figure 4.13. The images resulting from this step are the ones that are going to be used on the rest of the workflow, except the merge step, where the original images are the ones used.

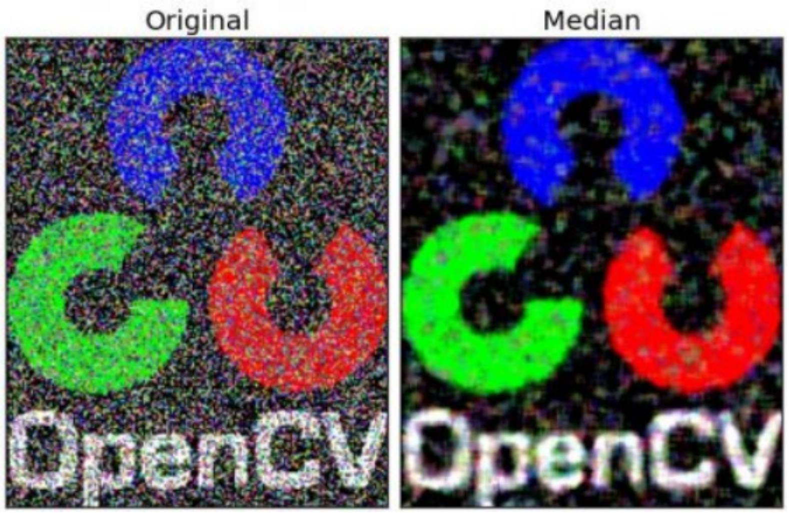


Figure 4.12: Noise removal

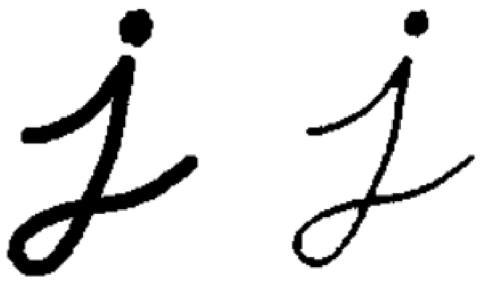


Figure 4.13: Before dilation (left) and after (right)

Input: n TIFF images

Output: n TIFF processed images

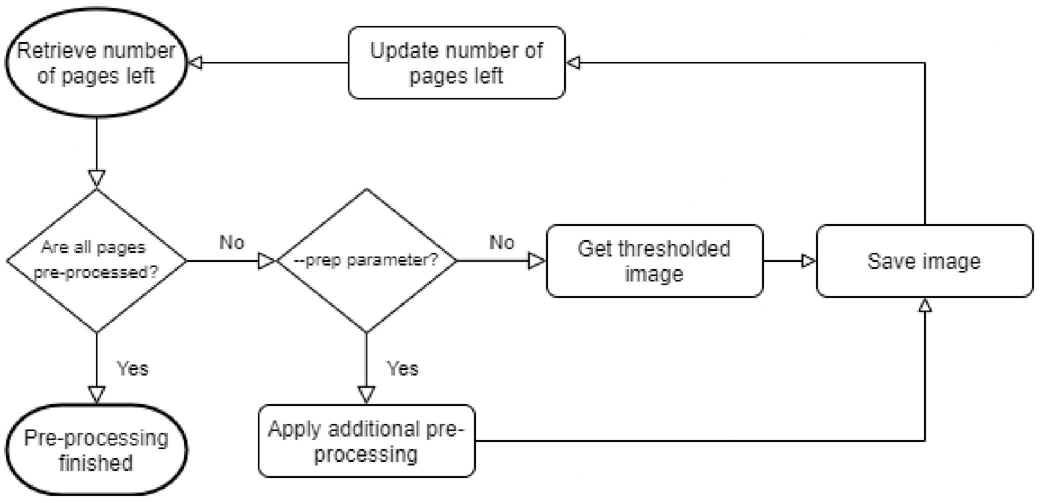

Retrieve number of pages left
Update number of pages left
Are all pages pre-processed?
No
--prep parameter?
No
Get thresholded image
Save image
Yes
Yes
Pre-processing finished
Apply additional pre-processing

Figure 4.14: Pre-processing process

Segmentation

This step that was also introduced in this new version and is performed by the *segment.py* script, can be divided into two different sub-steps (Figure 4.15). The first sub-step to perform the segmentation of the images is a first OCR run. This run is done with the default settings of the Tesseract API and results on an hOCR file for each one of the images corresponding to the different pages. These hOCR files are stored in the tmp folder.

In the second sub-step, with the information gathered in the hOCR files, all the text lines in the images are cropped and saved as separated images. After that, the information regarding all the recognized text lines is stored in a JSON file, called regions.JSON. There is an entry for each one of the images, each entry containing the corresponding information. The information gathered includes the id of the text line, the bounding box values (bbox property), the coordinates of the text line in the image, the path to the original image, the text contained in that line, and the path to the corresponding hOCR file. Additional fields like the recognized text and the words confidence are also created but remain empty until the next step. An example of an entry on the JSON file can be seen on Figure 4.16.

Input: TIFF image

Output: n hOCR files, with n=original TIFF images + n TIFF images, with n=number of segments found

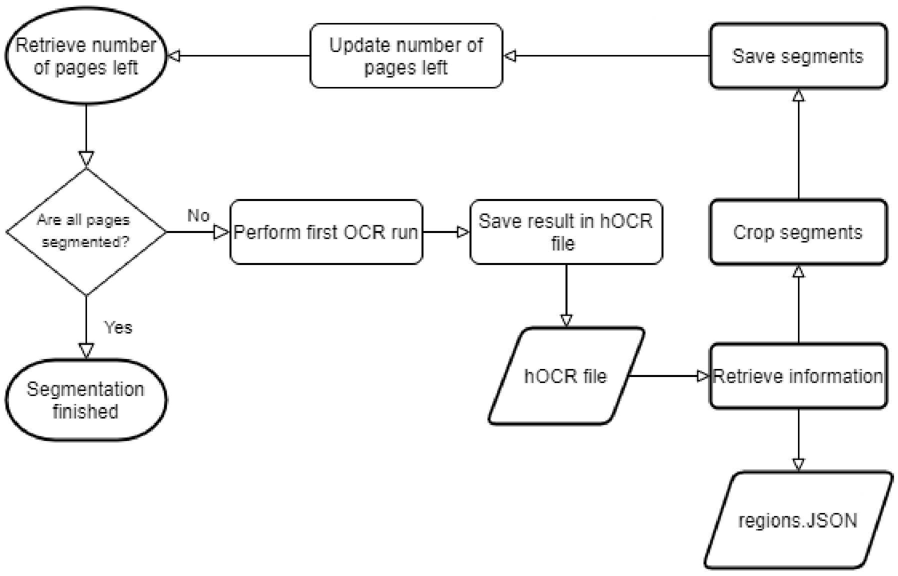
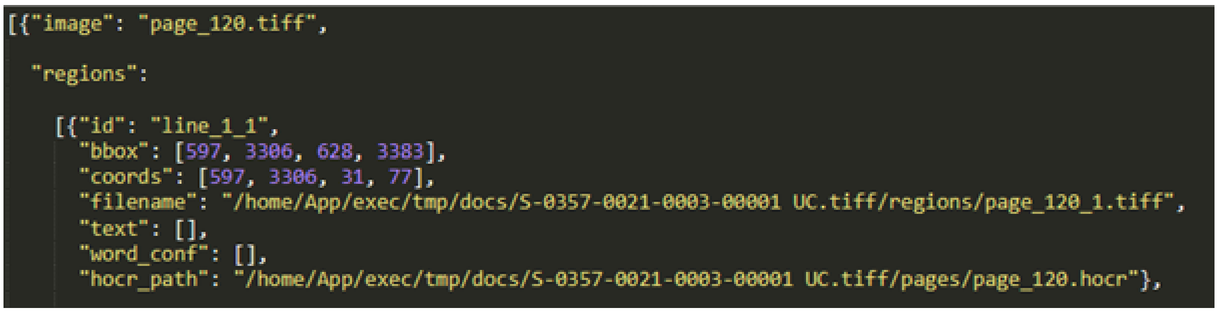


Figure 4.15: Segmentation process



```
[{"image": "page_120.tiff",

  "regions":

    [{"id": "line_1_1",
      "bbox": [597, 3306, 628, 3383],
      "coords": [597, 3306, 31, 77],
      "filename": "/home/App/exec/tmp/docs/S-0357-0021-0003-00001 UC.tiff/regions/page_120_1.tiff",
      "text": [],
      "word_conf": [],
      "hocr_path": "/home/App/exec/tmp/docs/S-0357-0021-0003-00001 UC.tiff/pages/page_120.hocr"},
```

Figure 4.16: Example of an entry on *regions.JSON* file

OCR

The OCR step (Figure 4.17), which is actually the second time the Tesseract API is called to do text recognition, is performed by the *ocr.py* script. This time the recognition is done on the text segments collected in the previous step. By calling the Tesseract API, with the "Page Segmentation Mode" parameter set to PSM.SINGLE_LINE, it is expected that the recognition results will be more accurate because the system is being told that the images being processed contain only one line of text. In addition to the recognized text, the corresponding word confidences are collected as well. In the end, the *regions.JSON* is updated on all the text and word_conf fields.

Input: n TIFF images, with n=number of segments + regions.JSON

Output: updated regions.JSON

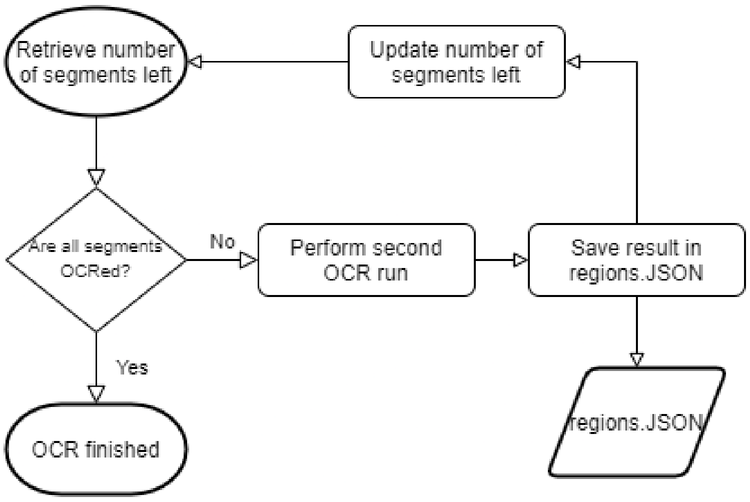

Retrieve number of segments left
Update number of segments left
Are all segments OCRed?
No
Perform second OCR run
Save result in regions.JSON
Yes
OCR finished
regions.JSON

Figure 4.17: OCR process

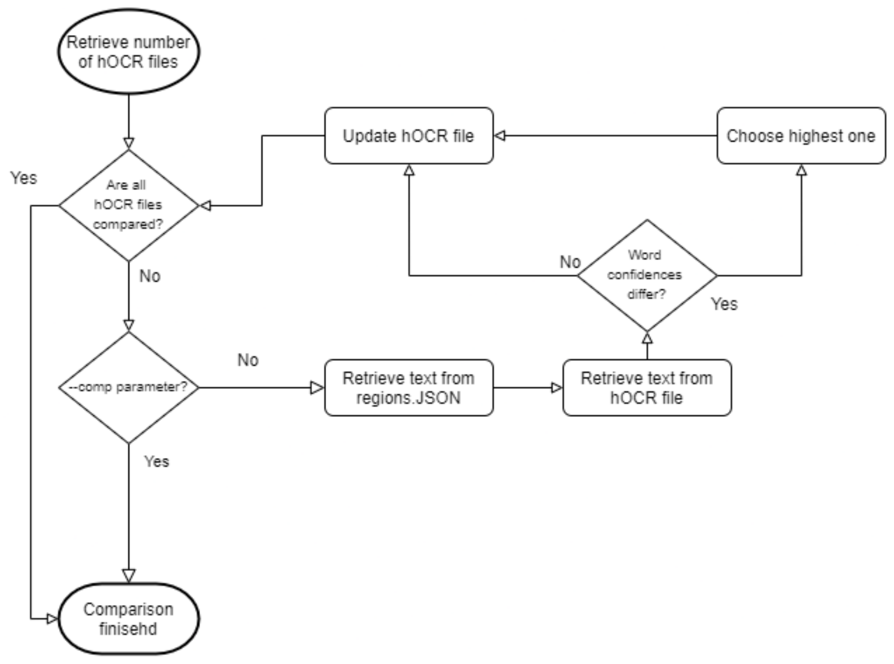
Comparison

Now that there are two different OCR runs, there is a necessity to compare which one contains the best results. To do so, there is a step called Comparison (Figure 4.18), performed by the *compare.py* script.
In this step, the text recognized in the first OCR run, made on the Segmentation step and stored in the hOCR files, and the text recognized in the previous run and stored in the *regions.JSON* file is compared. This script compares all the text lines found in both runs and checks if the word confidences are different between the corresponding bounding boxes. If the word confidences is different, the highest one is chosen. This leads to a change on the hOCR file, where the recognized text and their corresponding word confidences are updated.
To parse and update the hOCR file, two additional auxiliary scripts were created. The first one was the *parser_r.py*, that uses the bs4 library, usually used for scrapping information out of web pages. This script allowed parsing the content contained in the hOCR files. The second script was the *parser_w.py* that used the xml library, specifically the ElementTree module, used for creating XML data, to write into the hOCR files.
In the case where the –comp parameter was used, this step did not occur, and only the recognized text of the first OCR run would be considered. This parameter could be used in cases where time efficiency is more important than accuracy.

Input: n hOCR files + regions.JSON
Output: n updated hOCR files

Merging

Finally, the last step was merging all the data back into a single PDF (Figure 4.19). This step was performed by the *merge.py* script. This script is an adaptation of an open-source script that is part of



Retrieve number of hOCR files
Yes
Are all hOCR files compared?
No
Update hOCR file
Choose highest one
No
Word confidences differ?
Yes
No
--comp parameter?
Retrieve text from regions.JSON
Retrieve text from hOCR file
Yes
Comparison finisehd

Figure 4.18: Comparison process

the hocr-tools repository [1].

As opposed to the first version of the workflow, where the merge step only had to merge single page PDF files back into a multi-paged one, this time, the merge step works differently.

After the hOCR files being ready, their content is parsed again to gather the recognized text and its location on the image. Using the open-source version of the Report Lab tools, which includes a Python library, a PDF file with two layers is generated: the first layer is the original TIFF image, while the second layer is the recognized text in its correct position, written in an invisible font. This is done for each hOCR file and its corresponding TIFF image, creating the individual pages of the PDF file. In the end, a PDF file with all the pages is generated, resulting in the desired outcome of the workflow.

Input: n hOCR files + n original TIFF images

Output: PDF file

4.4.3 Deployment

As it was mentioned before, the environment of development of this solution was Linux. However, the workflow was designed to work in all platforms, specifically on Windows10, as stated in the requirements. When the deployment was done to this operating system to test its feasibility, it was found that one of the components of the workflow, although compatible with Windows machines, was too complex for someone with basic IT knowledge to install. This component that was part of the Tesseract software would not allow the workflow to be natively installed on ARMS computers.

[1] https://github.com/ocropus/hocr-tools

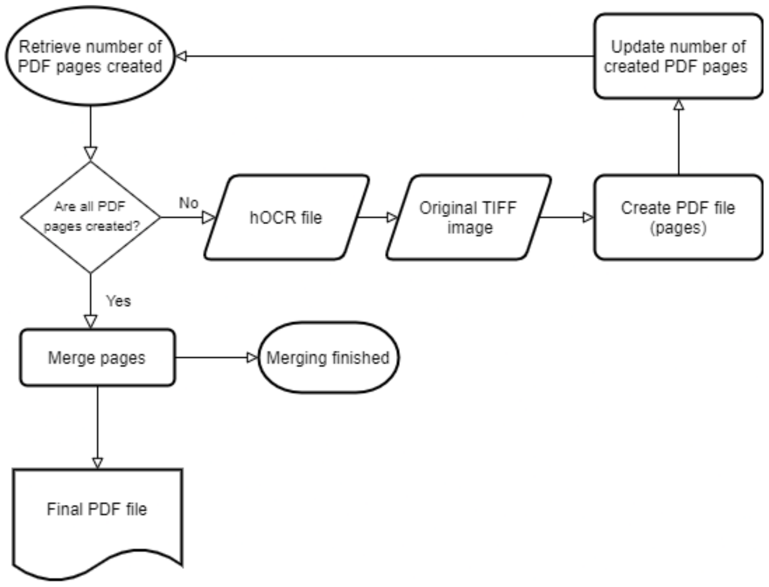


Figure 4.19: Merging process

To solve this setback, it was necessary to add another task to the development of the proposed solution: create a virtual environment to run the workflow. Taking this into account, and after considering between two solutions, the creation and configuration of a virtual machine or the creation of a Docker container, the latter was chosen.

After some research about this technology, a Dockerfile containing all the steps necessary to install and configure all the components of the workflow was produced, enabling the deployment of the workflow on Windows computers.

The Docker image, besides all the required software and packages that needed to be installed, also included the creation of a Docker volume, which is a file system mounted on the Docker container in order to preserve the data generated by the running container. In other words, it works as a shared folder between the virtual environment and the host computer. This feature is useful so that the users will not have to worry about transferring the images to be processed into the virtual environment, since they only must be placed in a specific folder and then be accessed by both the host and the virtual environment.

Chapter 5

Demonstration and assessment

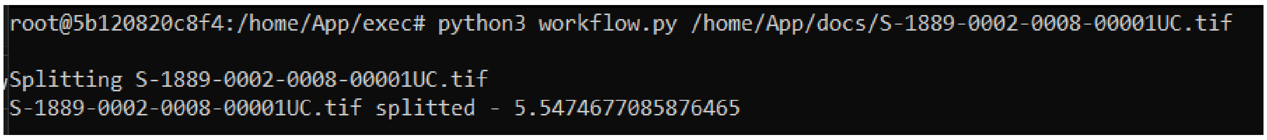
This chapter aims to demonstrate how the workflow works, as well as validate its results. This chapter is divided into two parts. In the first part, the processes described in the previous chapter will be demonstrated. In the second part, an evaluation of the results obtained during the testing of the developed workflow will be made. The workflow assessment was done in two ways: through tests in the development environment, both in terms of efficiency and effectiveness and through real-world use on ARMS computers. The results of these assessments will be presented in this chapter.

5.1 Demonstration

This section presents a demonstration of the behavior of the workflow during the different steps. If the results of each step are the desired ones, that step can be validated as working. In addition to the main steps, the auxiliary steps, such as the ability to keep the state of execution, will also be demonstrated and validated. The chosen image to validate the results is a multi-page TIFF with 13 pages and 74,7 MB.

5.1.1 Split a multi-page TIFF image

The first step to be validated is the ability to split multi-page TIFF images into individual ones. For this task to succeed, when running the workflow, the image given as input should be split into several pages. Those pages should then be stored in the corresponding sub-folder of the tmp folder. As it is possible to observe in Figure 5.1, after running the workflow, the first task that is performed is the splitting. It is also possible to observe on Figure 5.2 that the individual pages of the original image are stored independently.



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.5474677085876465
```

Figure 5.1: Splitting task

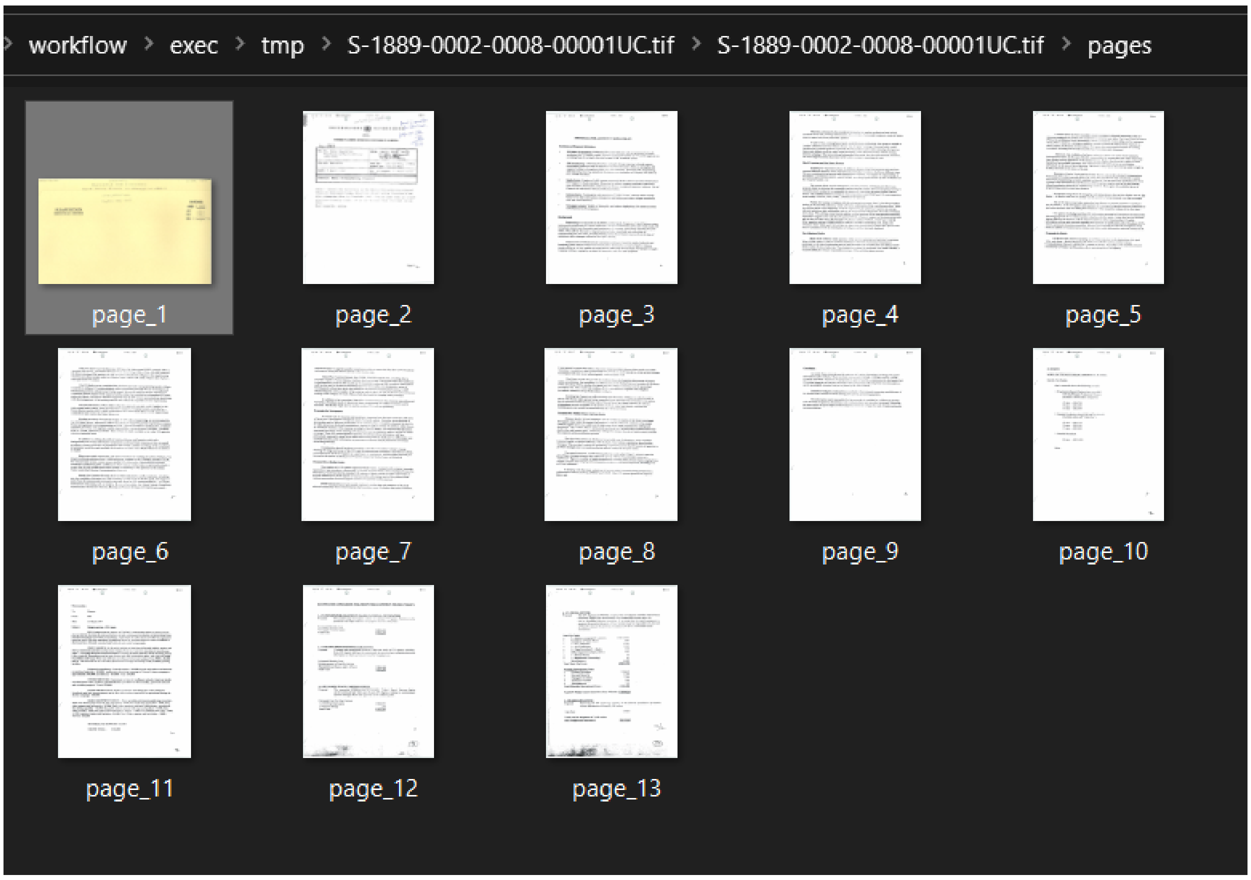
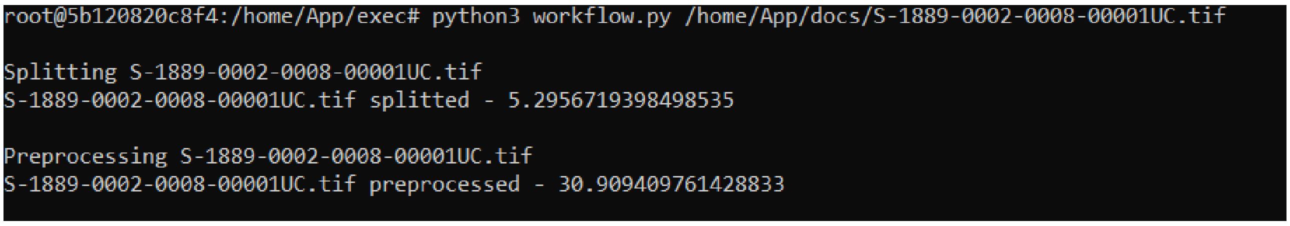

workflow › exec › tmp › S-1889-0002-0008-00001UC.tif › S-1889-0002-0008-00001UC.tif › pages
page_1
page_2
page_3
page_4
page_5
page_6
page_7
page_8
page_9
page_10
page_11
page_12
page_13

Figure 5.2: Result of splitting task

5.1.2 Pre-process an image - Simple

For the pre-processing to work correctly, the system should process the original images obtained in the previous step and produce a thresholded version of them.
In Figure 5.3, it is possible to see that the workflow performed the pre-processing step, and in Figure 5.4 the images were processed and stored in the correct place.



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.2956719398498535

Preprocessing S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif preprocessed - 30.909409761428833
```

Figure 5.3: Pre-processing task

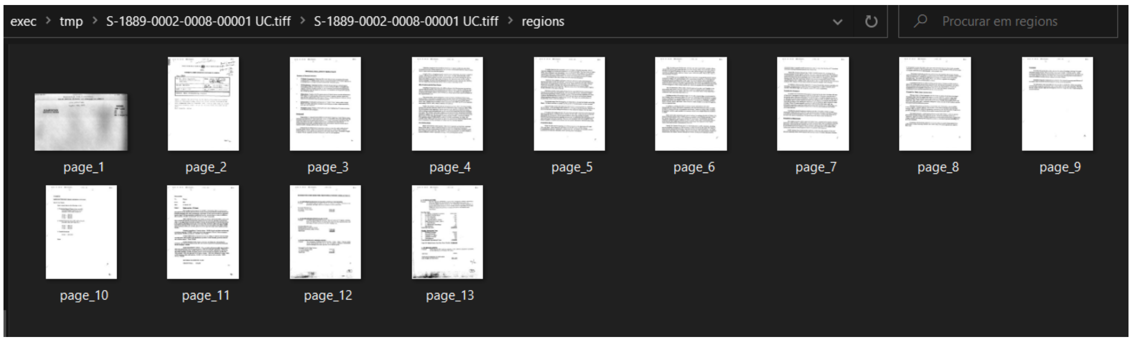
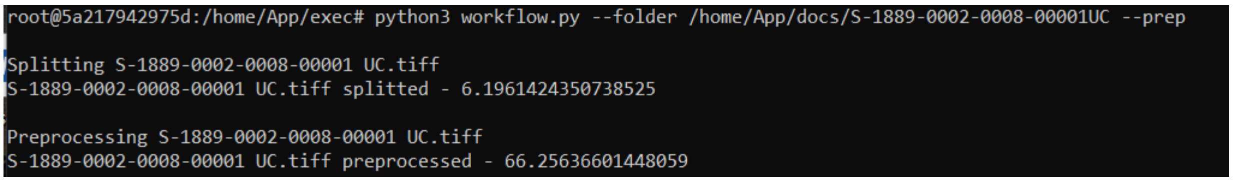


Figure 5.4: Result of pre-processing task

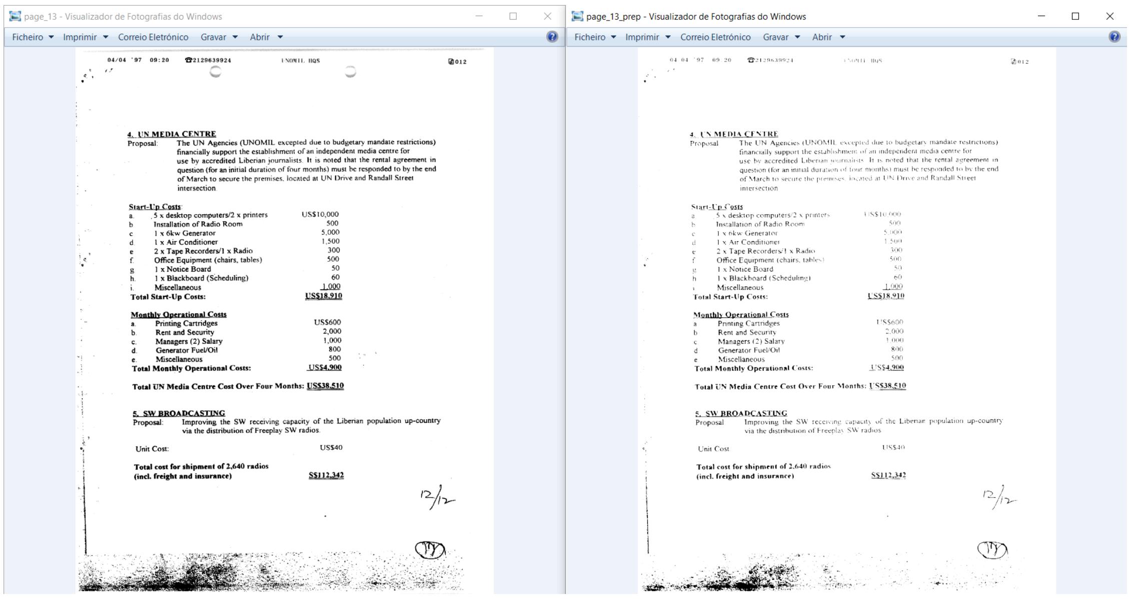
5.1.3 Pre-process an image - Additional

Similar to the previous task, the original image should be processed and stored when applying this step of the workflow. The difference is that the use of the –prep parameter enables additional processing on the images.
As seen on Figure 5.5, after introducing the correct command and the splitting step, the document applied the pre-processing step. To confirm that the pre-processing is different from the default one, the individual images have to be opened and checked. As observed in Figure 5.6, the image on the left (default pre-processing) and the image on the right (additional pre-processing) belong to the same original image but have different final versions.



```
root@5a217942975d:/home/App/exec# python3 workflow.py --folder /home/App/docs/S-1889-0002-0008-00001UC --prep

Splitting S-1889-0002-0008-00001 UC.tiff
S-1889-0002-0008-00001 UC.tiff splitted - 6.1961424350738525

Preprocessing S-1889-0002-0008-00001 UC.tiff
S-1889-0002-0008-00001 UC.tiff preprocessed - 66.25636601448059
```

Figure 5.5: Additional pre-processing task


page_13 - Visualizador de Fotografias do Windows
Ficheiro
Imprimir
Correio Eletrónico
Gravar
Abrir
page_13_prep - Visualizador de Fotografias do Windows

04/04 '97 09:20 ☎2129639924 UNOMIL HQS ☑012

4. UN MEDIA CENTRE
Proposal: The UN Agencies (UNOMIL excepted due to budgetary mandate restrictions) financially support the establishment of an independent media centre for use by accredited Liberian journalists. It is noted that the rental agreement in question (for an initial duration of four months) must be responded to by the end of March to secure the premises, located at UN Drive and Randall Street intersection.

Start-Up Costs:		
a.	5 x desktop computers/2 x printers	US$10,000
b	Installation of Radio Room	500
c	1 x 6kw Generator	5,000
d.	1 x Air Conditioner	1,500
e	2 x Tape Recorders/1 x Radio	300
f.	Office Equipment (chairs, tables)	500
g	1 x Notice Board	50
h.	1 x Blackboard (Scheduling)	60
i.	Miscellaneous	1,000
Total Start-Up Costs:		**US$18,910**

Monthly Operational Costs		
a.	Printing Cartridges	US$600
b.	Rent and Security	2,000
c.	Managers (2) Salary	1,000
d.	Generator Fuel/Oil	800
e.	Miscellaneous	500
Total Monthly Operational Costs:		**US$4,900**

Total UN Media Centre Cost Over Four Months: US$38,510

5. SW BROADCASTING
Proposal: Improving the SW receiving capacity of the Liberian population up-country via the distribution of Freeplay SW radios.

Unit Cost: US$40

Total cost for shipment of 2,640 radios (incl. freight and insurance) **S$112,342**

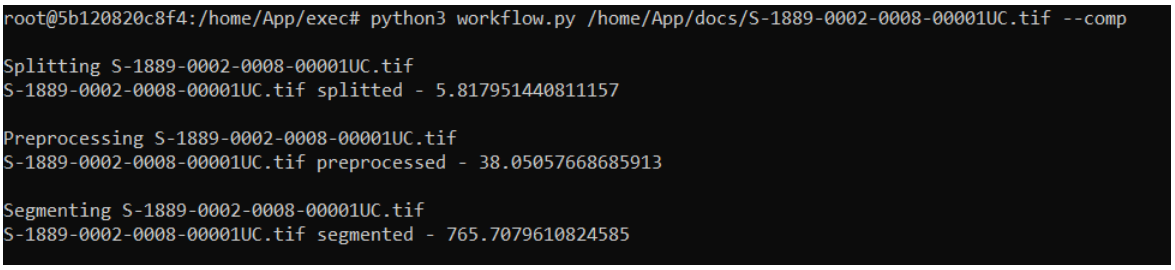
12/12

Figure 5.6: Result of additional pre-processing task

5.1.4 Segment an image

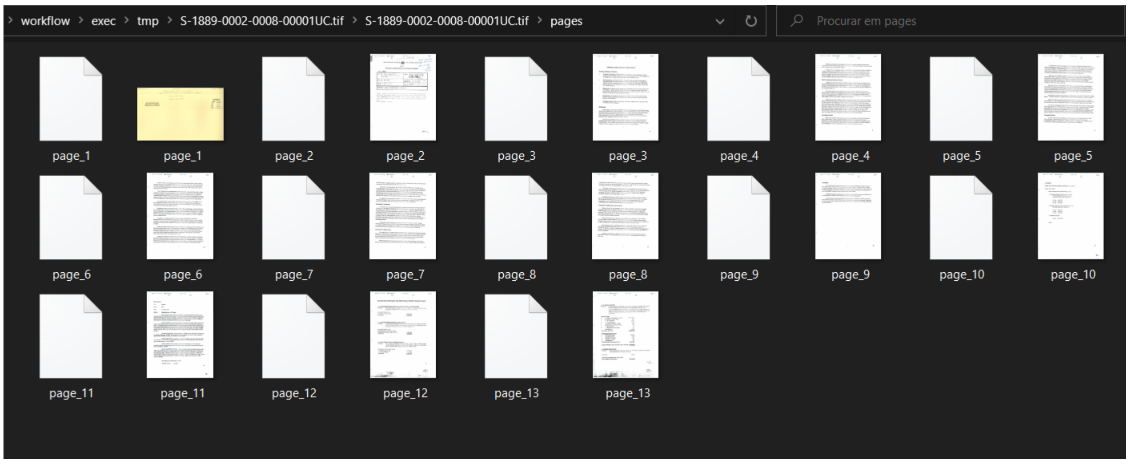
The next step to be validated is the segmentation of the images obtained in the previous step. For this step to be validated, three different tasks have to be performed correctly. First, a first OCR run is performed, which results in creating an hOCR file for each page. The second task is the segmentation of text lines present in the images, which result in cropping and storing those lines. The third task is creating the *regions.JSON* file where the information about the segmented images is stored

In Figure 5.7, it is possible to observe that the workflow is performing the segmentation step. At the same time, on the corresponding sub-folder of the tmp folder, the hOCR files are being generated, which validates the first task (Figure 5.8). In the other sub-folder, responsible for storing the segmentation results, it is possible to check that the segments are being cropped and stored (Figure 5.9). Thus, the second task is also validated. While those segments are being found, task 3 is being performed. The *regions.JSON* was already created and is being constantly updated. In Figure 5.10, it is possible to check the final version of the *regions.JSON* file.



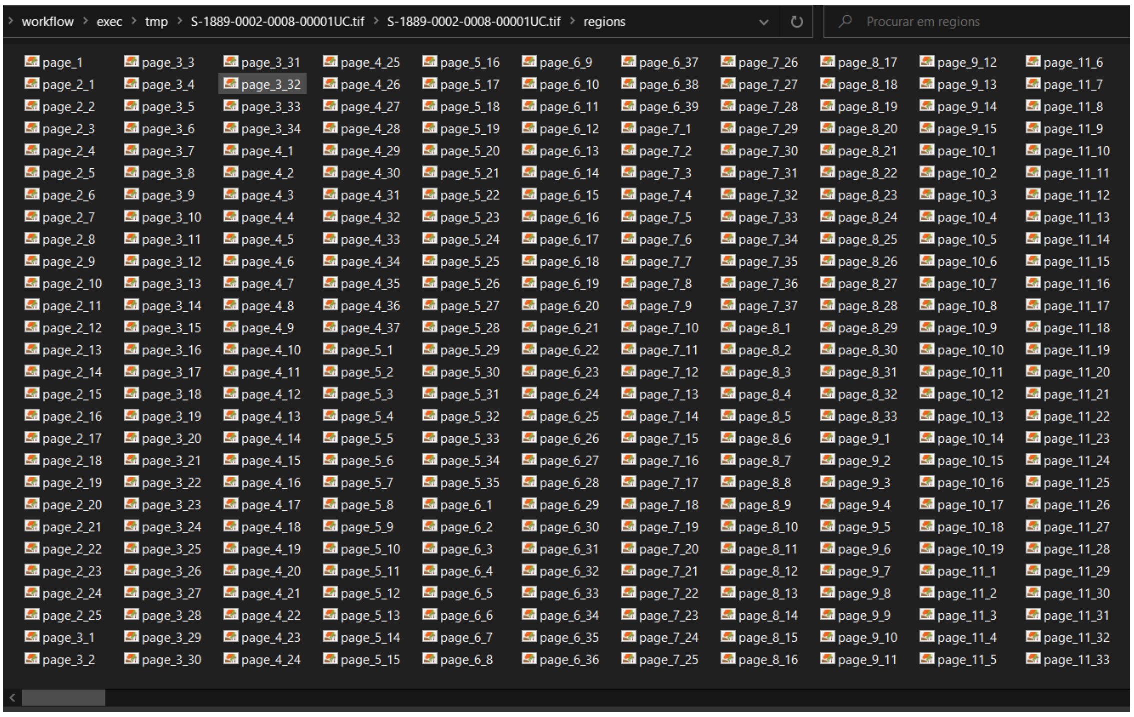
```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif --comp

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.817951440811157

Preprocessing S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif preprocessed - 38.05057668685913

Segmenting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif segmented - 765.7079610824585
```

Figure 5.7: Segmentation task



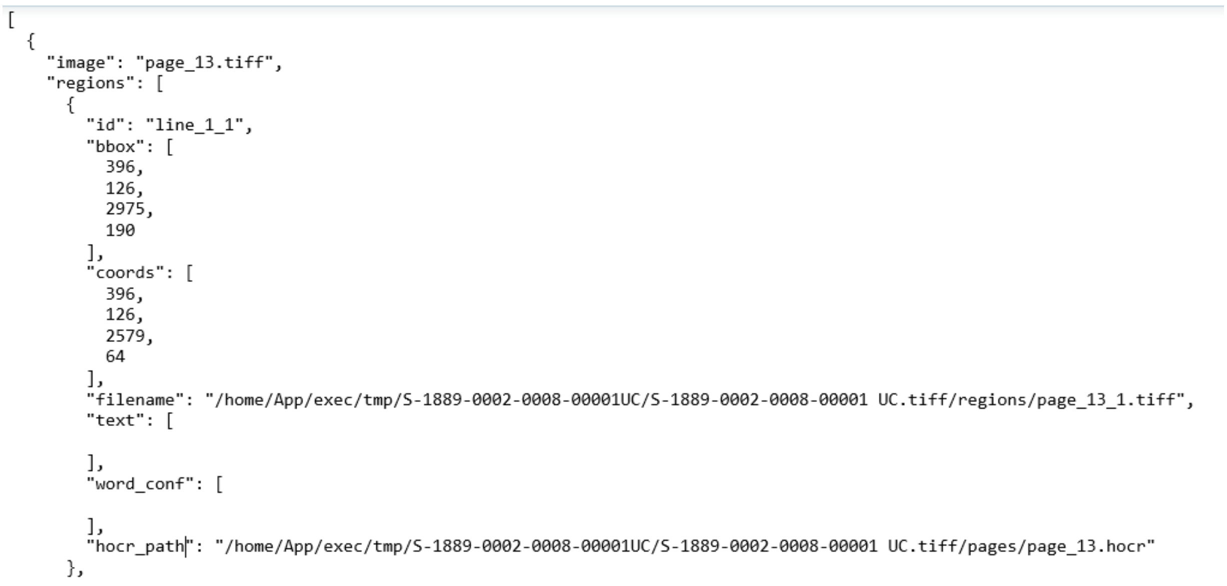
```
> workflow > exec > tmp > S-1889-0002-0008-00001UC.tif > S-1889-0002-0008-00001UC.tif > pages      Procurar em pages

page_1   page_1   page_2   page_2   page_3   page_3   page_4   page_4   page_5   page_5
page_6   page_6   page_7   page_7   page_8   page_8   page_9   page_9   page_10  page_10
page_11  page_11  page_12  page_12  page_13  page_13
```

Figure 5.8: First result of segmentation task



```
> workflow > exec > tmp > S-1889-0002-0008-00001UC.tif > S-1889-0002-0008-00001UC.tif > regions      Procurar em regions

page_1     page_3_3   page_3_31  page_4_25  page_5_16  page_6_9   page_6_37  page_7_26  page_8_17  page_9_12  page_11_6
page_2_1   page_3_4   page_3_32  page_4_26  page_5_17  page_6_10  page_6_38  page_7_27  page_8_18  page_9_13  page_11_7
page_2_2   page_3_5   page_3_33  page_4_27  page_5_18  page_6_11  page_6_39  page_7_28  page_8_19  page_9_14  page_11_8
page_2_3   page_3_6   page_3_34  page_4_28  page_5_19  page_6_12  page_7_1   page_7_29  page_8_20  page_9_15  page_11_9
page_2_4   page_3_7   page_4_1   page_4_29  page_5_20  page_6_13  page_7_2   page_7_30  page_8_21  page_10_1  page_11_10
page_2_5   page_3_8   page_4_2   page_4_30  page_5_21  page_6_14  page_7_3   page_7_31  page_8_22  page_10_2  page_11_11
page_2_6   page_3_9   page_4_3   page_4_31  page_5_22  page_6_15  page_7_4   page_7_32  page_8_23  page_10_3  page_11_12
page_2_7   page_3_10  page_4_4   page_4_32  page_5_23  page_6_16  page_7_5   page_7_33  page_8_24  page_10_4  page_11_13
page_2_8   page_3_11  page_4_5   page_4_33  page_5_24  page_6_17  page_7_6   page_7_34  page_8_25  page_10_5  page_11_14
page_2_9   page_3_12  page_4_6   page_4_34  page_5_25  page_6_18  page_7_7   page_7_35  page_8_26  page_10_6  page_11_15
page_2_10  page_3_13  page_4_7   page_4_35  page_5_26  page_6_19  page_7_8   page_7_36  page_8_27  page_10_7  page_11_16
page_2_11  page_3_14  page_4_8   page_4_36  page_5_27  page_6_20  page_7_9   page_7_37  page_8_28  page_10_8  page_11_17
page_2_12  page_3_15  page_4_9   page_4_37  page_5_28  page_6_21  page_7_10  page_8_1   page_8_29  page_10_9  page_11_18
page_2_13  page_3_16  page_4_10  page_5_1   page_5_29  page_6_22  page_7_11  page_8_2   page_8_30  page_10_10 page_11_19
page_2_14  page_3_17  page_4_11  page_5_2   page_5_30  page_6_23  page_7_12  page_8_3   page_8_31  page_10_11 page_11_20
page_2_15  page_3_18  page_4_12  page_5_3   page_5_31  page_6_24  page_7_13  page_8_4   page_8_32  page_10_12 page_11_21
page_2_16  page_3_19  page_4_13  page_5_4   page_5_32  page_6_25  page_7_14  page_8_5   page_8_33  page_10_13 page_11_22
page_2_17  page_3_20  page_4_14  page_5_5   page_5_33  page_6_26  page_7_15  page_8_6   page_9_1   page_10_14 page_11_23
page_2_18  page_3_21  page_4_15  page_5_6   page_5_34  page_6_27  page_7_16  page_8_7   page_9_2   page_10_15 page_11_24
page_2_19  page_3_22  page_4_16  page_5_7   page_5_35  page_6_28  page_7_17  page_8_8   page_9_3   page_10_16 page_11_25
page_2_20  page_3_23  page_4_17  page_5_8   page_6_1   page_6_29  page_7_18  page_8_9   page_9_4   page_10_17 page_11_26
page_2_21  page_3_24  page_4_18  page_5_9   page_6_2   page_6_30  page_7_19  page_8_10  page_9_5   page_10_18 page_11_27
page_2_22  page_3_25  page_4_19  page_5_10  page_6_3   page_6_31  page_7_20  page_8_11  page_9_6   page_10_19 page_11_28
page_2_23  page_3_26  page_4_20  page_5_11  page_6_4   page_6_32  page_7_21  page_8_12  page_9_7   page_11_1  page_11_29
page_2_24  page_3_27  page_4_21  page_5_12  page_6_5   page_6_33  page_7_22  page_8_13  page_9_8   page_11_2  page_11_30
page_2_25  page_3_28  page_4_22  page_5_13  page_6_6   page_6_34  page_7_23  page_8_14  page_9_9   page_11_3  page_11_31
page_3_1   page_3_29  page_4_23  page_5_14  page_6_7   page_6_35  page_7_24  page_8_15  page_9_10  page_11_4  page_11_32
page_3_2   page_3_30  page_4_24  page_5_15  page_6_8   page_6_36  page_7_25  page_8_16  page_9_11  page_11_5  page_11_33
```

Figure 5.9: Second result of segmentation task



```
[
  {
    "image": "page_13.tiff",
    "regions": [
      {
        "id": "line_1_1",
        "bbox": [
          396,
          126,
          2975,
          190
        ],
        "coords": [
          396,
          126,
          2579,
          64
        ],
        "filename": "/home/App/exec/tmp/S-1889-0002-0008-00001UC/S-1889-0002-0008-00001 UC.tiff/regions/page_13_1.tiff",
        "text": [

        ],
        "word_conf": [

        ],
        "hocr_path": "/home/App/exec/tmp/S-1889-0002-0008-00001UC/S-1889-0002-0008-00001 UC.tiff/pages/page_13.hocr"
      },
```

Figure 5.10: Third result of segmentation task

5.1.5 Perform second OCR run on the segmented image

After the segmentation and the first OCR run, the next step to be validated is the second OCR run. To be considered successful the segmented images have to suffer OCR processing, updating the fields “text” and “word_conf” in the *regions.JSON* file.
In Figure 5.11, it is visible that the workflow is performing the OCR step. Before it is concluded, the correspondent fields on the JSON file are empty (Figure 5.10). After this step is concluded, the “text” and “word_conf” fields are already filled (Figure 5.12).

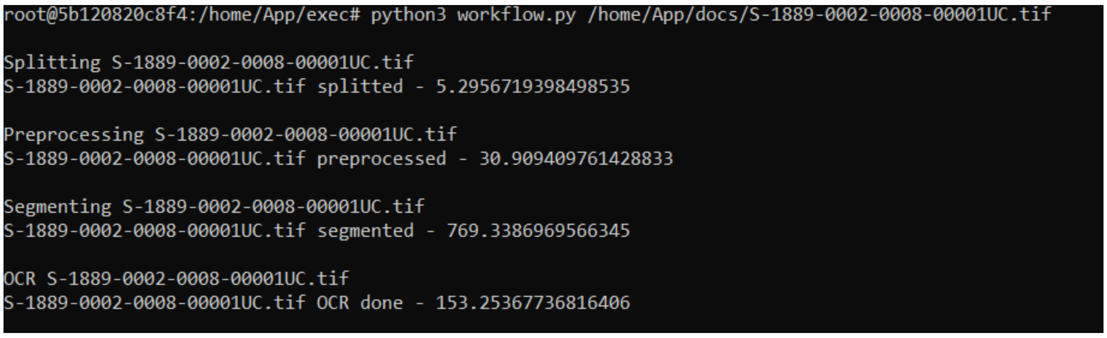
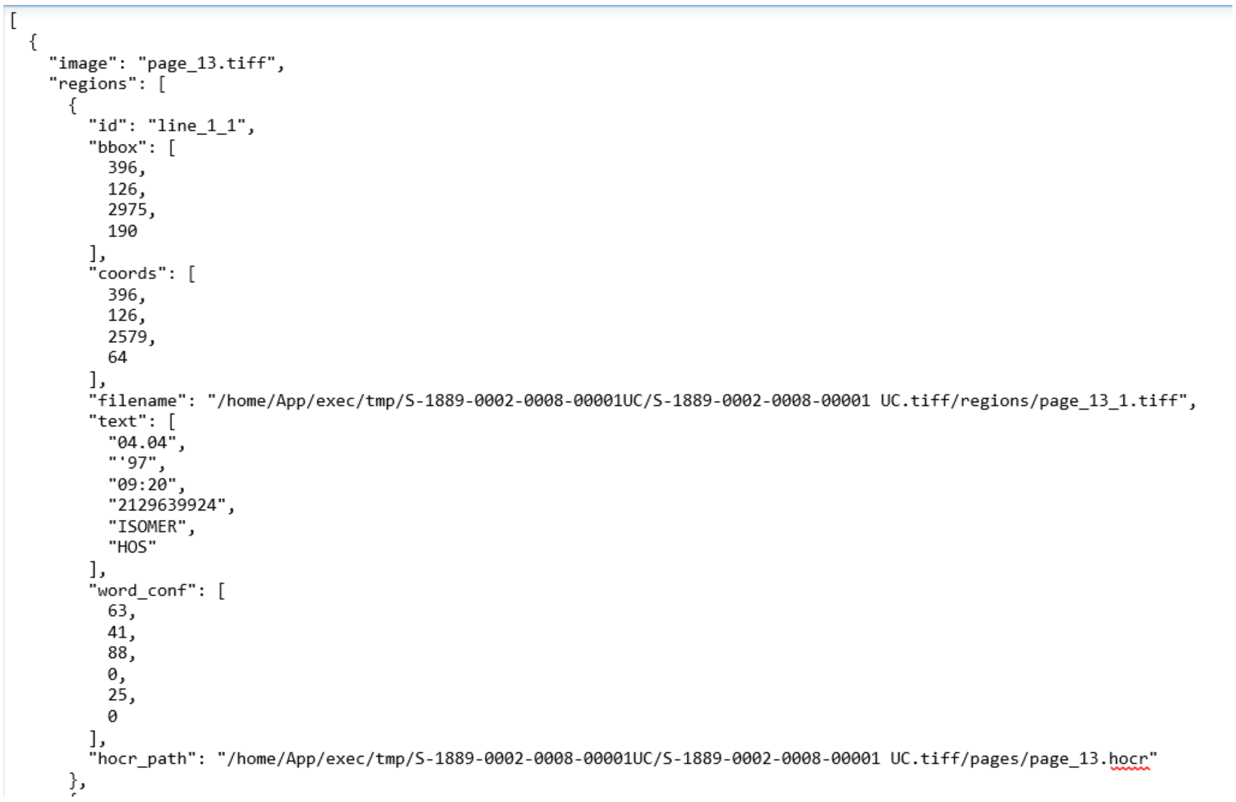

```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.2956719398498535

Preprocessing S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif preprocessed - 30.909409761428833

Segmenting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif segmented - 769.3386969566345

OCR S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif OCR done - 153.25367736816406
```

Figure 5.11: Second OCR task

5.1.6 Compare results from OCR

The fifth step to be validated is the comparison of both results obtained during the OCR runs. As described before, after the first and second runs of the OCR, it is necessary to compare both results to choose the best one. The first OCR run results are stored in the hOCR files, while the second OCR run



```
[
  {
    "image": "page_13.tiff",
    "regions": [
      {
        "id": "line_1_1",
        "bbox": [
          396,
          126,
          2975,
          190
        ],
        "coords": [
          396,
          126,
          2579,
          64
        ],
        "filename": "/home/App/exec/tmp/S-1889-0002-0008-00001UC/S-1889-0002-0008-00001 UC.tiff/regions/page_13_1.tiff",
        "text": [
          "04.04",
          "'97",
          "09:20",
          "2129639924",
          "ISOMER",
          "HOS"
        ],
        "word_conf": [
          63,
          41,
          88,
          0,
          25,
          0
        ],
        "hocr_path": "/home/App/exec/tmp/S-1889-0002-0008-00001UC/S-1889-0002-0008-00001 UC.tiff/pages/page_13.hocr"
      },
```

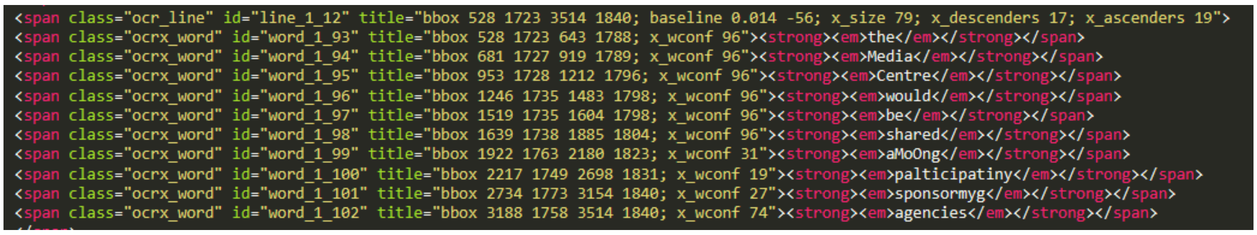
Figure 5.12: Result of second OCR task

results are stored in the *regions.JSON*. For this step to be considered successful, it is necessary to find one text line where the result of both OCR runs differ. If this step works, the result with the highest value of word confidence is the chosen one.

It is essential to mention that what is being evaluated in this step is not the OCR accuracy but if the criteria to choose the best results (in this case, the highest word confidence value) is performing well.

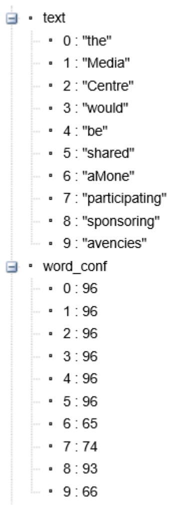
In Figure 5.13, the hOCR result shows that in this specific line of text, the word found with the id="word1_100" was 'palticipatiny', and the correspondent word confidence was 19. In Figure 5.14, it is possible to observe the result obtained in the second OCR run for the same text line. Finally, in Figure 5.15 is visible that the workflow just performed the comparison step. If this step was successful, the text line result mentioned above should have been changed in the hOCR file.

As seen on Figure 5.16, the result was changed, and the step can be validated.



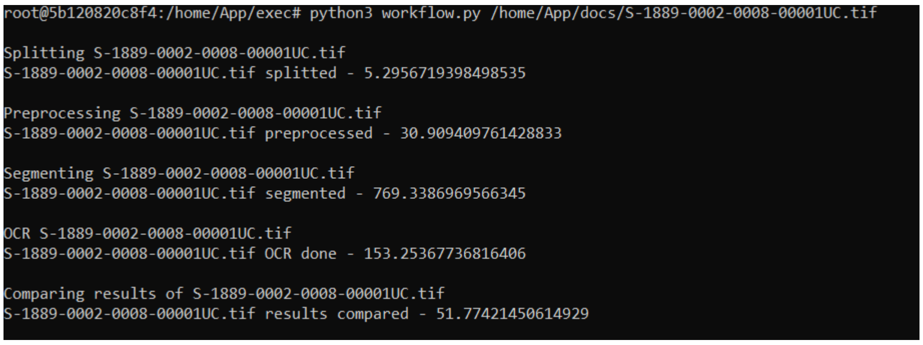
```
<span class="ocr_line" id="line_1_12" title="bbox 528 1723 3514 1840; baseline 0.014 -56; x_size 79; x_descenders 17; x_ascenders 19">
<span class="ocrx_word" id="word_1_93" title="bbox 528 1723 643 1788; x_wconf 96"><strong><em>the</em></strong></span>
<span class="ocrx_word" id="word_1_94" title="bbox 681 1727 919 1789; x_wconf 96"><strong><em>Media</em></strong></span>
<span class="ocrx_word" id="word_1_95" title="bbox 953 1728 1212 1796; x_wconf 96"><strong><em>Centre</em></strong></span>
<span class="ocrx_word" id="word_1_96" title="bbox 1246 1735 1483 1798; x_wconf 96"><strong><em>would</em></strong></span>
<span class="ocrx_word" id="word_1_97" title="bbox 1519 1735 1604 1798; x_wconf 96"><strong><em>be</em></strong></span>
<span class="ocrx_word" id="word_1_98" title="bbox 1639 1738 1885 1804; x_wconf 96"><strong><em>shared</em></strong></span>
<span class="ocrx_word" id="word_1_99" title="bbox 1922 1763 2180 1823; x_wconf 31"><strong><em>aMoOng</em></strong></span>
<span class="ocrx_word" id="word_1_100" title="bbox 2217 1749 2698 1831; x_wconf 19"><strong><em>palticipatiny</em></strong></span>
<span class="ocrx_word" id="word_1_101" title="bbox 2734 1773 3154 1840; x_wconf 27"><strong><em>sponsormyg</em></strong></span>
<span class="ocrx_word" id="word_1_102" title="bbox 3188 1758 3514 1840; x_wconf 74"><strong><em>agencies</em></strong></span>
```

Figure 5.13: Result from first OCR task



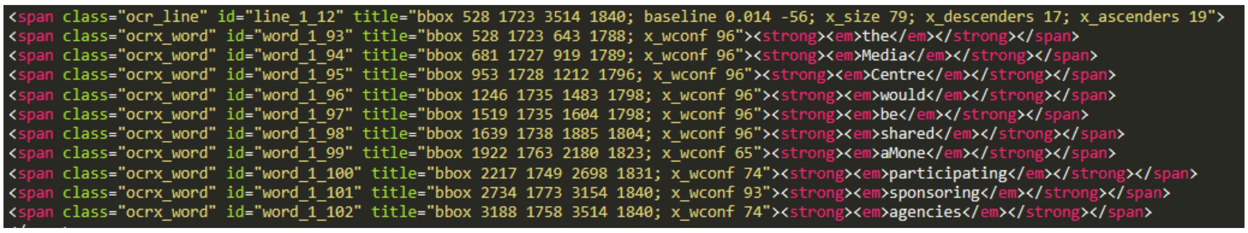
```
text
  0 : "the"
  1 : "Media"
  2 : "Centre"
  3 : "would"
  4 : "be"
  5 : "shared"
  6 : "aMone"
  7 : "participating"
  8 : "sponsoring"
  9 : "avencies"
word_conf
  0 : 96
  1 : 96
  2 : 96
  3 : 96
  4 : 96
  5 : 96
  6 : 65
  7 : 74
  8 : 93
  9 : 66
```

Figure 5.14: Result from second OCR task



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.2956719398498535

Preprocessing S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif preprocessed - 30.909409761428833

Segmenting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif segmented - 769.3386969566345

OCR S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif OCR done - 153.25367736816406

Comparing results of S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif results compared - 51.77421450614929
```

Figure 5.15: Compare results from OCR task

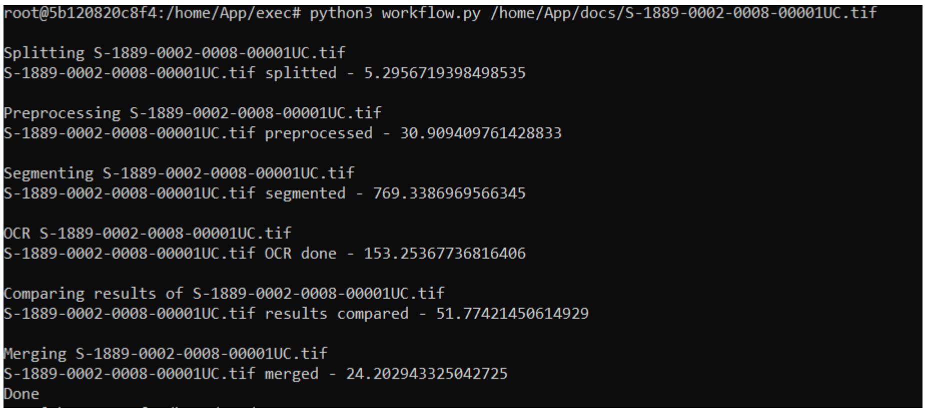


```
<span class="ocr_line" id="line_1_12" title="bbox 528 1723 3514 1840; baseline 0.014 -56; x_size 79; x_descenders 17; x_ascenders 19">
<span class="ocrx_word" id="word_1_93" title="bbox 528 1723 643 1788; x_wconf 96"><strong><em>the</em></strong></span>
<span class="ocrx_word" id="word_1_94" title="bbox 681 1727 919 1789; x_wconf 96"><strong><em>Media</em></strong></span>
<span class="ocrx_word" id="word_1_95" title="bbox 953 1728 1212 1796; x_wconf 96"><strong><em>Centre</em></strong></span>
<span class="ocrx_word" id="word_1_96" title="bbox 1246 1735 1483 1798; x_wconf 96"><strong><em>would</em></strong></span>
<span class="ocrx_word" id="word_1_97" title="bbox 1519 1735 1604 1798; x_wconf 96"><strong><em>be</em></strong></span>
<span class="ocrx_word" id="word_1_98" title="bbox 1639 1738 1885 1804; x_wconf 96"><strong><em>shared</em></strong></span>
<span class="ocrx_word" id="word_1_99" title="bbox 1922 1763 2180 1823; x_wconf 65"><strong><em>aMone</em></strong></span>
<span class="ocrx_word" id="word_1_100" title="bbox 2217 1749 2698 1831; x_wconf 74"><strong><em>participating</em></strong></span>
<span class="ocrx_word" id="word_1_101" title="bbox 2734 1773 3154 1840; x_wconf 93"><strong><em>sponsoring</em></strong></span>
<span class="ocrx_word" id="word_1_102" title="bbox 3188 1758 3514 1840; x_wconf 74"><strong><em>agencies</em></strong></span>
```

Figure 5.16: Result of compare results from OCR task

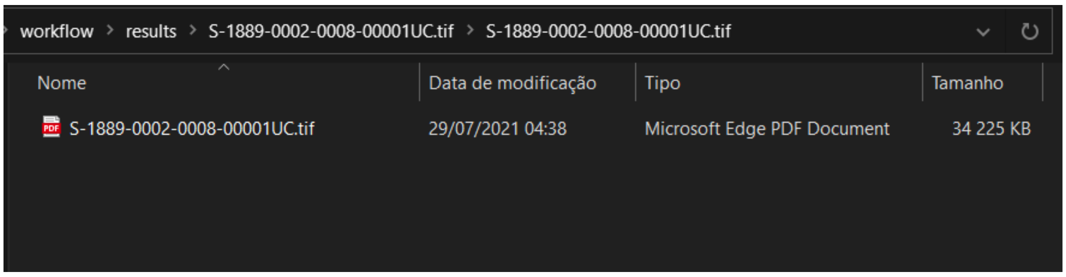
5.1.7 Merge results

The final step of the workflow is merging the results into a single PDF file. To perform this step, the hOCR files and the original images, obtained in the splitting phase, are merged into a PDF file with two layers: text and the original image. As observed in Figure 5.17, the workflow is performing the merging step. The source files and images to perform the merge are stored in the respective sub-folder of the tmp folder. After the merge is completed, all the source files and images are deleted, and a PDF file with the same name as the original document is generated in the results folder (Figure 5.18).



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.2956719398498535

Preprocessing S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif preprocessed - 30.909409761428833

Segmenting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif segmented - 769.3386969566345

OCR S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif OCR done - 153.25367736816406

Comparing results of S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif results compared - 51.77421450614929

Merging S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif merged - 24.202943325042725
Done
```

Figure 5.17: Merging task



workflow > results > S-1889-0002-0008-00001UC.tif > S-1889-0002-0008-00001UC.tif

Nome	Data de modificação	Tipo	Tamanho
S-1889-0002-0008-00001UC.tif	29/07/2021 04:38	Microsoft Edge PDF Document	34 225 KB

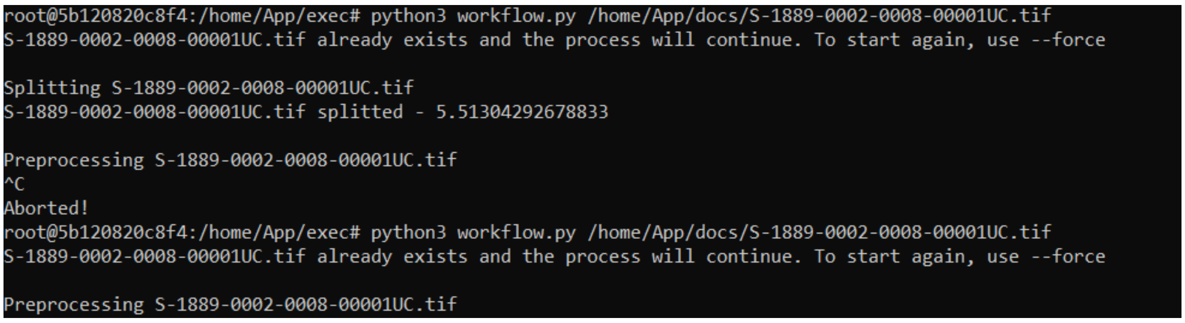
Figure 5.18: Result of merging task

5.1.8 Continue from interruption

After demonstrating and validating the main steps of the workflow, it is also essential to check if some of the other features are functioning. One of those features is the ability of the workflow to keep the state of execution so that after the workflow is interrupted, it is able to continue from where it stopped.
In Figure 5.19, it is possible to see that the workflow was interrupted during the pre-processing step. After that, if the workflow is started again, it should continue from the pre-processing step instead of starting from the beginning.
Thus, the behavior was the expected one, and this step can be validated.

5.1.9 Forced start

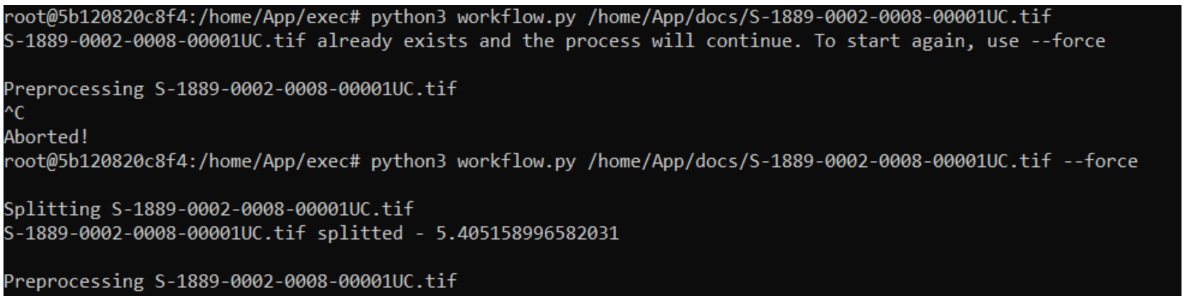
In contrast to the previous task, where the workflow must continue from where it stopped, to validate this one, the workflow should start the processing from the beginning, even if it was at an advanced stage.



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif already exists and the process will continue. To start again, use --force

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.51304292678833

Preprocessing S-1889-0002-0008-00001UC.tif
^C
Aborted!
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif already exists and the process will continue. To start again, use --force

Preprocessing S-1889-0002-0008-00001UC.tif
```

Figure 5.19: Continue from interruption task

Figure 5.20 shows that the workflow was interrupted during the pre-processing step. When the command to run the workflow is introduced again, this time with the –force flag, the workflow starts from the beginning, validating this task.



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif already exists and the process will continue. To start again, use --force

Preprocessing S-1889-0002-0008-00001UC.tif
^C
Aborted!
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif --force

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.405158996582031

Preprocessing S-1889-0002-0008-00001UC.tif
```

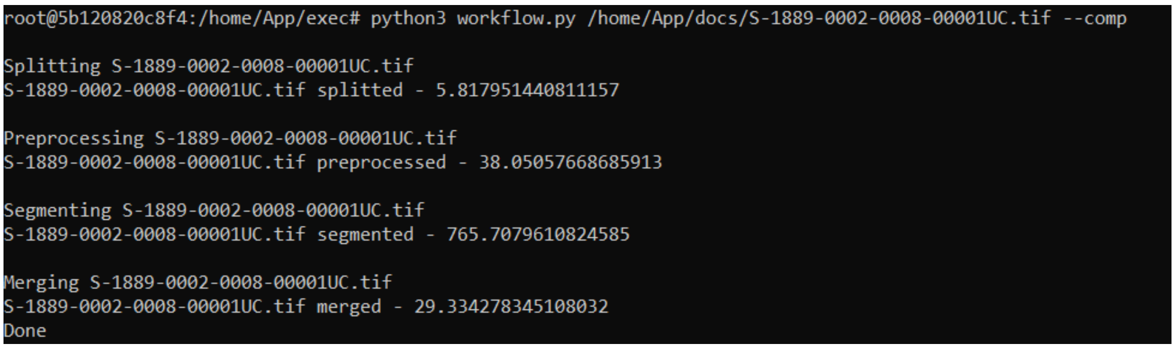
Figure 5.20: Forced start task

5.1.10 Skip comparison

The last task to be validated is the proper functioning of the –comp parameter, which allows the workflow to skip the OCR and comparison steps. This means that when this parameter is used, only one OCR run is performed. Therefore, the workflow should only execute the splitting, pre-processing, segmentation, and merge steps to validate this task.
On Figure 5.21, which shows the execution of the workflow with the –comp parameter, it is possible to see that both the OCR and the comparison steps were skipped, and for that reason, this task can be validated.

5.2 Assessment

The assessment of the developed workflow can be split into two different phases. First, after the development ended, the workflow was deployed and installed in the ARMS computers to test its feasibility in a real-world environment. Second, at the same time, another phase to evaluate the workflow was



```
root@5b120820c8f4:/home/App/exec# python3 workflow.py /home/App/docs/S-1889-0002-0008-00001UC.tif --comp

Splitting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif splitted - 5.817951440811157

Preprocessing S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif preprocessed - 38.05057668685913

Segmenting S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif segmented - 765.7079610824585

Merging S-1889-0002-0008-00001UC.tif
S-1889-0002-0008-00001UC.tif merged - 29.334278345108032
Done
```

Figure 5.21: Skip comparison task

performed in the same environment as it was developed. This evaluation was performed both in terms of efficiency and effectiveness.

5.2.1 Real-world environment

The assessment on a real-world environment is fundamental since it is this assessment that will tell if the workflow can be used by ARMS.

Installation and setup

To perform this evaluation, two sessions were scheduled to install the workflow. One session with a staff member responsible for digitizing and processing the digitized images (next referred to as User1), and another session with a staff member responsible for other tasks, not directly related to the processing of the digitized images (referred to as User2). Prior to these sessions, the code was made available in a GitHub repository, together with a user guide prepared specifically for this purpose.
In addition to the source code, the only thing that was required to install previously was the Docker desktop software and the Linux kernel update package.
The sessions to install the workflow were performed using the shared screen feature of the Microsoft Teams software. To assess how easy it was to install it, the users were required to try to perform the installation by themselves, following the steps on the user guide. However, if there were questions, they could be answered to ensure that the workflow would work.
The first session with User1 took about thirty minutes. Issues related to the installation and setup of the Docker desktop software in ARMS computers were the reason why it took so long. These computers have a strict policy on the type of software that can be installed, for security reasons, so administrator privileges were needed.
Another issue raised during that installation session was that the first version of the user guide required the user to introduce some commands on the Windows command line. Since the user did not have enough IT knowledge to perform those actions, two bash scripts were prepared to ease its work. One bash script to build the Docker image from the Dockerfile, that only has to be used once, and another bash script to run the Docker, that has to run every time the workflow is used.

For the second session, the user guide was changed to introduce the recently created bash scripts. After these changes and with the previous experience obtained in the previous session, it was easier to solve the issues that could arise during the installation and setup of the workflow. For that reason, the session took less time, being completed in about fifteen minutes.
The use of Docker containers to deploy the workflow ended up being a good idea since it turns the process a lot easier, taking less time to do it. In addition, instead of having to install all the libraries and dependencies, only the Docker software has to be installed.

Testing

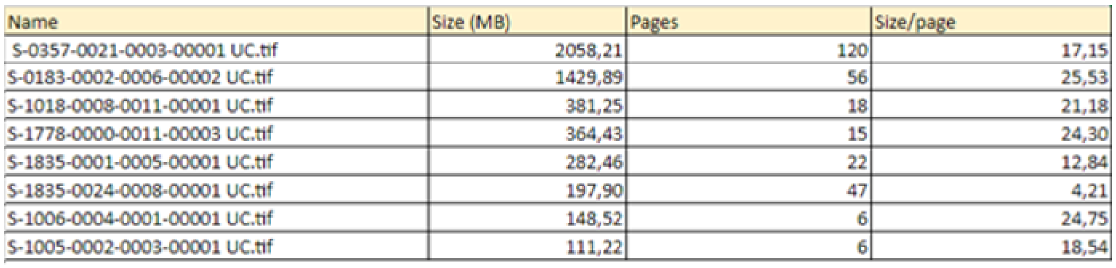
The testing of the workflow in a real-world environment was performed for two weeks. Unfortunately, due to delays in the elaboration of this thesis, it was not possible to perform extensive testing with an appropriate results collection.
The workflow was tested with a few scanned documents by User1, and the feedback was given by e-mail and through one of the biweekly meetings.
The user reported two different bugs on the execution of the workflow and one processing error. Both bugs were related to problems in the development of the workflow and were quickly fixed. The processing error happened during the processing of one of the samples, and it was caused by one faulty TIFF image that was causing the workflow to crash.
Positive feedback was also given, and accordingly, to the user who performed the tests, the OCR results are better than those obtained by the previously used software, Adobe Acrobat Pro DC.
In addition to that, the user also stated that running the workflow without the –prep parameter produced very bad results when compared with the results obtained with that parameter. However, the option to use the workflow without this parameter was kept for when the scanned documents are already legible and do not need that extra step.

5.2.2 Development environment

Besides the functional assessment made on real-world environment, a more extensive testing had to be performed to measure the performance of the proposed solution. For this purposed, a batch of files, containing TIFF images, were prepared, and made available by the ARMS team, to be tested during and after the development of the workflow.
The performance was measured in two different ways. It was measured in terms of efficiency, where the chosen metric was the time each step takes to process. It was also measured in terms of effectiveness, where the OCR accuracy is assessed based on what the expected result is. The environment used to obtain the results was a machine running Windows10, with an Intel(R) Core (TM) i7-10510U CPU @ 1.80GHz 2.30 GHz and 16GB RAM. The workflow was running inside a Docker container, running Ubuntu18.
The results on the assessment of the efficiency and effectiveness of the developed workflow are presented in the next two subsections.

Efficiency

As stated above, the chosen metric to measure the efficiency of the proposed solution was the time the workflow took to perform each task. For this evaluation, nine different files were chosen from the available batch. Although it is difficult to obtain a set of documents that represent the majority of the documents usually processed by ARMS, these documents were chosen based on their size and number of pages, with the objective of obtaining a greater representation of what the documents processed by the workflow may be. The size of each file (in megabytes) and their number of pages can be seen in Table 5.1. As it is possible to observe, the documents vary from 111,22MB and 6 pages to 2058,21MB and 120 pages.



Name	Size (MB)	Pages	Size/page
S-0357-0021-0003-00001 UC.tif	2058,21	120	17,15
S-0183-0002-0006-00002 UC.tif	1429,89	56	25,53
S-1018-0008-0011-00001 UC.tif	381,25	18	21,18
S-1778-0000-0011-00003 UC.tif	364,43	15	24,30
S-1835-0001-0005-00001 UC.tif	282,46	22	12,84
S-1835-0024-0008-00001 UC.tif	197,90	47	4,21
S-1006-0004-0001-00001 UC.tif	148,52	6	24,75
S-1005-0002-0003-00001 UC.tif	111,22	6	18,54

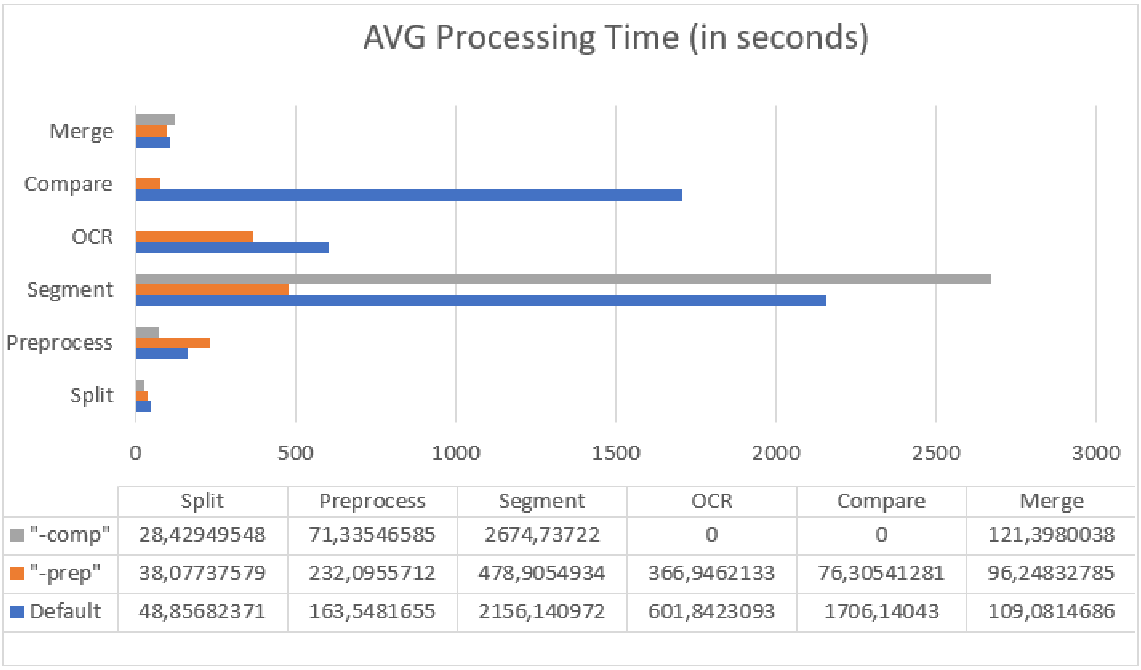
Table 5.1: Files used in the assessment

The metric to be assessed was the time each task took to finish. For that purpose, the source code of the workflow was changed, to retrieve that information. The obtained values were then saved into an Excel file.

The obtained values were collected after the workflow was used with three different parametrizations: with the default parameters, with the –prep parameter and with the –comp parameter. The use of –prep parameter was important to see if improved images resulted in an improvement of the execution time. On the other hand, the use of the parameter –comp only serves to compare the final execution time, since two steps (second OCR pass and comparison) are skipped. Figure 5.22 shows the average execution time of each task for the three different parametrizations (individual results are available in Appendix C). The total times for each parameter can be checked in Table5.2.

By looking at the results obtained it is possible to reach the following conclusions:

- Overall, the workflow usually performs faster if the –prep parameter is used. It performs the slowest using the default parameters.
- The only task where the –prep parameter takes longer than the others, is the Pre-processing.
- The segmentation task takes much more time if no additional pre-processing is provided.



	Split	Preprocess	Segment	OCR	Compare	Merge
▨ "-comp"	28,42949548	71,33546585	2674,73722	0	0	121,3980038
🟧 "-prep"	38,07737579	232,0955712	478,9054934	366,9462133	76,30541281	96,24832785
🟦 Default	48,85682371	163,5481655	2156,140972	601,8423093	1706,14043	109,0814686

Figure 5.22: Average processing time using three different parametrizations

	Total time (in seconds)
--comp	2895,9
--prep	1288,578
Default	4785,610

Table 5.2: Total processing time using three different parametrizations

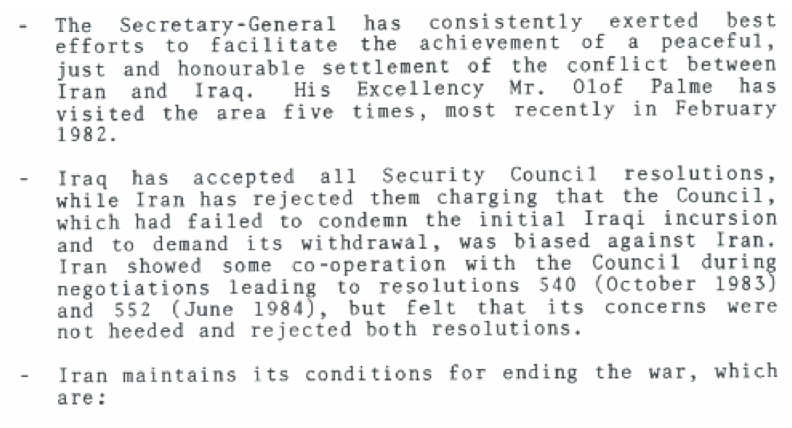
Effectiveness

The accuracy of the OCR was the chosen metric to measure the effectiveness of the workflow. However, since the majority of the documents at ARMS do not present well-structured text, it is difficult to calculate specific metrics like the Character Error Rate (CER) and the Word Error Rate (WER) using the original images. In addition to that, measuring the OCR accuracy implicates the manual transcription of the images to obtain the ground truth (i.e., the expected result), which is a laborious and time-consuming task.

To perform this assessment, three images were cropped so that the OCR system processed only well-structured text. Next, those images were processed by the workflow developed in this dissertation and by the Adobe Acrobat Pro DC software. After that, the results of each run were compared with the expected result after manually transcribing the images.

In addition to the visual comparison, the comparison method was the calculation of the Character Error

Rate and the Word Error Rate. These metrics use the Levenshtein distance [19] to calculate differences between two different strings. The CER calculates the differences on a character level, while the WER calculates it on a word level (which is more relevant in this project's scope). The closer these values are to zero, the fewer errors there were when recognizing the text. The calculation of these values was done through a Python library called fastwer.
The chosen images were selected accordingly to their legibility. The first image (Figure 5.23) contained black text on white background, providing good legibility. The second image (Figure 5.24) was less legible since it had black text on beige background. Finally, the third image (Figure 5.25) was even less legible due to its light blueish text on beige background.

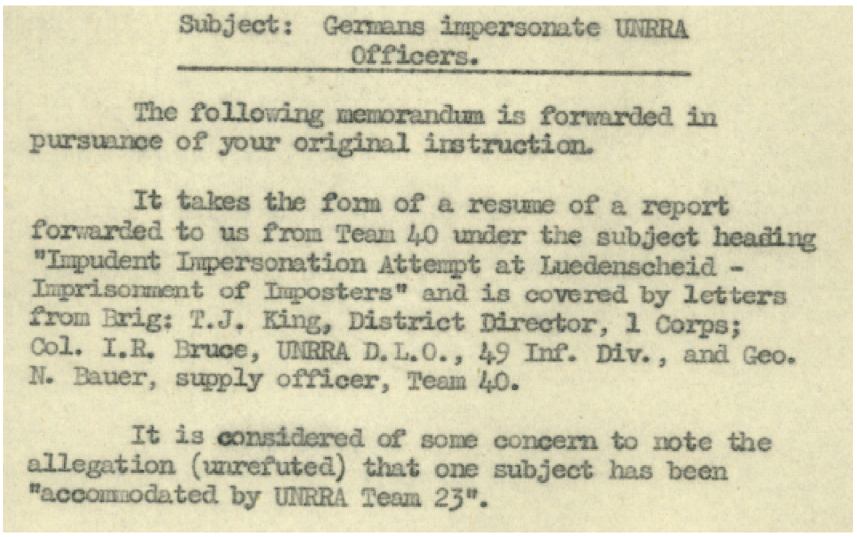


- The Secretary-General has consistently exerted best efforts to facilitate the achievement of a peaceful, just and honourable settlement of the conflict between Iran and Iraq. His Excellency Mr. Olof Palme has visited the area five times, most recently in February 1982.

- Iraq has accepted all Security Council resolutions, while Iran has rejected them charging that the Council, which had failed to condemn the initial Iraqi incursion and to demand its withdrawal, was biased against Iran. Iran showed some co-operation with the Council during negotiations leading to resolutions 540 (October 1983) and 552 (June 1984), but felt that its concerns were not heeded and rejected both resolutions.

- Iran maintains its conditions for ending the war, which are:

Figure 5.23: Image 1 used in the assessment



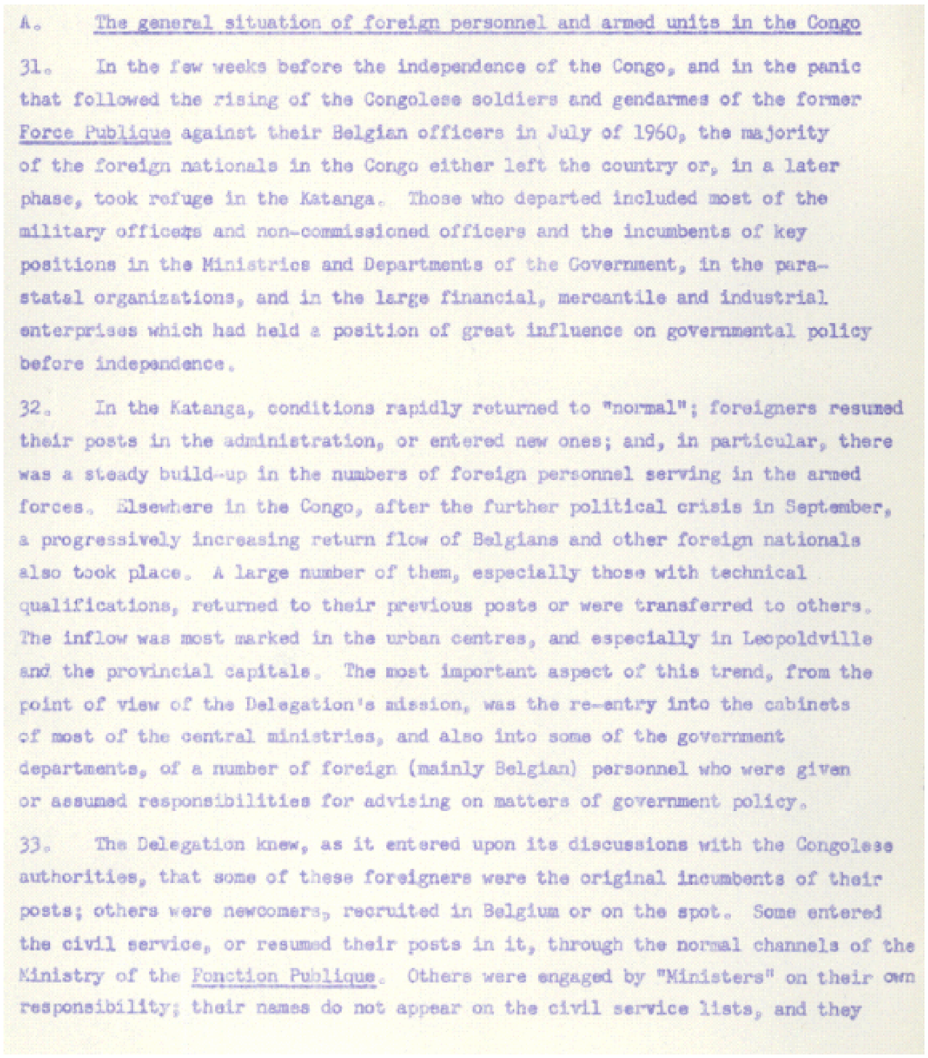
Subject: Germans impersonate UNRRA Officers.

The following memorandum is forwarded in pursuance of your original instruction.

It takes the form of a resume of a report forwarded to us from Team 40 under the subject heading "Impudent Impersonation Attempt at Luedenscheid - Imprisonment of Imposters" and is covered by letters from Brig: T.J. King, District Director, 1 Corps; Col. I.R. Bruce, UNRRA D.L.O., 49 Inf. Div., and Geo. N. Bauer, supply officer, Team 40.

It is considered of some concern to note the allegation (unrefuted) that one subject has been "accommodated by UNRRA Team 23".

Figure 5.24: Image 2 used in the assessment



A. <u>The general situation of foreign personnel and armed units in the Congo</u>

31. In the few weeks before the independence of the Congo, and in the panic that followed the rising of the Congolese soldiers and gendarmes of the former <u>Force Publique</u> against their Belgian officers in July of 1960, the majority of the foreign nationals in the Congo either left the country or, in a later phase, took refuge in the Katanga. Those who departed included most of the military officers and non-commissioned officers and the incumbents of key positions in the Ministries and Departments of the Government, in the para-statal organizations, and in the large financial, mercantile and industrial enterprises which had held a position of great influence on governmental policy before independence.

32. In the Katanga, conditions rapidly returned to "normal"; foreigners resumed their posts in the administration, or entered new ones; and, in particular, there was a steady build-up in the numbers of foreign personnel serving in the armed forces. Elsewhere in the Congo, after the further political crisis in September, a progressively increasing return flow of Belgians and other foreign nationals also took place. A large number of them, especially those with technical qualifications, returned to their previous posts or were transferred to others. The inflow was most marked in the urban centres, and especially in Leopoldville and the provincial capitals. The most important aspect of this trend, from the point of view of the Delegation's mission, was the re-entry into the cabinets of most of the central ministries, and also into some of the government departments, of a number of foreign (mainly Belgian) personnel who were given or assumed responsibilities for advising on matters of government policy.

33. The Delegation knew, as it entered upon its discussions with the Congolese authorities, that some of these foreigners were the original incumbents of their posts; others were newcomers, recruited in Belgium or on the spot. Some entered the civil service, or resumed their posts in it, through the normal channels of the Ministry of the <u>Fonction Publique</u>. Others were engaged by "Ministers" on their own responsibility; their names do not appear on the civil service lists, and they

Figure 5.25: Image 3 used in the assessment

To compare the results, the workflow was used with two different parameters: the default parameters and the –prep parameter. This way, it was possible to infer how helpful the latter could be.
The obtained results when using the workflow with the default parameters, the –prep parameter and when using the Adobe Acrobat Pro DC software, can be seen in Table 5.3.
When analyzing the results, it is possible to conclude that in the set of images used, the proposed solution of this dissertation provided better results than the Adobe Acrobat Pro DC software, when recognizing the text contained in the images. Regarding the use of the –prep parameter, the results have shown that it can be useful in a certain type of images. However, it does not always provide better results than the default parameter, that is why it is necessary to keep both options available.
All the text recognized by the different options used, can be seen in Appendix D.

	Default		**--prep**		**Adobe**	
	CER (%)	WER (%)	CER (%)	WER (%)	CER (%)	WER (%)
Image 1	1,05	3,36	1,05	3,36	6,19	65,55
Image 2	68,72	76,04	6,02	21,88	No text	No text
Image 3	16,17	27,42	34,66	77,56	61,98	97,3

Table 5.3: CER (%) and WER (%) of the three experiments

Chapter 6

Conclusions

This chapter aims to present what conclusions have been reached after completing this dissertation while outlining what can be done in the future to improve it.
It is divided into three different sections. The first section presents the main challenges faced during the course of this project. After that, a discussion of the results obtained is provided. Finally, the last section goes through the future work that can be done in order to improve the solution proposed in this thesis.

6.1 Main challenges and lessons learned

Although there are already solutions available in the market capable of performing optical character recognition, the goal of this dissertation was broader than that. This was, in fact, one of the components of the workflow, but it also had to be able to fulfill all the ARMS requirements, performing other tasks in the most automated way possible. Although this sounds like an easy task, it turned out to be quite a challenge.
To begin with, I had no knowledge about this topic, so I had to deepen my knowledge of these matters, namely about the PDF and TIFF formats and the existing text recognition technologies.
The second challenge was to understand what types of documents were processed by ARMS. After being provided with some examples of TIFF images, there was an attempt to characterize the type of documents used, creating a type-collection. However, this work ended up being left behind since the type of files processed is so inhomogeneous that it would be almost impossible to create a workflow that would respond to the specific needs of each type of document. Many times changes were made to the design because the results were better that way, only to test other types of images and realize that this was not the ideal solution after all. So, the solution was to design a more general workflow with some parameterization to adjust to existing needs.
The third challenge was to understand how the text recognition process works. Understanding how images should be prepared so that the process occurs in the best possible way, how decisions are made within the system, or how the collected information is processed are fundamental steps in developing a solution of this type. However, the Tesseract software, namely the Python wrapper of its API, tesserocr,

is relatively well documented, which allows exploring all the functionalities of this technology.

The fourth major challenge was to find a way to deploy the workflow since there was an incompatibility between the development environment and the environment in which the workflow would be used. It was then found that the best fitting technology was the Docker software. For that purpose, it was necessary to explore this technology to set up a container to deploy and install the solution in ARMS's computers.

Finally, one of the lessons learned from this dissertation was focusing on the planned goal. Although there is always the need and desire to add new features, it is better to have a workflow that works from start to finish, but that is simpler than a workflow with several extra features, but that cannot deliver what it promised. With limited time to develop this dissertation and while trying to balance the relationship between complexity and efficiency of the final solution, it was a challenge to achieve this final result.

6.2 Discussion of the results

As mentioned earlier, it was necessary to develop a new workflow for processing documents scanned for ARMS since the current workflow lacked automation, and sometimes satisfactory results. It was with this in mind that this dissertation was composed.

After presenting the problem, it was necessary to start outlining what would become the final solution. For this, it was essential to deepen the knowledge about the relevant file formats, namely PDF and TIFF. Furthermore, after this study, it was also essential to know the state of the art of OCR technology since this is one of the fundamental parts of the workflow. After gathering all this knowledge, an analysis of the existing solution and the stakeholder requirements was made to design a solution.

In an initial phase, this development resulted in a simple workflow with the goal of serving as a proof-of-concept. The goal of this design choice was to have a complete (i.e., workflow that functioned from start to finish) to present to the ARMS team.

Since this workflow was too simplistic and had the potential to add other functionalities, the initial workflow started to be incremented until it reached a final version. In this final version, new functionalities such as pre-processing the images or using two different segmentation types were added.

Upon completion of this final version, it was deployed to the computers of the ARMS team in order to test the feasibility of using this workflow. At the same time, performance assessments were made, to measure the efficiency and effectiveness of the workflow. The results of these evaluations were satisfactory, although there is room for improvement (the "Future Work" section of this chapter presents some solutions for improving the existing solution).

In conclusion, this dissertation was able to deliver what it set out to do, namely a complete workflow capable of processing a set of digitized images, resulting in a PDF file with a text layer, ready to be uploaded to the ARMS public database.

6.3 Future Work

It is important to emphasize that although this project was developed in partnership with ARMS to meet their needs, nothing prevents this project from being used in other contexts requiring a workflow of this type. Furthermore, the fact that it was built in a modular way allows new functionalities to be added and changing existing ones.
However, even though this is a functional workflow, improvements can be made to ensure that this can be the de facto workflow used by the ARMS staff. In order to improve the proposed solution, there was a collaboration with another dissertation, which is already adding ways to improve the use of this workflow, namely a more appealing interface for the user. One of the functionalities available in this interface will be the possibility of manually correcting the results obtained by the OCR.
Another topic that could be explored in the future is improving the pre-processing techniques used since this is a fundamental part of the workflow. Creating a trained model capable of characterizing the existing document types and applying the best techniques automatically would be an interesting idea to explore. On the topic of trained models, it would also be good to create specific dictionaries with frequently used terms in ARMS documents so that there are fewer errors in recognizing textual content.
Through the evaluations made of the existing solution, it was also apparent that the following features need to be improved:

- Improve the size of the final PDF files using compression techniques.
- Explore multithreading so that the processing can be done faster.
- Create better heuristics to compare the decisions made since the highest word confidence does not always correspond to the correct result.

Bibliography

[1] United nations charter and statute of the international court of justice, 1945. URL https://treaties.un.org/doc/publication/ctc/uncharter.pdf.

[2] R. Claus. The united nations archives. *The American Archivist*, 10(2):129–132, 1947. ISSN 03609081. URL http://www.jstor.org/stable/40288557.

[3] T. Newton. Record-keeping requirements for digitization, Apr 2009. URL https://archives.un.org/sites/archives.un.org/files/RM_PDFs/record-keeping_requirements_for_digitization.pdf.

[4] R. Wiggins, H. Davidson, H. Harnsberger, J. Lauman, and P. Goede. Image file formats: Past, present, and future1. *Radiographics : a review publication of the Radiological Society of North America, Inc*, 21:789–98, 11 2000. doi: 10.1148/radiographics.21.3.g01ma25789.

[5] *TIFF - Revision 6.0*. Adobe Developers Association, https://www.adobe.io/content/dam/udp/en/open/standards/tiff/TIFF6.pdf, June 1992. Retrieved: 20-07-2021.

[6] ISO. Document management—portable document format—part1:pdf1.7. ISO 32000-2012 1:2008, International Organization for Standardization, Geneva, Switzerland, 2008.

[7] E. E. Fournier D'Albe. On a type-reading optophone. *Proceedings of the Royal Society of London. Series A, Containing Papers of a Mathematical and Physical Character*, 90(619):373–375, 1914. doi: 10.1098/rspa.1914.0061.

[8] L. Eikvil. Ocr - optical character recognition, 1993.

[9] N. Anderson, G. Muhlberger, and A. Antonacopoulos. Optical character recognition impact best practice guide impact project. 2013.

[10] A. Somani. Council post: The future of ocr is deep learning, Sep 2019. URL https://www.forbes.com/sites/forbestechcouncil/2019/09/10/the-future-of-ocr-is-deep-learning/.

[11] R. Smith. History of the tesseract ocr engine: what worked and what didn't. *Proceedings of SPIE - The International Society for Optical Engineering*, pages 02–, 02 2013. doi: 10.1117/12.2010051.

[12] N. Islam, Z. Islam, and N. Noor. A survey on optical character recognition system. *ITB Journal of Information and Communication Technology*, 12 2016.

[13] Tesseract User Manual. URL `https://tesseract-ocr.github.io/tessdoc/`.

[14] R. Smith. An overview of the tesseract ocr engine. volume 2, pages 629 – 633, 10 2007. ISBN 978-0-7695-2822-9. doi: 10.1109/ICDAR.2007.4376991.

[15] M. Brisinello, R. Grbić, M. Pul, and T. Andelić. Improving optical character recognition performance for low quality images. In *2017 International Symposium ELMAR*, pages 167–171, 2017.

[16] T. Breuel. The ocropus open source ocr system. volume 6815, page 68150, 01 2008. doi: 10.1117/12.783598.

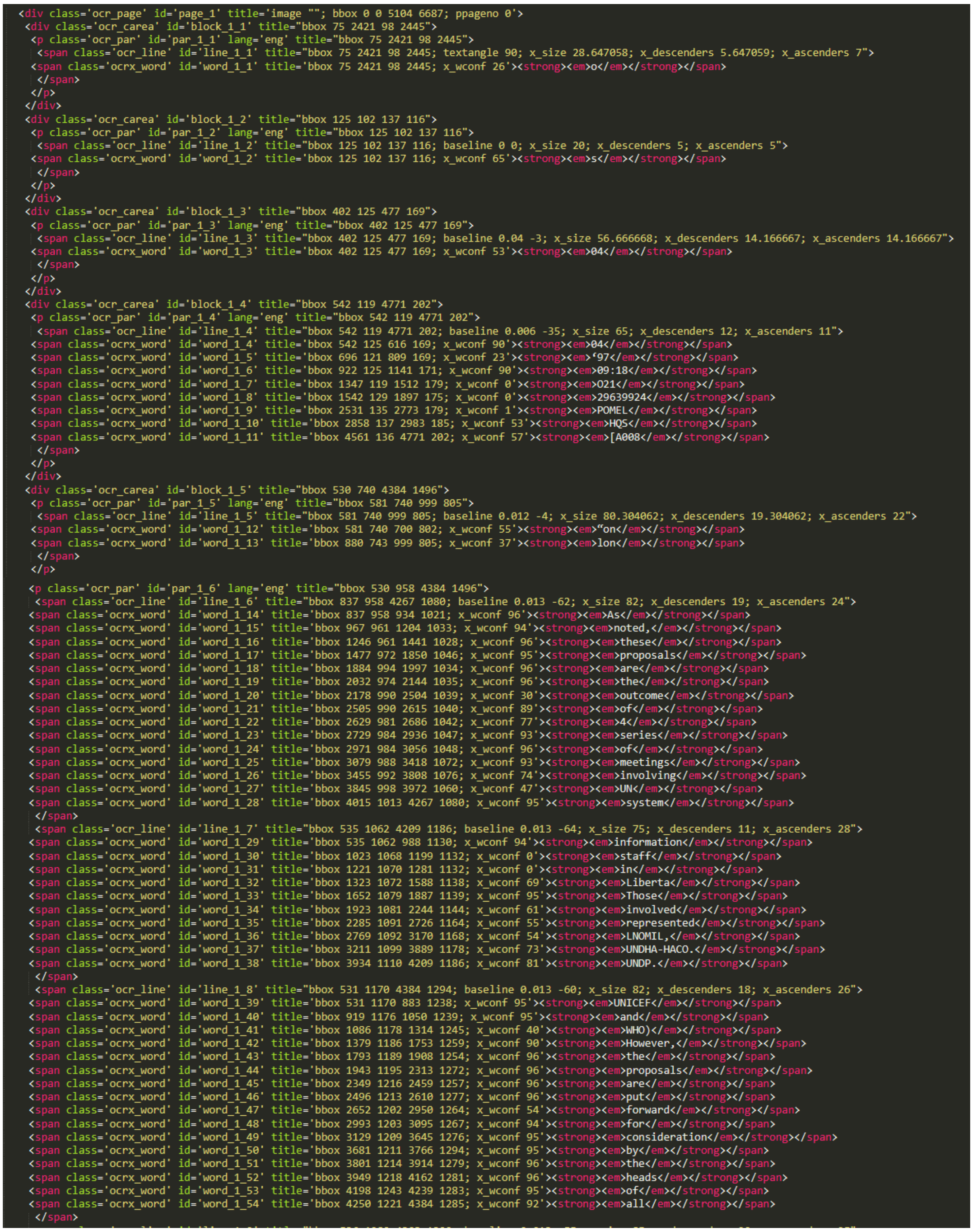
[17] T. Breuel and U. Kaiserslautern. The hocr microformat for ocr workflow and results. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, volume 2, pages 1063–1067, 2007. doi: 10.1109/ICDAR.2007.4377078.

[18] hOCR - OCR Workflow and Output embedded in HTML, 02 2020. URL `http://kba.cloud/hocr-spec/1.2/#baseline`.

[19] A. S. Lhoussain, G. Hicham, and Y. Abdellah. Adaptating the levenshtein distance to contextual spelling correction. *International Journal of Computer Science and Applications*, 12(1):127–133, 2015.

Appendix A

Example of hOCR file



```html
<div class='ocr_page' id='page_1' title='image ""; bbox 0 0 5104 6687; ppageno 0'>
 <div class='ocr_carea' id='block_1_1' title="bbox 75 2421 98 2445">
  <p class='ocr_par' id='par_1_1' lang='eng' title="bbox 75 2421 98 2445">
   <span class='ocr_line' id='line_1_1' title="bbox 75 2421 98 2445; textangle 90; x_size 28.647058; x_descenders 5.647059; x_ascenders 7">
  <span class='ocrx_word' id='word_1_1' title='bbox 75 2421 98 2445; x_wconf 26'><strong><em>o</em></strong></span>
   </span>
  </p>
 </div>
 <div class='ocr_carea' id='block_1_2' title="bbox 125 102 137 116">
  <p class='ocr_par' id='par_1_2' lang='eng' title="bbox 125 102 137 116">
   <span class='ocr_line' id='line_1_2' title="bbox 125 102 137 116; baseline 0 0; x_size 20; x_descenders 5; x_ascenders 5">
  <span class='ocrx_word' id='word_1_2' title='bbox 125 102 137 116; x_wconf 65'><strong><em>s</em></strong></span>
   </span>
  </p>
 </div>
 <div class='ocr_carea' id='block_1_3' title="bbox 402 125 477 169">
  <p class='ocr_par' id='par_1_3' lang='eng' title="bbox 402 125 477 169">
   <span class='ocr_line' id='line_1_3' title="bbox 402 125 477 169; baseline 0.04 -3; x_size 56.666668; x_descenders 14.166667; x_ascenders 14.166667">
  <span class='ocrx_word' id='word_1_3' title='bbox 402 125 477 169; x_wconf 53'><strong><em>04</em></strong></span>
   </span>
  </p>
 </div>
 <div class='ocr_carea' id='block_1_4' title="bbox 542 119 4771 202">
  <p class='ocr_par' id='par_1_4' lang='eng' title="bbox 542 119 4771 202">
   <span class='ocr_line' id='line_1_4' title="bbox 542 119 4771 202; baseline 0.006 -35; x_size 65; x_descenders 12; x_ascenders 11">
  <span class='ocrx_word' id='word_1_4' title='bbox 542 125 616 169; x_wconf 90'><strong><em>04</em></strong></span>
  <span class='ocrx_word' id='word_1_5' title='bbox 696 121 809 169; x_wconf 23'><strong><em>‘97</em></strong></span>
  <span class='ocrx_word' id='word_1_6' title='bbox 922 125 1141 171; x_wconf 90'><strong><em>09:18</em></strong></span>
  <span class='ocrx_word' id='word_1_7' title='bbox 1347 119 1512 179; x_wconf 0'><strong><em>O21</em></strong></span>
  <span class='ocrx_word' id='word_1_8' title='bbox 1542 129 1897 175; x_wconf 0'><strong><em>29639924</em></strong></span>
  <span class='ocrx_word' id='word_1_9' title='bbox 2531 135 2773 179; x_wconf 1'><strong><em>POMEL</em></strong></span>
  <span class='ocrx_word' id='word_1_10' title='bbox 2858 137 2983 185; x_wconf 53'><strong><em>HQS</em></strong></span>
  <span class='ocrx_word' id='word_1_11' title='bbox 4561 136 4771 202; x_wconf 57'><strong><em>[A008</em></strong></span>
   </span>
  </p>
 </div>
 <div class='ocr_carea' id='block_1_5' title="bbox 530 740 4384 1496">
  <p class='ocr_par' id='par_1_5' lang='eng' title="bbox 581 740 999 805">
   <span class='ocr_line' id='line_1_5' title="bbox 581 740 999 805; baseline 0.012 -4; x_size 80.304062; x_descenders 19.304062; x_ascenders 22">
  <span class='ocrx_word' id='word_1_12' title='bbox 581 740 700 802; x_wconf 55'><strong><em>“on</em></strong></span>
  <span class='ocrx_word' id='word_1_13' title='bbox 880 743 999 805; x_wconf 37'><strong><em>lon</em></strong></span>
   </span>
  </p>

  <p class='ocr_par' id='par_1_6' lang='eng' title="bbox 530 958 4384 1496">
   <span class='ocr_line' id='line_1_6' title="bbox 837 958 4267 1080; baseline 0.013 -62; x_size 82; x_descenders 19; x_ascenders 24">
  <span class='ocrx_word' id='word_1_14' title='bbox 837 958 934 1021; x_wconf 96'><strong><em>As</em></strong></span>
  <span class='ocrx_word' id='word_1_15' title='bbox 967 961 1204 1033; x_wconf 94'><strong><em>noted,</em></strong></span>
  <span class='ocrx_word' id='word_1_16' title='bbox 1246 961 1441 1028; x_wconf 96'><strong><em>these</em></strong></span>
  <span class='ocrx_word' id='word_1_17' title='bbox 1477 972 1850 1046; x_wconf 95'><strong><em>proposals</em></strong></span>
  <span class='ocrx_word' id='word_1_18' title='bbox 1884 994 1997 1034; x_wconf 96'><strong><em>are</em></strong></span>
  <span class='ocrx_word' id='word_1_19' title='bbox 2032 974 2144 1035; x_wconf 96'><strong><em>the</em></strong></span>
  <span class='ocrx_word' id='word_1_20' title='bbox 2178 990 2504 1039; x_wconf 30'><strong><em>outcome</em></strong></span>
  <span class='ocrx_word' id='word_1_21' title='bbox 2505 990 2615 1040; x_wconf 89'><strong><em>of</em></strong></span>
  <span class='ocrx_word' id='word_1_22' title='bbox 2629 981 2686 1042; x_wconf 77'><strong><em>4</em></strong></span>
  <span class='ocrx_word' id='word_1_23' title='bbox 2729 984 2936 1047; x_wconf 93'><strong><em>series</em></strong></span>
  <span class='ocrx_word' id='word_1_24' title='bbox 2971 984 3056 1048; x_wconf 96'><strong><em>of</em></strong></span>
  <span class='ocrx_word' id='word_1_25' title='bbox 3079 988 3418 1072; x_wconf 93'><strong><em>meetings</em></strong></span>
  <span class='ocrx_word' id='word_1_26' title='bbox 3455 992 3808 1076; x_wconf 74'><strong><em>involving</em></strong></span>
  <span class='ocrx_word' id='word_1_27' title='bbox 3845 998 3972 1060; x_wconf 47'><strong><em>UN</em></strong></span>
  <span class='ocrx_word' id='word_1_28' title='bbox 4015 1013 4267 1080; x_wconf 95'><strong><em>system</em></strong></span>
   </span>
   <span class='ocr_line' id='line_1_7' title="bbox 535 1062 4209 1186; baseline 0.013 -64; x_size 75; x_descenders 11; x_ascenders 28">
  <span class='ocrx_word' id='word_1_29' title='bbox 535 1062 988 1130; x_wconf 94'><strong><em>information</em></strong></span>
  <span class='ocrx_word' id='word_1_30' title='bbox 1023 1068 1199 1132; x_wconf 0'><strong><em>staff</em></strong></span>
  <span class='ocrx_word' id='word_1_31' title='bbox 1221 1070 1281 1132; x_wconf 0'><strong><em>in</em></strong></span>
  <span class='ocrx_word' id='word_1_32' title='bbox 1323 1072 1588 1138; x_wconf 69'><strong><em>Liberta</em></strong></span>
  <span class='ocrx_word' id='word_1_33' title='bbox 1652 1079 1887 1139; x_wconf 95'><strong><em>Those</em></strong></span>
  <span class='ocrx_word' id='word_1_34' title='bbox 1923 1081 2244 1144; x_wconf 61'><strong><em>involved</em></strong></span>
  <span class='ocrx_word' id='word_1_35' title='bbox 2285 1091 2726 1164; x_wconf 55'><strong><em>represented</em></strong></span>
  <span class='ocrx_word' id='word_1_36' title='bbox 2769 1092 3170 1168; x_wconf 54'><strong><em>LNOMIL,</em></strong></span>
  <span class='ocrx_word' id='word_1_37' title='bbox 3211 1099 3889 1178; x_wconf 73'><strong><em>UNDHA-HACO.</em></strong></span>
  <span class='ocrx_word' id='word_1_38' title='bbox 3934 1110 4209 1186; x_wconf 81'><strong><em>UNDP.</em></strong></span>
   </span>
   <span class='ocr_line' id='line_1_8' title="bbox 531 1170 4384 1294; baseline 0.013 -60; x_size 82; x_descenders 18; x_ascenders 26">
  <span class='ocrx_word' id='word_1_39' title='bbox 531 1170 883 1238; x_wconf 95'><strong><em>UNICEF</em></strong></span>
  <span class='ocrx_word' id='word_1_40' title='bbox 919 1176 1050 1239; x_wconf 95'><strong><em>and</em></strong></span>
  <span class='ocrx_word' id='word_1_41' title='bbox 1086 1178 1314 1245; x_wconf 40'><strong><em>WHO)</em></strong></span>
  <span class='ocrx_word' id='word_1_42' title='bbox 1379 1186 1753 1259; x_wconf 90'><strong><em>However,</em></strong></span>
  <span class='ocrx_word' id='word_1_43' title='bbox 1793 1189 1908 1254; x_wconf 96'><strong><em>the</em></strong></span>
  <span class='ocrx_word' id='word_1_44' title='bbox 1943 1195 2313 1272; x_wconf 96'><strong><em>proposals</em></strong></span>
  <span class='ocrx_word' id='word_1_45' title='bbox 2349 1216 2459 1257; x_wconf 96'><strong><em>are</em></strong></span>
  <span class='ocrx_word' id='word_1_46' title='bbox 2496 1213 2610 1277; x_wconf 96'><strong><em>put</em></strong></span>
  <span class='ocrx_word' id='word_1_47' title='bbox 2652 1202 2950 1264; x_wconf 54'><strong><em>forward</em></strong></span>
  <span class='ocrx_word' id='word_1_48' title='bbox 2993 1203 3095 1267; x_wconf 94'><strong><em>for</em></strong></span>
  <span class='ocrx_word' id='word_1_49' title='bbox 3129 1209 3645 1276; x_wconf 95'><strong><em>consideration</em></strong></span>
  <span class='ocrx_word' id='word_1_50' title='bbox 3681 1211 3766 1294; x_wconf 95'><strong><em>by</em></strong></span>
  <span class='ocrx_word' id='word_1_51' title='bbox 3801 1214 3914 1279; x_wconf 96'><strong><em>the</em></strong></span>
  <span class='ocrx_word' id='word_1_52' title='bbox 3949 1218 4162 1281; x_wconf 96'><strong><em>heads</em></strong></span>
  <span class='ocrx_word' id='word_1_53' title='bbox 4198 1243 4239 1283; x_wconf 95'><strong><em>of</em></strong></span>
  <span class='ocrx_word' id='word_1_54' title='bbox 4250 1221 4384 1285; x_wconf 92'><strong><em>all</em></strong></span>
   </span>
```

Appendix B

Available parameters

- –comp - Disables the line segmentation comparison, as explained in the “Segmentation” subsection. Usage:

  ```
  $ python3 workflow.py path_to_file –comp
  ```

- –folder - Instead of running the workflow in just a single file, all the files in the specified folder are processed. Usage:

  ```
  $ python3 workflow.py –folder path_to_folder
  ```

- –force - If the workflow was interrupted, instead of continuing from where it stopped, it starts from the beginning. Usage:

  ```
  $ python3 workflow.py path_to_file –force
  ```

- –help - Shows how to run the workflow and shows the existing parameters.Usage:

  ```
  $ python3 workflow.py –help
  ```

- –lang - This parameter is used to tell the workflow in which language the documents are written. More than one language can be selected. Default is English. This parameter is important for the OCR processing, and its use is recommended. Available languages have to be downloaded previously. Usage (example for portuguese and english):

  ```
  $ python3 workflow.py path_to_file –lang por –lang eng
  ```

- –prep - Performs some additional pre-processing techniques to the images, as explained in the “Pre-processing” subsection.

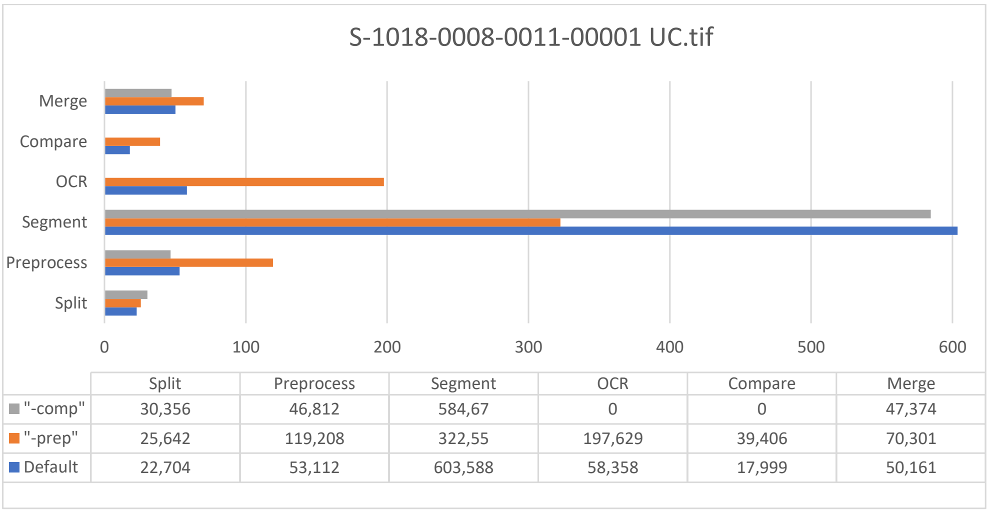
  ```
  $python3 workflow.py path_to_file –prep
  ```

- –tmp - Keeps the temporary files instead of deleting them at the end of the workflow

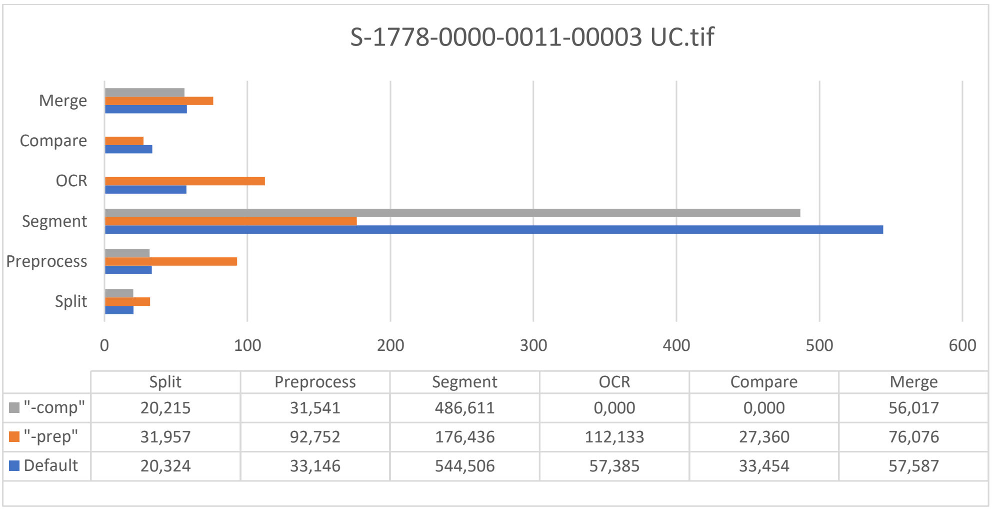
  ```
  $python3 workflow.py path_to_file –temp
  ```

Appendix C

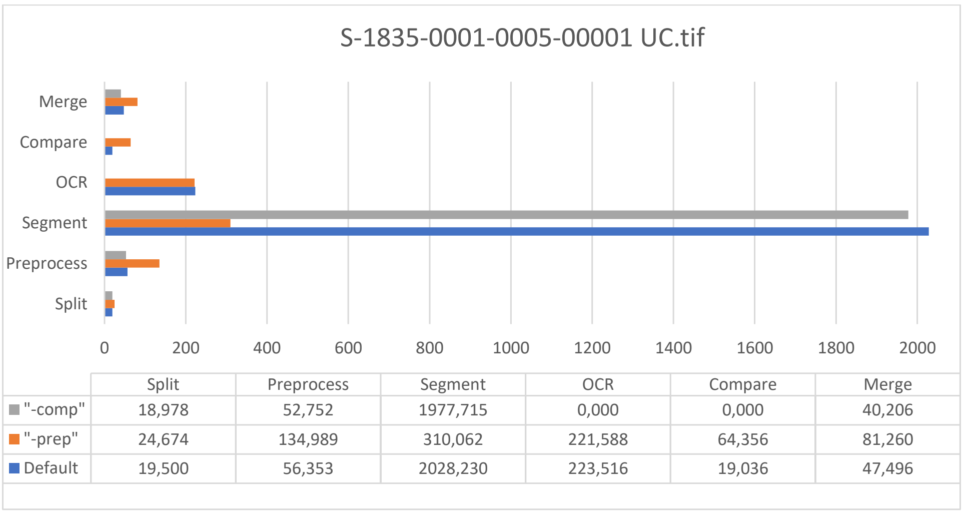
Individual results of efficiency assessment



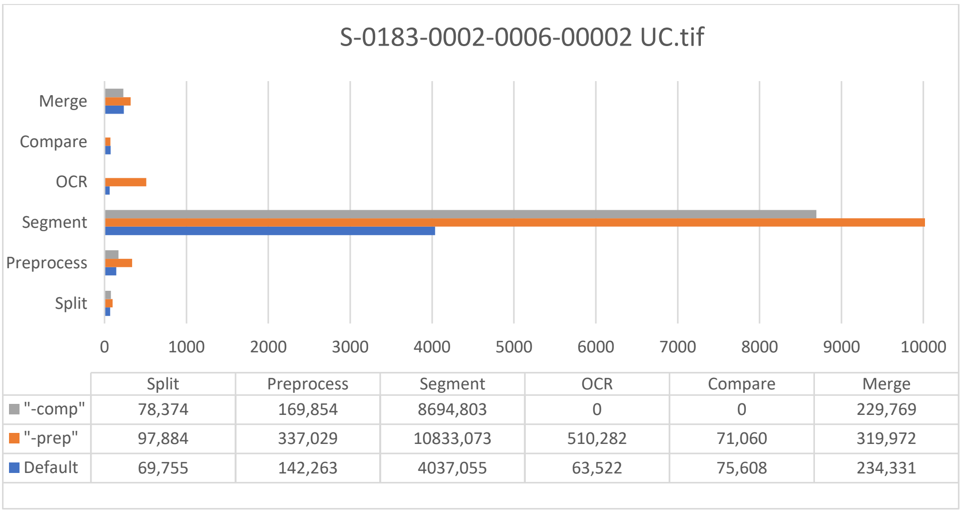
	Split	Preprocess	Segment	OCR	Compare	Merge
■ "-comp"	30,356	46,812	584,67	0	0	47,374
🟧 "-prep"	25,642	119,208	322,55	197,629	39,406	70,301
🟦 Default	22,704	53,112	603,588	58,358	17,999	50,161



	Split	Preprocess	Segment	OCR	Compare	Merge
■ "-comp"	20,215	31,541	486,611	0,000	0,000	56,017
🟧 "-prep"	31,957	92,752	176,436	112,133	27,360	76,076
🟦 Default	20,324	33,146	544,506	57,385	33,454	57,587

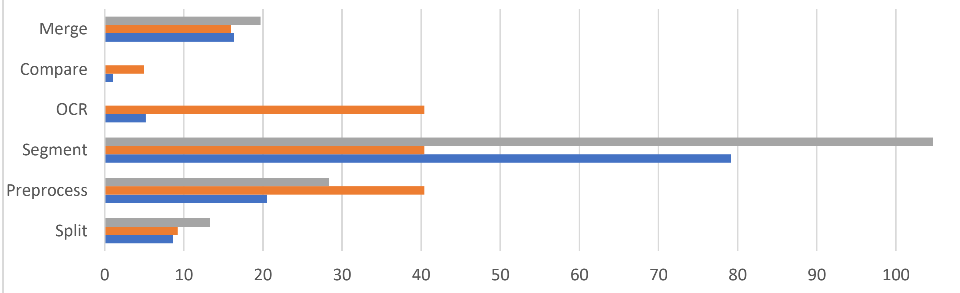


	Split	Preprocess	Segment	OCR	Compare	Merge
⬜ "-comp"	18,978	52,752	1977,715	0,000	0,000	40,206
🟧 "-prep"	24,674	134,989	310,062	221,588	64,356	81,260
🟦 Default	19,500	56,353	2028,230	223,516	19,036	47,496



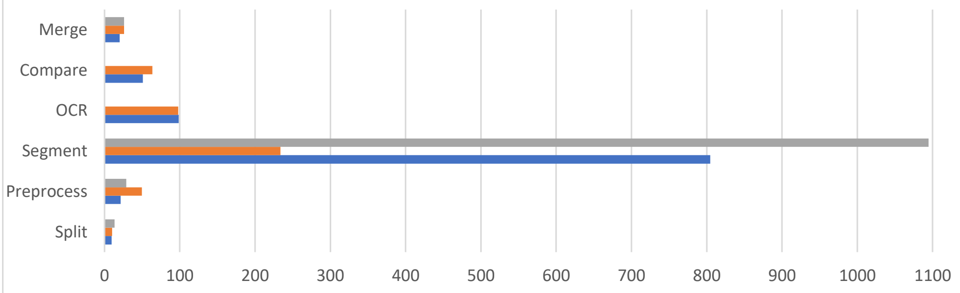
	Split	Preprocess	Segment	OCR	Compare	Merge
⬜ "-comp"	78,374	169,854	8694,803	0	0	229,769
🟧 "-prep"	97,884	337,029	10833,073	510,282	71,060	319,972
🟦 Default	69,755	142,263	4037,055	63,522	75,608	234,331

S-1005-0002-0003-00001 UC.tif



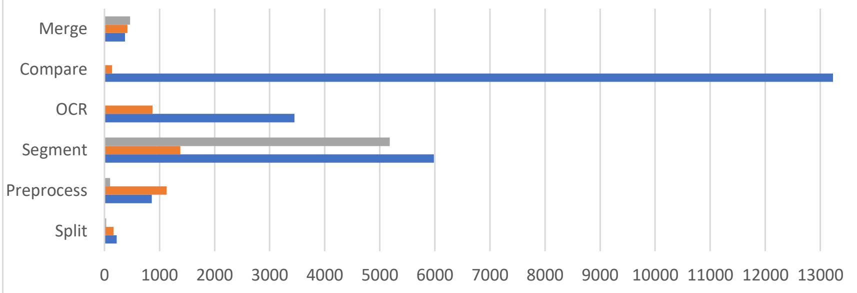
	Split	Preprocess	Segment	OCR	Compare	Merge
■ "-comp"	13,311	28,356	104,709	0,000	0,000	19,688
🟧 "-prep"	9,231	40,398	40,398	40,398	4,947	15,941
🟦 Default	8,638	20,510	79,159	5,179	1,029	16,332

S-1006-0004-0001-00001 UC.tif



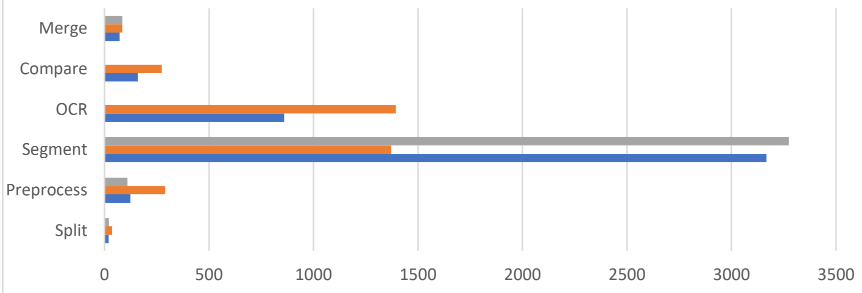
	Split	Preprocess	Segment	OCR	Compare	Merge
■ "-comp"	13,372	28,859	1094,670	0,000	0,000	25,841
🟧 "-prep"	10,087	49,694	233,536	97,771	63,531	25,990
🟦 Default	9,309	21,278	804,570	98,427	50,980	20,210

S-0357-0021-0003-00001 UC.tif



	Split	Preprocess	Segment	OCR	Compare	Merge
⬜ "-comp"	31,386	102,874	5180,257	0,000	0,000	467,507
🟧 "-prep"	166,366	1129,247	1376,567	871,564	136,998	415,132
🟦 Default	220,478	857,662	5984,214	3448,775	13290,847	373,786

S-1835-0024-0008-00001 UC.tif



	Split	Preprocess	Segment	OCR	Compare	Merge
⬜ "-comp"	21,443	109,636	3274,463	0,000	0,000	84,782
🟧 "-prep"	36,662	290,477	1371,695	1394,487	273,846	85,287
🟦 Default	20,145	124,061	3167,806	859,577	160,171	72,748

Appendix D

Individual results of effectiveness assessment

Image 1

Expected

- The Secretary-General has consistently exerted best efforts to facilitate the achievement of a peaceful, just and honourable settlement of the conflict between Iran and Iraq. His Excellency Mr. Olof Palme has visited the area five times, most recently in February 1982. - Iraq has accepted all Security Council resolutions, while Iran has rejected them charging that the Council, which had failed to condemn the initial Iraqi incursion and to demand its withdrawal, was biased against Iran. Iran showed some co-operation with the Council during negotiations leading to resolutions 540 (October 1983) and 552 (June 1984), but felt that its concerns were not heeded and rejected both resolutions. - Iran maintains its conditions for ending the war, which are:

Adobe

- The Secretary-Ge neral ha s con sis tently exer ted be st effort s to f aci li tate the achi evement of a peaceful, just and honourab le settlement of the conflict between Ir an and Iraq. His Excellency Mr. Olof Pa lme ha s visited the area fi ve times , most recen tly in Februar y 1982. - Iraq has accepted al l Secur ity Council r esol utions, while Iran has rejec ted them chargin g that the Cou ncil , which had failed to conde mn the initial Ir□qi incursion and to demand its withdrawal, was bia se d against I ran. Iran showed some co-operatio n with the Council during negotiation s lea di ng to re solut ions 540 (Oct ober 1983) and 552 (June 1984), but felt that it s concer ns were not heeded and rejected both res olu tions. - Iran maintains its condition s for ending the war, which are:

--prep

The Secretary-General has consistently exerted best efforts to facilitate the achievement of peaceful, just and honourable settlement of the conflict between Iran and Iraq. His Excellency Mr. Olof Palme has visited the area five times, most recently in February 1982. Iraq has accepted all Security Council resolutions, while Iran has rejected them charging that the Council, which had failed to condemn the initial Iraqi incursion and to demand its withdrawal, was biased against Iran. Iran showed some co-operation with the Council during negotiations leading to resolutions 540 (October 1983) and 552 (June 1984), but felt that its concerns were not heeded and rejected both resolutions. Iran maintains its conditions for ending the war, which are:

Default

The Secretary-General has consistently exerted best efforts to facilitate the achievement of peaceful, just and honourable settlement of the conflict between Iran and Iraq. His Excellency Mr. Olof Palme has visited the area five times, most recently in February 1982. Iraq has accepted all Security Council resolutions, while Iran has rejected them charging that the Council, which had failed to condemn the initial Iraqi incursion and to demand its withdrawal, was biased against Iran. Iran showed some co-operation with the Council during negotiations leading to resolutions 540 (October 1983) and 552 (June 1984), but felt that its concerns were not heeded and rejected both resolutions. Iran maintains its conditions for ending the war, which are:

Image 2

Expected

Subject: Germans impersonate UNRRA Officers. The following memorandum is forward in pursuance of your original instruction. It takes the form of a resume of a report forwarded to us from Team 40 under the subject heading “Impudent Impersonation Attempt at Luedenscheid – Imprisonment of Imposters” and is covered by letters from Brig: T. J. King District Director, I Corps; Col. I.R. Bruce, UNRRA D.L.O., 49 Inf. Div, and Geo. N. Bauer, supply officer, Team 40. It is considered of some concern to note the allegation (unrefuted) that one subject has been “accommodated by UNRRA Team 23”.

Adobe

Not text recognized

--prep

Subject: Germans impersonate UNRRA Officers. The following memorandum is forwarded in pursuance of your origim) instruction. It takes the fom of resume of report forwarded to us from Team 40 under the subject heading “Impudent Impersonation Attempt at Luedenscheid Imprisomment of Imposters* and is covered by letters from Brig: T.J. King, District Director, Corps; Col. I.R. Bruce, UNRRA D.L.0., 49 Inf. Div., and Geo. HN. Bauer, supply officer, Team 40. It 1s eonsidered of some coneern to note the allegation (umrefuted) that one subject has been "accommodated by UNRRA Team 23°

Default

imkes the foma of resume of report —* Me ate ES i: "ector od Corps; Col. ILR, ‘Bruce, UMRA D..1..0- 49 Inf. Div., and Geo. N. Bauer, supply officer, Team 4,0. futed) that one subject has been dated by ‘UNRRA Team 23".

Image 3

Expected

A. The general situation of foreign personnel and armed units in the Congo 31. In the few weeks before the independence of the Congo, and in the panic that followed the riing of the Congolese soldiers and gendarmes of the former Force Publique against their Belgian officers in July of 1960, the majority of the foreign nationals in the Congo either left the country or, in a later phase, took refuge in the Katanga. Those who departed included most of the military officers and non-comissioned officers and the incumbents of key positions in the Ministries and Departments of the Government, in the parastatal organizations, and in the large financial, mercantile and industrial enterprises which had held a position of the great influence on governmental policy before independence.32. In the Katanga, conditions rapidly returned to "normal"; foreigners resumed their posts in the administration, or entered new ones; and, in particular, there was a steady build-up in the numbers of foreign personnel serving in the armed forces. Elsewhere in the Congo, after the further political crisis in September, a progressively increasing return flow of Belgians and other foreign nationals also took place. A large number of them, especially those with technical qualifications, returned to their previous posts or ere transferred to others. The inflow was most marked in the urban centres, and especially in Leopoldville and the provincial capitals. The most important aspect of this trend, from the point of view of the Delegation's mission, was the re-entry into the cabinets of most of the central ministries, and also into some of the government departments, of a number of foreign (mainly Belgian) personnel who were given or assumed responsibilities for advising on matters of government policy.33. The Delegation knew, as it entered upon its discussions with the Congolese authorities, that some of these foreigners were the original incumbents of their posts; others were newcomers, recruited in Belgium or on the spot. Some entered the civil service, or resumed their posts in it, through the normal channels of the Ministry of Foction Publique. Others were engaged by "Ministers" on their own responsibility; their names do not appear on the civil service lists, and they

Adobe

```
Jlr.; In. the few lfeeka before the independence cf the Congos- and in the P(Ulic that followed
t.he r:l sing of the Congole,,e s.oldi-era flnd gendarmes of ·tbe former . Force ~l!s~ again.st
¥Jhei.r· Belgian oftleere itt JtL1Y of l960sr th~ majpr:tt7 < of the foreign national a ir1 the
Cor~<1 e1~tbe1• :i~eft the eow.1t17 01~~ •. in a< l.at.&r . . · .pha$e1 i~ook ro!uge in the
Kttanga. < thoae- t.rbo departed 1-li•:luded ~s~: ofc,t,.he: · ·. mi.li't;a~r of fie~& and
nott=commis,ioned of i.1icers ~u\d the incum,benti? . ·• .. '\ . • ·•. pq.sitioos in the
M:L~.1i::t?.,1•ics and D1tpa.:rtments o.f the Govarmnentir :ill :.,
.· stat.a.i organizlltionst and in t,b$ lut;e tinancialp mereantil.f and' .ip~U!tt~±~l. . .. . ·• ente~:pr-
isua uhich had held ~ posit loti ot great influenc.e on go.v~~ta.l ;pplt~ • betore L\'\dependence '"
;,·,: :-,_,
```

,:-;- ... ·-". ·• }2o .!n the K.atanga)l condlt.ions rapidl7 rot.urned. tt.o ~no1~l.•1 ; ~O.rt>fgn~rs ..
re~ifli:ed .· . .. . __ '•;; '1; . 4~ . their :poste 1n the adminittt.,r.a.tioni, or enti ered nw onee; a~d,
in : ~~· -r; t•tf$rJ
· . was a steady bttild.,,.u_p in th1a numbet·a of ttirei-gn per$onnel. a,rvd,ng ·,-J J.h~ armi:; • ,.,
.·· .. ,. ..,. ·•·•·c· I:orces,1 Elsewhex-e in the Congo, at·ter the tu.rt.her poli.tieal. crisiij ll, ..
;s,pti~bia~, a p.t'Og:t•assively inCA"ea.iairlg ret1i~n !'lad ot Bfdgiahs and other to~eigr.i,..
iqr.al.J. • · ·•···•··.•·· ... ,-,
also took placuv A large num.btll" of themp espe.eial.ly thOSl'l 'Wi:t,h. t~cllzli.cai ··
quaJ.1ft1oat1onas returned to their prt,viou.s post~ or i~ere traristtt"r~ .. :t:~·.o~eri1> -: .:';. ....
..:-;- ·."'"<'" The in.flow wa~ most marked in the ox-ban eentres~ and e~p>$c!~t7 .trl ~··. e~a,
the p;~ovinc:ial oapitale 'l'ne nost J.mp.,rtwt aspset- of th,ie ·tr~1.1.. .tram F<,1..nt of' n.e1i of
the Delega.itlon 11a nd~aion,;, tta.s the re=, .. entey intp t.fle. bilt6t$, . c C • ;;;f mios't ().r the
tientral. m.ni g·tri esfr and also i:nto some ot the: gov~rrimel\~ > · · departlnenta9 of a number
of' foreigr" (malnly Se.lgia.nJ perstlnnel w-ho ltfe~e :.~ire~
or assumed t?ei&pOtls.:Ulil.i't1es for adv.is.i.ng on mattertl$ of gvverri,me.t1tr •JJOUct/t •· ...
..·,:,y .. ·•
)3t.t The Delt:, gutit,n kr<u3w9 a.s it entarea upon it5 diseus&tonJ ,n.:th the C ,ie
authoritiea9 that some of t h~ea r or ei.gnera were the original :bic~ertt '· t · i..~ " H ·<\+,. <>
posts; others 1t.rsra newcomers~ recruited i.n Bel.glum. ol" on ·t,he apot~ So>• tn ~ii .;;\•."·.:>:
.,._. •:{<-" . ··.:.,-',"'" . ·• . • ······•·······•• the civil fiervicsn or l"es~ t.heir p0sts in it:, through the
nor;.~).. chaan1:1.s· qj th~ t'J.nistry of th~l ,taq~:d~!t. Py,2!!~lS!.6 Ct.her~ were engaged by
"Muct~t-ir~~- ,·~n ~~·~ei.f'-\ ~ ! responeibilitlri t,.htrlr names do no·, &J.tf>ear- t.>n t.he civil
aervtce li.fJtSs, 1·a..i~'. ~11e)"

--prep

4a in the See weere Cefire toe listepmrdence of tre congo, amd in the penis that folliowe tse
cise Lf the erg leee eoidlere arsed gendarme of the former Fopre yd) je agaioat Lieto selgian lf
fiewre in vuly af lea, the majority of the foreicn cath wie te tre singe elther left the seuntry or, in
Later ptupee, tow col ae ds time Nata Tete whe: Secartert included most of the military offi
cveme ant cn cemipetoned of ficere ant the Locumentes of key poweitione Uo tre Mio iat oe act
leper imete fo oe -rvernment, in the pare @tatel orgeniteatione, acd io ttre large Tinancial,
mercantile and indostrial q@nterpriees wo lire nels jwettion of great influence an governmental
poliay before trteceotence. Ae, Lo Ue Pataca, condithae rea idly returted to Snormel";
fomiigneres meme their poste Lo tim aurintetcation, oF emtere: co ones; and, ip pertioular, thare
wae sleaty Cull tol in tre sumeers of fi relan ,;erennne) serving ion the areed forces oewiere the
awigu, after the further political arisis in September, Roprogreteivel. Ii. eaelsg cetum fli of
4eletarne end other foreign nationals ale toc place Aolatge nomoer of trem, eecseclally thee
with teohniceal qoaiiticatione, retard t. tinsic peeticus poote of were traneferred to others, The
inflow was met aecee@t the urten centres, ad espectally in Leopaldville and the proriscial

tajitele. Mie mat important aspect of thie trend, from the pedot of view Sti -elegatla.'e mieeiwn, wae thie re-eitry into the cabinets ~f mel fF tie Set ral min tetries, ar ale ft mime Of te government Fepmrtmernte, of ee cimmpes fo ceig: Ccmeiriy Secglan; perenne] who were given 20 €O mel Fee ee! fee for wlyie lg ow mattere of govertment poltiey, we The lelepyatic aim, ae it enteces van ite tiecuselune with the Gongolese autroritioe, that ome of ese Surelmiere were the original inocumbente of thetr mele, others ere cen mers cel muited 4: teletamocroan the epot. ‘oes entered Siteugt the oormel chammele of the Se nee Lies were egared by "WKirietere® an thelr own tim Civil eervice, ree meri tren ce. teoode Finistr of tre ron. reepone loi lit: Boom ap macoun the civil eervice liete, and they

Default

4. The general situation of foreign pe and ara sin the 31. In the few weeks before the independence of the Congo, and in the panic that followed the rising of the Congolese soldiers and gendarmea of the former Force Publique against their Belgian officers induly ef 1960, the majority of the foreign nationale 1n the Congo either left the country.or, inalater” phase, took refuge in the Katanga... Those who departed included ‘military officerd and non-commissioned officers and the incumbents positions in the Ministries and Departments of the Government, in statel organizations, and in the large financial, mercantile and nterpriges which had held position of great influence on before independence’. In the Katanga, conditions rapidlyreturned to "normal"; ‘their posts in the administration, or entered new ones; and, in was steady build-up in the numbers of foreign personnel serving forces, Elsewhere in the Congo, after the further political orisia progressively increasing return flow of Belgiane and other forel, also took place...A large number of them, especially those with ‘qualifications, returned to their previous poste or were transf ‘The inflow was most marked. in the urban centres, and especially in and the provincial capitals, The most important aspect of thietrend, point of view of the Delegation's mission, was the re-entry into of most of the central ministries, and also into some of the government departments, of number of foreign (mainly Belgian) personnel whowere. or assumed responsibilities for advising on matters of government 336 The Delegation knew, as it entered upon its discussions with authorities, that some of these foreigners were the original incumbents posta; others were newcomers, recruited in Belgium or on the spot. the civil service, or resumed their posts in it, through the normal Ministry of the Fonction Publique. Others were engaged by "Ministers their esponsibility; their names do not appear on the civil service lists,