Suitability of OCR Engines in Information Extraction Systems—a Comparative Evaluation

ZACHARIAS ERLANDSSON
Suitability of OCR Engines in Information Extraction Systems—a Comparative Evaluation

ZACHARIAS ERLANDSSON
Abstract

Previous research has compared the performance of OCR (optical character recognition) engines strictly for character recognition purposes. However, comparisons of OCR engines and their suitability as an intermediate tool for information extraction systems has not previously been examined thoroughly. This thesis compares the two popular OCR engines Tesseract OCR and Google Cloud Vision for use in an information extraction system for automatic extraction of data from a financial PDF document. It also highlights findings regarding the most important features of an OCR engine for use in an information extraction system, when it comes to structure of output as well as accuracy of recognitions. The results show a statistically significant increase in accuracy for the Tesseract implementation compared to the Google Cloud Vision one, despite previous research showing that Google Cloud Vision outperforms Tesseract in terms of accuracy. This was accredited to Tesseract producing more predictable output in terms of structure, as well as the nature of the document which allowed for smaller OCR processing mistakes to be corrected during the extraction stage. The extraction system makes use of the aforementioned OCR correctional procedures as well as an ad-hoc type system based on the nature of the document and its fields in order to further increase the accuracy of the holistic system. Results for each of the extraction modes for each OCR engine are presented in terms of average accuracy across the test suite consisting of 115 documents.
Sammanfattning

Acknowledgements

I would like to thank my supervising professor, Jonas Moll, and my examiner, Cyrille Artho, for their continuous feedback and guidance during the course of the thesis project. I would also like to thank João Saleiro and the rest of the team at Doutor Finanças for their support during my time at their office in Lisbon, Portugal.

Lastly, I want to thank Benoit Baudry at KTH for taking the time to bounce thesis project ideas with me, prior to starting the project.
# Contents

1 Introduction ........................................... 1
   1.1 Problem Formulation ................................. 2
   1.2 Research Question .................................. 3
   1.3 Hypothesis .......................................... 4
   1.4 Research Value and Goals ............................ 4
   1.5 Research Challenges ................................ 5
   1.6 Limitation and Scope ................................. 5
   1.7 Ethics and Sustainability ............................ 6
   1.8 Report Outline ...................................... 6

2 Background ............................................. 7
   2.1 Research Area ........................................ 7
      2.1.1 History of Optical Character Recognition and Information Extraction .................................. 9
   2.2 Theory ................................................ 9
      2.2.1 Optical Character Recognition ..................... 9
      2.2.2 Containerization in Virtualization ................. 15
   2.3 Related Research and Software ....................... 16
   2.4 Knowledge Gap ....................................... 17
   2.5 Document Layout and Features ....................... 17
      2.5.1 Identification Information ....................... 18
      2.5.2 Bank Information and Bank Pages ................ 18
      2.5.3 Credits ........................................ 18
      2.5.4 Resulting Section ............................... 18
      2.5.5 Guarantees .................................... 19
      2.5.6 Amounts Section ................................ 19
   2.6 Summary ............................................ 19
3 Methods

3.1 Tools ................................................. 20
  3.1.1 Node.js ............................................ 21

3.2 Google Cloud Vision ................................. 21
  3.2.1 Google Cloud Storage ............................. 22
  3.2.2 API Call to Google Cloud Vision for OCR Processing 22

3.3 Tesseract OCR ........................................... 23
  3.3.1 PNG Conversion ..................................... 23

3.4 Extraction System ................................. 24
  3.4.1 Reading Input ...................................... 24
  3.4.2 Processing Input .................................... 24
  3.4.3 Producing Output ................................... 34

3.5 Testing and Test Suite ......................... 34
  3.5.1 Construction of Test Suite ........................ 34
  3.5.2 Implementation of Testing Framework .......... 35

3.6 Evaluation of Design Choices .................. 37

3.7 Hypothesis Testing ................................. 37

3.8 Containerization with Docker .................. 38

3.9 Summary ........................................... 38

4 Results .................................................. 40

4.1 Bare-bones Rule-based Extraction .................. 40

4.2 Introduction of Type System ..................... 40

4.3 Attempted Correction of OCR Engine Misdetctions 42

4.4 Final Holistic Solution .......................... 46

4.5 Accuracy by Field .................................... 46
  4.5.1 Tesseract OCR ..................................... 46
  4.5.2 Google Cloud Vision ............................. 48

4.6 Accuracy by Data Type ........................... 50
  4.6.1 Tesseract OCR ..................................... 50
  4.6.2 Google Cloud Vision ............................. 50

4.7 Hypothesis Test ....................................... 50

4.8 Summary ........................................... 51

5 Discussion .............................................. 52

5.1 Design Decisions ..................................... 52
  5.1.1 Divide-and-conquer Methodology .................. 52
  5.1.2 Bare-bones Rule-based Extraction ............... 53
  5.1.3 Type System ....................................... 54
Chapter 1

Introduction

This report presents a thesis project in the field of information extraction in combination with the use of OCR (optical character recognition) engines, and highlights OCR engines’ suitability for use in information extraction systems. The assignment entails trying to find the features of an OCR engine that, when combined with an extraction system, maximizes accuracy for information extraction of financial documents. The financial documents that the system is developed for are so-called owner-protected PDFs, which means that certain operations such as copying and printing are disabled. This means that the PDF cannot be processed as a normal PDF, and must be read using OCR techniques. Two solutions were built on top of two different OCR engines, Google’s Cloud Vision API as well as Tesseract OCR which is an open-source solution originally developed by HP but now maintained by Google [1, 2]. The research of this thesis is concerned with the quality of OCR in information extraction systems with data from a Portuguese financial institution.
1.1 Problem Formulation

The problem treated in this thesis project is to develop a system for information extraction of a particular PDF document, and also to evaluate the usability of the two OCR engines Google Cloud Vision and Tesseract OCR for this specific application. The principal is a company active in the field of economic counseling, where they on a daily basis receive a lot of documents—the most
important one being the *Mapa CRC* which is a statement issued by the Portuguese bank authority (Banco de Portugal) which lists all the credits of an individual. As seen in Figure 1.1, the document is in Portuguese. For an OCR tool to achieve maximal accuracy when performing text recognition, it needs to be trained for the correct language, since OCR tools perform post-processing using a dictionary to restrict the potential candidates for an unknown word. This document currently comes in the form of an owner-protected PDF. This means that certain operations on it are blocked, such as copying and printing, which also means that existing PDF parsing libraries cannot process the document properly [3]. In most cases this protection is removable using a third party library, but an approach that can be applied to all protected PDFs without needing to unlock the file was chosen. This was because it was deemed as important for the system to function regardless of whether the protection could be removed or not, and also to be able to handle potential future changes to the protection methodology. Currently, the data from the documents is re-keyed into a web interface where loan administrators had to manually enter the data taken from the document. The number of PDFs that need to be processed in a realistic use-case for this specific company can be up to approximately 100 a day. The layout of the PDFs is clear-cut with clear guidelines, but the length of the PDFs can vary depending on how many loans a particular applicant has. The info that needs to be extracted from these documents is related to each loan the loan-taker has, such as the monthly payments for that specific loan as well as credit type and the name of the bank where that loan has been carried out.

### 1.2 Research Question

The objective of the project is to examine how accurate a software system for extraction of PDF data built on top of an OCR engine (in this case, Tesseract or Google Cloud Vision) can be made. These systems are tested in a professional, real-life scenario with a large number of documents, to ensure variance between data. The research question of the project is: *What are the characteristics of an OCR engine needed to achieve maximal accuracy for automatic information extraction of financial documents?*

Accuracy in this report is a measurement of to what extent data is extracted correctly by the holistic system consisting of both the OCR engine and the extraction system that makes use of it. This research question also invites to a discussion regarding the strengths and weaknesses of the two mentioned OCR
engines, as well as a more general discussion regarding the most important characteristics of an OCR engine when it comes to information extraction of structured documents. A discussion along these lines can highlight where research effort can be concentrated for these and other OCR engines as well as give hints as to how to remedy potential flaws in OCR engines when developing tools for information extraction.

The sub-question of the thesis, which may be of greater interest to the principal, is: Is the best one of these two implemented systems accurate enough to be used in practice? The principal, i.e. the company where the master thesis project is carried out, expresses an accuracy requirement of 90% accurately extracted data using the solution in order for it to be used in practical applications. The solutions are examined by running the extraction solutions over a range of documents and then comparing the extractions to their fully correct counterparts.

1.3 Hypothesis

Because of previous research comparing the two techniques, the first and main hypothesis of the project is that the IE system built on top of Google Cloud Vision will perform better (= more accurately) than the one built using Tesseract. This is because of the fact that Cloud Vision has been proven to perform better on a wider range of images, and with the document being structured in a certain way, correct word recognition is very important in order to structure it as good as possible in order to facilitate accurate data extraction. A support or denial of this hypothesis could provide some guidelines for software developers aspiring to build an information extraction system for a certain kind of document.

1.4 Research Value and Goals

The societal interest for a scientifically proven method for performing tasks like this is big, since it has the potential to streamline business workflows and save companies and organizations lots of resources. From their view, a desired outcome is a scientifically proven tool for accurately and efficiently extracting the needed information from these kinds of documents and thus reducing or even removing the need of manual re-keying of such data. When it comes to conducting a scientific study, a desired outcome is a quantitative analysis
of information extraction systems built on top of different OCR engines that should be able to be used as baseline for making decisions on which tool might work best in different use-cases and also what the features are that makes it the best fit. The objective of the thesis project can be deemed fulfilled if it can be shown scientifically that conclusions can or cannot be drawn when it comes to deciding between different methods for information extraction for a given application. The other objective, corresponding to the sub-research-question, is to judge whether any (or both) of the tools are accurate enough to be able to be put in production by the principal. The finished work is also of interest to software engineers aspiring to build information extraction systems built on top of OCR engines in order to be used for scientific or professional applications. The report also highlights some weaknesses of current solutions, and thus provide some pointers for future development of systems used in similar situations.

1.5 Research Challenges

The challenges of the project lies in trying to structure the output received from the OCR engines in such a way that the data could be extracted with as much accuracy as possible. This presents itself with a lot of challenges, as text may be recognized in an order that is hard to predict beforehand. To remedy some of the shortcomings of the OCR engines, certain inherent attributes in the document can be used. As an example, a name is not expected to consist of only numbers—if it looks like it does, the name must have ended up somewhere else and this information may allow for it to be found.

1.6 Limitation and Scope

The project is limited to only one specific type of document as opposed to a number of documents from the same sector. A problem like this could be interesting to look at and compare the performance between the two OCR engines over a greater range of document types. However, this is not possible considering the limited scope of the project. Additionally, there is only sufficient time to be able to test two different OCR engines. It would have been interesting to include additional OCR engines and see how their performance compared to Tesseract’s and Google Cloud Vision’s.
1.7 Ethics and Sustainability

As the research is done in the realm of computer science, its effects on society are implicit. Automation of mundane and monotonous tasks may either make employees superfluous or give them the possibility to spend their time performing more meaningful tasks. Depending on the outcome, this can have both positive and negative outcomes. Hypothesizing about this further however lies outside the scope of this project. If processing of certain documents could be automated instead of done manually, associated labour costs would be reduced which could make services such as health care cheaper and allow for decisions regarding issues such as financial aid to be taken quicker. Effects such as these have the possibilities to make life easier for individuals in society with greater needs than others.

1.8 Report Outline

In Chapter 2, Background, the theories that the performed work depends on are explained thoroughly. It is also here related research can be found as well as the project’s hypothesis. Chapter 3 deals with the methodology chosen in this project, such as the chosen programming language and how the communication with the OCR engines was handled. It is also here that the test suite and how it was built is explained, as well as how it is used throughout the project. Chapter 4 deals with the achieved results of the project for each of the four extraction modes, with tables and figures granting an overall view of the outcome. In Chapter 5, the results and what they implicate is discussed freely. The final chapter, Chapter 6, concludes the project in its entirety and also includes areas that could be discovered further in future research.
Chapter 2

Background

In the field of PDF parsing and information extraction, there are several obstacles to overcome in order to, as accurately as possible, be able to automate the tedious tasks of manually extracting data from PDF documents. The need of re-keying data can also be error-prone, seeing as the human factor is introduced into the process [4]. Automating this process, or at least parts of it, has the potential to save a great amount of resources for organizations and companies. One problem that is a bit closer to what one might encounter in a typical use-case in the private sector is the matter of extracting specific data from a greater set of documents.

2.1 Research Area

The main field of this research can be contributed to the area of information extraction (IE). The point of an IE system is taking raw material obtained from a source and then refining and reducing it into a structured form that can hold the information that the user actually finds useful in the raw material [5]. One example of this could be to decide which department of a university a certain dissertation belongs to for a corpus of documents [6]. The main challenges of information extraction lies in extracting information from unstructured documents—this could be sources such as news articles, public announcements and the like. Because of this, natural language processing (NLP) has a clear role in IE. Factors such as well-defined extraction tasks, the use of real-world texts as input as well as easy-to-measure performance makes it an interesting area for many scientists in the area of NLP [5]. However, NLP is not really needed as a method in this thesis project, since the PDFs that are being dealt with throughout this project are semi-structured and not completely
unstructured, which may be the case in news articles for example [7].

The field of IE lies between the fields of information retrieval, IR, and NLP. Information retrieval concerns itself with the problem of retrieving written information, in a general sense. The need for work in this area grew as the computerization of society increased, and in 1950 the first concrete descriptions of how this could be done materialized [8]. The most notable applications of IR are search engines such as Google and Yahoo! Search.

The raw material that the IE is performed on can be obtained through different methods. It could be raw text scraped from online documents or websites. It could also be text that, for some reason or other, needs to be read using OCR, or optical character recognition. OCR as a scientific field is a subset of the wider field known as pattern recognition [9]. Pattern recognition is a broad term that implies recognizing patterns and regularities in data in a lot of different forms—this can be anything from waveform classification to classification of geometric figures [10].

OCR is the process of turning text in images—be it hand-written or printed—into machine-encoded text. It is one of the most important image analysis tasks that we encounter, and deals with real-life problems such as localizing vehicle license plates, reading text for visually impaired users as well as understanding hand-written office forms [11].

OCR as a commercially available software first made its entrance in the 1970s with a company called Recognition Equipment Inc. that developed a system for automatically scanning receipts from gasoline purchases [12]. After that, it was applied for a number of different scenarios such as passport processing and postal tracking. Since 2008 Adobe has included support for OCR on any PDF file. One of the most recent big technical advancements in the field was the introduction of the MNIST database in 2013 which is a database with handwritten digits used for training image processing systems [13].

The intersection of IE and OCR is interesting, since it is close to automating a lot of monotonous tasks, and it also has the potential to change a lot of jobs ranging over a lot of sectors. Manually entering data into a system is slow and resource-intensive, and hence also expensive for companies and organisations [14]. It has been shown that manually entering or re-keying data into a system, hence introducing the potential for human error, also increases the error rate
of the entering of said data [4]. Depending on where this error happens, the consequences can in certain situations be grave (such as in health care). These are problems that can be remedied or even solved completely with automation.

To recap, the topic of the project more specifically is to examine how accurate a software system for extraction of PDF data built on top of an OCR engine (in this case, Tesseract or Google Cloud Vision) can be made. This will then be tested in a professional, real-life scenario with a large number of documents.

### 2.1.1 History of Optical Character Recognition and Information Extraction

The first example of commercial IE software was called ATRANS and was designed to handle international banking telexes [5]. The developers of that project took advantage of the fact that the format of the telexes was very predictable. A system that made use of more recent advancements in NLP was the Jasper system, that was used for extracting information from corporate earnings reports [5]. Further developments in the field of IE are tightly coupled with advancements in the field of NLP. In the early stages of IE, competition-based conferences were held every few years to highlight recent advancements [15].

One of the most pivotal moments in the history of OCR research was the development of the omni-font OCR, which could perform character recognition on virtually any font, that went into use in the late 1960s [16]. Nowadays, a plethora of engines are available ranging from open-source to proprietary software [17]. The fields of document image analysis (DIA) and IR are both supersets of OCR, as many of its descendants require OCR. The field of DIA concerns itself with algorithms that are used to obtain a computer-readable description from digital images, and most often rely heavily on OCR as almost all line drawings contain text [18].

### 2.2 Theory

#### 2.2.1 Optical Character Recognition

OCR as a process generally consists of several sub-processes, in order to be able to perform the character recognition as accurately as possible. In broad
terms, these sub-processes are 1) pre-processing, segmentation and classification, 2) character recognition and 3) post-processing [11].

**Pre-processing**

The main objectives of pre-processing is to optimize the image as much as possible in order for the actual character recognition to be as accurate as possible. This can be done using a range of different techniques [11]. The first step of pre-processing generally consists of *sub-sampling* or scaling down images [19]. Sub-sampling both increases speed of processing and may also potentially produce more accurate results for certain tasks. Different methods can be used for this purpose, such as *nearest neighbor interpolation* [19]. In certain applications de-skewing the image is vital to the quality of the character recognition of the document—but before skew estimation can be done, recognizing the text and non-text segments of the image is important [19]. This can be done by making use of a neural network—examples of usable networks are *Radial Basis Function neural network* and the *STN-OCR neural network* [19, 20]. De-skewing the digital image can be done by calculating the *Cumulative Scalar Product* (CSP) between windows of text blocks that are filtered with *Gabor Filters* at different angles, where the skew angle is found to be the maximal CSP between the text blocks [19]. The result of de-skewing text in an image can be seen in Figure 2.2. The step following de-skewing of the text in the image is to *binarize* the image, in order to be able to process it faster as well as reducing the storage space of the image [19]. Binarization is done by converting pixel values that range between 0 and 255 to pixel values that are either 0 or 1 (black or white pixels). *Noise removal* is then performed on the binarized image, which is done by using a moving window that traverses the image with a smaller window inside it that sets the pixels of the inner windows to 0 if they are the only ones that are non-zero in the bigger window [19]. This can be seen in Figure 2.3. The *segmentation* and *classification* stages are the
Figure 2.2: Text skew correction [22].

Figure 2.3: Noise removal with a moving window with an outer size of 5x5 and an inner size of 3x3.
Segmentation and classification is the process of dividing the binarized and noise-reduced image into different blocks, where a block can either be a text block or a non-text block [19]. The decision regarding whether a block is a text block or a non-text block is made considering the amount of text in the region compared to the total area of the block—this is done by comparing the regions found in the binarized image with the areas in the unbinarized image to calculate the text-to-area ratio in that block [19]. Different thresholds are then used to decide whether a block is a text block, text/non-text merged block or a fully non-text block. The result of a classification regarding text and non-text blocks can be seen in Figure 2.1.

For the text blocks, further segmentation is needed. The text blocks need to be divided into lines, words and characters in order to be able to be processed properly [19]. Segmentation inside text blocks is done recursively, where a text block is divided into lines—a line is divided into words—and a word is divided into single characters. This can be seen in Figure 2.4. When it comes to the line segmentation, a threshold value for the horizontal projection on the y-axis of the potential blocks can be used [19]. For word and character segmentation, different techniques are available. One example of this is Kimura
and Shridhar’s algorithm and its four steps; initial segmentation, multicharacter detection, splitting module and recognition [23].

Character Recognition

For the character recognition, most modern software make use of feature detection [24]. This means dividing data into a number of domain-specific features. For medical patients, this could be a set of symptoms [24]. In our case, written characters, this could be features such as lines, closed loops, line intersections and so forth [24]. Using the features of a character, decisions can be made regarding what character a character in the image actually represents. For the classification, the k-nearest neighbor algorithm can be used [25]. This algorithm classifies objects based on proximity to training examples in the feature space, where the most common class among its $k$ nearest neighbors is chosen.

For Tesseract, which has been made open-source, the algorithm operates in two steps [26]. For the first step, the unknown characters are iterated one by one. In this first iteration, a class pruner creates a list of candidate characters that the current unknown character might correspond to [26]. This list is made by iterating through an unknown’s features. For each feature, a look-up is made to find classes that can match for a specific feature and a bit-vector is created, after which all features’ bit vectors are summed to form the list that will be used in the second step [26]. After the first step is finished, all the characters that are deemed satisfactory are passed as training data to an adaptive classifier in order to be able to, more correctly, recognize characters further down on the same page [26]. Since information is learned throughout the page, the beginning of a page cannot be classified as accurately as the bottom of a page. Because of this, a second pass of the whole page is done in order to be able to utilize all the training data [26]. A simple example of the algorithm can be seen in Figure 2.5. It has been demonstrated that OCR engines can benefit from the use of an adaptive classifier such as in the case with the Tesseract engine [27]. The role and the strength of the adaptive classifier (as opposed to the static classifier utilized in the first step) is its font-sensitive properties, since it is trained on only the contents of one document delivered as output from the static classifier [26]. Apart from the difference in training data between the classifiers, the adaptive classifier is better able to distinguish between upper and lower case characters due to baseline/x-height normalization [26].
The adaptive classifier moves along the characters of the image, creating candidate lists and classifying them. At the same time, the adaptive classifier is learning.

The A was deemed classified with sufficient confidence. The classifier moves along.

The B could not be classified with sufficient confidence. It will be revisited during the second run.

The C could be classified with sufficient confidence, the D could not. They will be revisited during the second run using the learned information.

All the letters unclassified from the first step are revisited and classified.

Figure 2.5: The Tesseract algorithm with a simple example consisting of four characters to be classified.
Post-processing and Making Use of the OCR Results

Post-processing can be utilized in order to further improve the accuracy of recognition in OCR techniques. In the case of Tesseract, a dictionary is utilized in order to put a constraint on the viable candidates for a detected word [26]. In other cases, a simple API call to a spelling suggestor can be utilized in order to improve OCR accuracy [28].

For the extraction system implemented in this thesis project the extraction is done using the plain text file resulting from the OCR processing. This involves manipulating the text file output in different ways, before finally extracting the data. Examples of manipulation could be removing words that are not needed (such as generic footer text), and separating known keywords from their respective data fields. Because of the nature of the documents (them being purely digital), the difficulty mostly lies in trying to structure the recognized text rather than deal with potential errors from the OCR-processing.

2.2.2 Containerization in Virtualization

Thorough testing of the solution is performed in Docker containers. Containerization is a form of virtualization technology [29]. Virtualization (more specifically virtual machine technology) enables a single physical machine to run two or more operating systems at the same time, for purposes such as space and time multiplexing and for applications in fields such as cloud computing and high performance computing [30]. The multiplexing is facilitated by a privileged kernel known as a hypervisor which in turn provides the illusion of one or more real machines [30]. This can in turn be used in order to separate software installations from the physical hardware configuration, which can be useful for software developers in order to avoid incompatibilities and problems with conflicting software packages [30]. Containers as a technology are less resource and time-consuming compared to VMs (virtual machines) [29], and a visual representation of their differences can be seen in Figure 2.6. The main points that a container can provide a developer is a lightweight portable runtime—capable to develop, test and deploy applications to a large number of servers as well as interconnect containers [29]. A container image is based on lightweight images whereas VMs are based on full, monolithic images [29]. The container itself consists of a writable layer in a system architecture that lies on top of any number of different, read-only layers where all layers consist of images. Examples of these layers could be Apache images, Emacs images and Linux images [29]. Containers can be of great help to developers or even
just users of software to cut down considerably on the time it takes to get a needed environment up and running, by pre-packaging the needed software along with its necessary dependencies [31].

Figure 2.6: A comparison of the architecture for Virtual Machines and Docker containers [32].

### 2.3 Related Research and Software

The Tesseract engine has, since it has been available for a longer period of time, been researched more thoroughly than Google Cloud Vision, but previous work has been done comparing Tesseract and Google Cloud Vision and their purely character recognising capabilities as well [33, 34]. However, in this study and others like it they were not tested for IE applications. For the cases where they were tested against each other, Google Cloud Vision outperformed Tesseract OCR [33, 35]. These tests were not performed on any kind of structured document, but rather on standalone images containing text in some format—be it hand-written, printed or in pictures. Their accuracy has not been compared for structured documents earlier, and their feasibility for information extraction of documents has not been discussed.

When it comes to unprotected PDFs, a wider range of options are available, since direct access to the text and other elements is allowed. Previous work has for example examined the possibility of converting the data of PDF documents into different formats, such as NXML [36, 37].

Extracting data from unprotected PDF documents is known as PDF parsing—this is usually done by converting the contents of the PDF into a structured form. There are also a wider range of tools and frameworks available for this purpose. One case-study has been performed using the open-source library
PDFminer for extracting information regarding the department under which an academic dissertation has been written [6]. This experiment was performed on scanned PDFs rather than machine-produced ones. Other experiments have also been performed using academic papers as the ground for information retrieval [38]. There have also been experiments where more refined operations have been performed with the extracted data, such as classifying the rhetorical category of a certain text section [39].

Existing proprietary software for automatic processing of standardized documents such as bank excerpts, invoices and receipts work by either specifying exactly where (in co-ordinates) in the document a certain field appears, or by specifying certain keywords in the document. One of the most used ones today is DocParser that handles a plethora of user cases ranging from performing OCR on scanned PDFs to performing extractions on unlocked PDFs uploaded to their cloud platform [40].

2.4 Knowledge Gap

Many of the mentioned experiments have been performed in the academic world, but similar experiments have not been performed for documents of different origin, such as documents from different sectors that are formatted in a different way compared with scientific papers. This work is also scientifically relevant, since not much previous work has been done in comparing different open-source OCR engines specifically for information extraction of documents. Previous work has mainly been occupied with testing one single solution for a given data set [6], or a comparison between the two but for strictly OCR purposes [33].

2.5 Document Layout and Features

The document that is processed in this thesis project, Mapa CRC, has a clear and robust format that makes it a good candidate for the development of an information extraction system. The document consists of a number of fields that all correspond to a singular value; the objective of the information extraction system is to, as accurately as possible, be able to extract these fields’ values. The layout and format of the document will be presented in this section in detail, as well as some domain-specific features of the values.
2.5.1 Identification Information

Figure 2.7: Header containing the identification details of the individual, as seen on each page of the PDF.

The Identification Information of a customer for whom the Mapa CRC is created is listed at the top of each page of the document, as in Figure 2.7. This consists of a name, identification type as well as an identification number—these three values were all extracted from the document.

2.5.2 Bank Information and Bank Pages

The document consists of bank pages, which hold information that needs extracting, and summary pages and other unnecessary pages that the principal has expressed no need for. An example of a bank page can be seen in Figure 1.1. At the top of each bank page, there is a value that needs extracting, which is the name of the particular bank the following credits belongs to. This is denoted by the string "Informacao comunicada pela instituicao...".

2.5.3 Credits

The main bulk of information that needs to be extracted from the documents is located in the credits section of a bank page. The credits section is here defined as the section of a bank page in which one or two separate credits are located. The idea of the extraction system is to divide these credits (if there are two) and to extract their respective values in such a way that it is clear which values come from which credit. For the example seen in Figure 1.1, the particular bank page consists of two separate credits which are the two boxes of fields and values with a grey border around them. Each credit can be further divided into smaller sections.

2.5.4 Resulting Section

The so-called resulting section of a credit consists of the first seven fields and their values, as well as an optional section for guarantees which can be seen as
an even smaller grey box in the example of Figure 1.1, with additional values inside.

### 2.5.5 Guarantees

The guarantees section is an optional section of a credit, which in the test suite contained up to three triplets of values, but potentially it may also be able to hold additional triplets. Extraction of this section is done together with the entirety of the Resulting section, as separation of sections that are separated horizontally in the document is a difficult task, because of the manner the document is returned from the OCR processing.

### 2.5.6 Amounts Section

The amounts section is the section consisting of exactly eight values at the bottom of each credit. This section is separated from the Resulting section in the implementation of the extraction system to facilitate modularity in the system. This section always looks identical between credits (apart from the values of course), and the format is very standardized and easy to predict during the extraction stage.

### 2.6 Summary

The main area of this research is information extraction. The raw material that the information extraction was performed on in this application is plain text resulting from OCR processing of the financial document *Mapa CRC*. OCR as a process consists of three sub-processes; pre-processing, character recognition and post-processing. The information extraction took place after the post-processing stage and also included some additional post-processing methodology. The holistic system was containerized using Docker for testing purposes. The hypothesis, based on previous findings in the current research area, was that the information extraction system would perform better on the Google Cloud Vision results than the Tesseract results. The nature of the document *Mapa CRC*, with it being structured, allows for information from the document being used in the extraction stage. This can be information such as expected values, which can be used to reorder values that have arrived out of order.
Chapter 3

Methods

In this chapter, the methodology applied in the project is explained. The tools and how they are used is explained in further detail, and a description of the extraction system implemented follows. How the evaluation of the system was carried out, as well as the Docker containerization of the project, can also be found here.

3.1 Tools

![Google Trends search for the top 5 free/open-source OCR engines available [17].](image)

Out of the open-source/free OCR engines, the two most popular ones based on Google Trends’ data are by far Google Cloud Vision and the Tesseract OCR
engine, as can be seen in Figure 3.1. Based on this, the thesis project dealt with these two top engines, Google Cloud Vision and Tesseract. It makes the most sense for the industry to test the two top-mentioned tools, as they are most likely the two that new developers are looking to decide between as well. Tesseract was originally developed by HP Laboratories as a proprietary software, but was taken over by Google in 2006 and made available open-source shortly thereafter [1]. Tesseract is written in C++ and available in its entirety on GitHub, and has support for Portuguese. Google Cloud Vision is a part of the Google Cloud Platform which is a suite of cloud services hosted by Google [2]. Google Cloud Vision offers pretrained models in an API available for free (up to a certain API call quota) through their Cloud platform. Google Cloud Vision has pretrained models for Portuguese as well.

3.1.1 Node.js

JavaScript was chosen as the programming language for implementing the extraction system, and to avoid the need of executing code in a browser the runtime environment Node.js was used for client-side scripting. Node.js was used because Google Cloud Vision supports it through a client library, and additionally the Shelljs library is available for Node.js which allows for implementation of Unix shell commands on top of the Node.js API [41, 42]. Executing Unix shell commands from Node proved useful for the Tesseract implementation of the information extraction system. This is because Tesseract is available, in addition to their C++ library, as a command line tool. Using Shelljs, the command line tool can be used from a Node script.

Performance bottlenecks and memory management were not the top priorities for this project, and thus performance limitations of certain languages was not taken into consideration. This means that a language in which projects are easy to set up can be used. Also, a plethora of tools are available for free via NPM, or the Node Package Manager [43].

3.2 Google Cloud Vision

Google Cloud Vision is the first of the two OCR engines that was used for the information extraction task. It is available through an API call, and Google also supports client libraries in a plethora of languages—C#, Go, Java, Node.js, PHP, Python and Ruby [41].
3.2.1 Google Cloud Storage

Google Cloud Storage is required for the API call to Google Cloud Vision. Google Cloud Storage is an online data storage service provided by Google that lets you as a developer store data in what they call *Buckets*. Google Cloud Storage’s API lets you upload files to your buckets, download files from your buckets as well as move things between your buckets. The reason why Google Cloud Storage is needed for Google Cloud Vision is that the API call to the OCR engine does not let you upload files explicitly, but instead takes file(s) in your buckets as input. After Google Cloud Vision has processed the file(s), the file(s) are placed in another bucket that you specify in the request object in your API call. When you have received the callback from Google Cloud Vision that the file is fully processed, you can then download the file from your specified bucket. Because of this, the fully dynamic infrastructure of processing a PDF document from start to finish first included a call to Google Cloud Storage in order to upload the PDF to a bucket holding unprocessed PDFs. The dynamic process also included another call to Google Cloud Storage, this time for downloading the JSON resulting from the OCR processing of the original PDF.

3.2.2 API Call to Google Cloud Vision for OCR Processing

When the PDF is placed in a Cloud Storage bucket and is ready for processing by the OCR engine, an API call is made to Google Cloud Vision. In the API call you can specify certain attributes of the input object, such as whether to include results derived from the geo information of the document. All these attributes are specified in the request object that is included in the API call. In the request object you also specify the language(s) present in the PDF file.

In the callback to the API call, Cloud Vision lets you know whether the processing was successful or not. If it was, you can be sure that Cloud Vision has created a JSON file in the designated output bucket. This JSON file, and consequently the JavaScript object, holds the output from each of the pages of the PDF in a nested structure where each page is an object with one of the fields being the detected text in plain text, as one long String. The JSON is structured as in Figure 3.2.

For the information extraction system, the second step consisted of calling
Figure 3.2: Structure of the output object produced by Google Cloud Vision.

the Google Cloud Vision API. The first step, as mentioned previously, was to upload the PDF to the specified Google Cloud Bucket. When that process was finished, a request object to the Google Cloud Vision API was built using the filename and the bucket of the uploaded file, along with options regarding the OCR processing. The options here are the manner in which the PDF should be processed (as a document, and not as any image containing text) as well as the language the text is written in, which in this case is Portuguese.

3.3 Tesseract OCR

When it comes to Tesseract’s OCR engine, it is both available as a C++ library and as a command line tool. To get the solution up and running as quickly as possible, the approach of using a library for executing shell commands from Node was chosen. This also allowed for re-use of big portions of the extraction engine between the two solutions which saved a lot of development time.

3.3.1 PNG Conversion

Tesseract, as opposed to Google Cloud Vision, does not accept PDFs as input to their OCR engine. Instead, a conversion must be done to an image format, which was chosen to PNG. Tesseract make use of the image analy-
sis open-source library Leptonica, thus only allowing formats that Leptonica accepts—BMP, PNM, PNG, JFIF, JPEG and TIFF [44]. The approach chosen here was to execute a shell command to GraphicsMagick, a command-line tool for image processing, in order to convert the PDF files into PNG format [45]. Here, each page of the PDF file is converted to a separate PNG file, resulting in $n$ PNG files where $n$ is equal to the number of pages in the PDF. After this, Tesseract can be called with a list of all these $n$ PNG files for processing them consecutively. Tesseract then outputs the recognitions in TXT format.

### 3.4 Extraction System

#### 3.4.1 Reading Input

The tools produce their output as either a long String in a JavaScript object (Google Cloud Vision) or as a set of TXT files (Tesseract). Either way, the output consists of the recognized text in the image as plain text. The next task is to split this text into a manageable format. This is done by splitting the input at every new line, which results in an array where every element of the array corresponds to one line of the resulting plain text.

#### 3.4.2 Processing Input

With the data in the same format for the two solutions (i.e., an array consisting of lines of the plain text returned by either of the two engines), the data is then due to be processed and the necessary fields extracted. This is done by processing each page of the PDF consecutively.

**Pre-processing Input and Type System**

Depending on the OCR engine used, some pre-processing is needed in order to achieve maximal accuracy of the data extraction. This step is performed before the actual processing and extraction of data. Here, potential incorrect extractions may be remedied to some extent.

It is also here *value types* are encountered for the first time. All the values in the array of lines are strings—but to increase the accuracy of the extraction engine, custom types for the values of the strings are implemented. This is usable for a few scenarios. One of these scenarios is when a set of values are
• TYPE_DENOMINATION
• TYPE_NUM
• TYPE_ID
• TYPE_DATE
• TYPE_TEXT

Figure 3.3: List of the classes of the ad-hoc type system.

extracted in incorrect order—but they may still be in correct order after sorting them in order of types. The value types that are present throughout the document can be seen in Figure 3.3.

TYPE_DENOMINATION is the type for strings denoting a denomination, i.e. a numeric monetary value. Since the document is in Portuguese, these are in the currency of € (Euros). If there’s a "€" sign in the string, it is fairly certain that the string is representing a monetary value and nothing else. TYPE_NUM and TYPE_ID are both types of strings containing only numeric values. The way they are distinguished is that TYPE_ID are numbers with 4 digits, and TYPE_NUM all other numbers. This assumption is fairly safe judging by the context of the document. This is because the fields that have values of TYPE_NUM typically have very low values—since they are representing the numbers of debtors for a certain loan and the number of guarantees a certain credit has. It can be seen as extremely unlikely to have ≥ 1000 debtors for the same loan or for a loan to have ≥ 1000 guarantees. TYPE_DATE are strings with the format X-Y-Z, where X is a year starting with either 1, 2 or 9 and Y is a two-digit number between 01 and 12, and Z a numeric two-digit value between 01 and 31. 9 is needed because credits can lack an end date, and in that case the end date is listed as 9999-12-31. TYPE_TEXT is the type for strings that do not fit in any other category, which makes the type system cover 100% of the strings that are due to be processed.

Consider the order of the keys and their values from Figure 3.4. The types of the values for the different fields can be seen in Table 3.1. Let us say that the values are ordered like in Table 3.2 by Tesseract or Cloud Vision: If they were read in order of appearance, they would all be read incorrectly (their order is shuffled between the two tables). But if they are read in order of appearance
<table>
<thead>
<tr>
<th>Field</th>
<th>Value Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total em dívida</td>
<td>TYPE_DENOMINATION</td>
</tr>
<tr>
<td>do qual, em incumprimento</td>
<td>TYPE_DENOMINATION</td>
</tr>
<tr>
<td>Vencido</td>
<td>TYPE_DENOMINATION</td>
</tr>
<tr>
<td>Abatido ao ativo</td>
<td>TYPE_DENOMINATION</td>
</tr>
<tr>
<td>Potencial</td>
<td>TYPE_DENOMINATION</td>
</tr>
<tr>
<td>Prestação</td>
<td>TYPE_DENOMINATION</td>
</tr>
<tr>
<td>Entrada incumpr.</td>
<td>TYPE_TEXT</td>
</tr>
<tr>
<td>Periodicidade</td>
<td>TYPE_TEXT</td>
</tr>
</tbody>
</table>

Table 3.1: Fields of the PDF and the type of their respective values.

Table 3.2: Example of an ordering of values read from the Tesseract or Cloud Vision’s output.
taking their type into account, they are all read correctly. The order of the TYPE_TEXT fields are mutually correct (Entrada incumpr., Periodicidade) as well as the TYPE_DENOMINATION fields.

The last action that is taken in the pre-processing stage, is to standardize the strings to be processed. This involves standardizing the format of TYPE_DENOMINATION strings, adding a white space between the numeric value and the € sign (if needed), turning 107€ into 107 € for a more standardized output. Similarly to how the OCR correction is handled, if the type checker is disabled standardization is performed at set points throughout the process.

**Incorrect OCR Classifications**

The remedies for incorrect recognitions depend on the OCR engine used for the specific solution. For the Tesseract engine, it is quite common that a "7" gets translated as a "/"-symbol. Again, with domain knowledge, there are no encountered fields that are supposed to contain nothing but numbers (and potentially €) and "/" signs—so during the type check, if a string contains only numbers when the "/"-sign is stripped—it is assumed that the type of that string is TYPE_NUM (or TYPE_ID). During the pre-processing stage, all the lines of the document are iterated upon and if the type-checker has determined that the string is not of TYPE_TEXT, it is passed to a function that replaces all "/" signs with "7". With Tesseract, there are also cases where a "0" gets wrongly detected as an "O". This means that a denomination value of "0 €" would be returned as "O €" by Tesseract. The method to deal with this is very similar to the approach chosen for the "7" to "/" misdetection—namely, adding an exception for it in the type checker as well as a fix for it in the pre-processing of the lines of the document. Table 3.3 shows a selection of corrections that can be made. There also exists situations where remedying or even detecting...
Table 3.3: Examples of possible corrections that can be performed.

<table>
<thead>
<tr>
<th>Uncorrected Text from OCR Engine</th>
<th>Text Corrected by IE System</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text/ 199/-12-31</td>
<td>Text/ 1997-12-31</td>
</tr>
<tr>
<td>1/</td>
<td>17</td>
</tr>
<tr>
<td>1///</td>
<td>1777</td>
</tr>
<tr>
<td>O €</td>
<td>0 €</td>
</tr>
<tr>
<td>2009</td>
<td>2009</td>
</tr>
<tr>
<td>10/ €</td>
<td>107 €</td>
</tr>
</tbody>
</table>

incorrect OCR classifications cannot be done. This is in cases where numbers get mixed up by either of the OCR engines (although they are more prevalent for Tesseract). Situations like this could be when a 7 gets classified as a 1 (or vice versa), 107 € -> 101 € for example. In situations like this, it is impossible to know if 101 € really is the correct value or not. Tesseract also has problems with some Portuguese words, as they contain a lot of apostrophes and letters that may be hard to correctly detect, such as c-cedilla (Ç).

For testing purposes, a version of the extraction system that does not make use of the corrections was also tested in order to evaluate the effectiveness of the implemented OCR-corrections. This means that all values in the left column of the table above would be left as is. For the testing stage, to properly be able to test each design choice independently, it must be able to isolate them. This means that it must be possible to correct misdetections without the use of the type checker as well. To achieve this, instead of doing a one-pass over all the lines before processing them, they are processed at different stages throughout the extraction. For example, it is known that there should not be slashes in values for the personal number field—so the correction can confidently be performed there, even without knowing the type of the value to be put there—since slashes are not wanted in the final extraction. Along these lines, this can be done at various points throughout the process.

**Bare-bones Rule-based Extraction**

For the sake of evaluating the design choices made throughout the project, with the type system as well as the OCR correcting steps, a bare-bones rule-based extraction is tested in order to be able to compare the different solutions. This also has the possibility to shed some light on the problem for aspiring devel-
opers that want to develop information extraction systems for documents that may or may not have data fields which are clearly classifiable—generally, the less conclusions that can be drawn regarding expected outputs of certain fields, the less accuracy should be expected. In the bare-bones version of the extraction system, information was only processed with regards to indices in which it was encountered, and not with regards to how the value looks explicitly.

**Divide and Conquer**

The methodology that is applied throughout the extraction pipeline is to attempt to subdivide the document and its contents as much as possible, in order to facilitate a methodical extraction scheme. This also makes the system easier to debug. How the document’s contents are optimally divided is shown in Figure 3.5. This is done by using certain cut-off keywords, which makes this a completely rule-based approach. An example of this could be "Tipo de Responsabilidade", which in the extraction system marks the start of a credit (and, of course, the end of the previous if there is one). Using the sections of the document as modular units, extractions can be performed.

**Extracting Values**

Extractions are almost exclusively made on modular sections of the document. This facilitates a much easier extraction process should the division have been done sufficiently accurately. However, the identification information at the top of each bank page as well as the name of a bank (Informação comunicada pela instituição:...) can be done straight from the "raw" document as extraction of those values does not warrant a thorough modularisation. A reason for this is that these are the only values that normally get extracted together with their keys (on the same line), which makes finding and extracting them a simple task. For the rest of the values throughout the document, however, the values are extracted separate from their keys. In these situations, the divide-and-conquer methodology comes in handy.

On the modular sections, different approaches can be pursued in order to facilitate an extraction that is as accurate as possible. The approach here is depending on whether the type system and/or the OCR correctional approaches are used or not. If the type system is used, values can be type-checked in the modular sections and sorted according to type. If it is not used, indices of values are considered and the values are assigned according to their order, without looking at the values explicitly. If the OCR correctional procedures
Figure 3.5: An example document displaying the manner in which the divide-and-conquer methodology is applied.

are used, certain OCR misdetections that have been made can be corrected to an extent. It is also on the modular sections that certain keywords are removed that are known to be present in the document but are not needed for the actual extraction. Examples of these could be generic headers or the like, so in order to not accidently extract these as values they can be removed if needed.

For extracting the values of the so-called Resulting section, which is the sec-
tion containing the first seven values of a credit as well as potential guarantees, a reliable division between the first seven values and the guarantees using keywords cannot be made, since these are not separated vertically. This is a problem, since both OCR engines attempts to process the document from the top left corner to the bottom right corner, reading from left to right and top to bottom. This is instead dealt with by the type system—or a check of the number of encountered values should the type-checker be inactivated. If the type system is used, it can be deduced that there are as many guarantees as there are values of denomination type in the section. If it is not used, the conclusion is that there are as many guarantees as there are additional values (other than the initial seven) divided by three (Tesseract) or two (Google Cloud Vision). This is because Google Cloud Vision cannot confidently find the single-number value that’s usually denoting the *Numero* field of a guarantee.

**Making Use of the type system**

During the extraction stage, the type system introduced earlier in this section can be utilized. The way that the type system is used in general terms is that the lines of a modular section are iterated, typed, and kept in an array holding values of the same type. The fields of the section are then iterated, and a look-up of what kind of value that the specific field is expecting is done. According to the result of this look-up, a value is then popped from the respective array. In some cases a field might have a value that is one of two possible types: for this scenario, a look-up is done in both of the two type arrays where the size of the array and the number of values left decide which value is chosen in order to remove the possibility that a value that is meant for another field is chosen.

**Ad-hoc Solutions**

![Figure 3.6: A Resulting section with a value spanning several rows.](image)

Some ad-hoc solutions are needed for certain known features of the OCR engine or the PDF document. Knowledge from performed extractions and data set examination are the reason as to why these design decisions have been
made. One example of this is text values spanning several lines of the document, and thus getting detected (rightfully so) by the OCR engines as two separate lines. An example of this can be seen in Figure 3.6. However, in order to perform a correct extraction both these lines need to be extracted correctly, and assigned to the same field. The way this can be remedied is by iterating over all the value lines of a modular section, and concatenating text lines if they belong together. Using the nature of the document this can be done confidently, because text values always start with an upper case letter. The situation is remedied by iterating over the values, and concatenating a line with its predecessor if it does not start with a capital letter. These fields all start with a capital letter, so if a line starts with a lower case it should belong to the previous line. This would transform the array ["Line 1", "line 1 continued", "Line 2", "Line 3"] into ["Line 1 line 1 continued", "Line 2", "Line 3"].

In certain situations, separation of keys and values may be warranted. This is for cases when keys and their values end up on the same line—which is not the expected behaviour. As an example, they might have gotten extracted as in Table 3.4. Running the array above in a function that separates the values using known keywords would instead result in the following array, where the keys that are found shuffled with a value are pushed down until it is an array consisting of pure keys and pure values, as in Table 3.5.

Table 3.4: An array of strings where the keys and their values are mixed up.

<table>
<thead>
<tr>
<th>Total em dívida</th>
<th>66 509 € do qual, em incumprimento</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 €</td>
</tr>
<tr>
<td>Vencido</td>
<td>0 €</td>
</tr>
<tr>
<td>Abatido ao ativo</td>
<td>0 €</td>
</tr>
<tr>
<td>Potencial</td>
<td>0 €</td>
</tr>
<tr>
<td>Prestação</td>
<td>138 € Entrada incumpr.</td>
</tr>
<tr>
<td>Não Aplicável Periodicidade</td>
<td>Mensal</td>
</tr>
</tbody>
</table>

Mensal
Table 3.5: Array of strings after key-value separation has been performed.

**Dealing With OCR Engine Shortcomings**

During the process of extracting values from the OCR processed document, certain flaws with the OCR engines must be remedied or dealt with in order to limit their impact on the total achieved accuracy. Some flaws can be remedied, such as easily recognizable misdetected numbers, while some others are impossible to do anything about—such as values not being recognized at all by the OCR engines. The method for detecting and correcting OCR misdetactions has been presented previously, but there also exists situations where the value of the field is not received at all, or very rarely. This is an issue that is exclusive to Google Cloud Vision. The situation where this occurs is when Google Cloud Vision tries to detect certain values that, in all of the present test cases, are single-number values. There are two fields where this occurs; *Numero*, which is one of the fields belonging to every guarantee, and *No devedores no contrato*, which is one of the fields in the Resulting section. Across the document suite, it is clear that *Numero* cannot be found in any of the present documents, and occurrences of *No devedores no contrato* are extremely rare.
3.4.3 Producing Output

The format of the output has been specified together with the principal, and the choice fell on producing a JavaScript object that can be exported as a JSON file. The structure of the object follows the structure of the PDF documents—a JSON represents a complete extraction of a PDF file with any number of pages. The structure of the JSON file can be seen in Section A.1 of the appendix, taking Figure 1.1 on page 2 as the example file.

3.5 Testing and Test Suite

The testing of the extraction system was done by comparing the extractions of a set of documents \( n = 115 \) against the expected extractions of the same documents. In this section, the construction of this test suite as well as how the testing framework was implemented is explained.

3.5.1 Construction of Test Suite

The test suite was constructed by first running the OCR extraction system that was believed to be the most reliable one over a set of documents \( n = 115 \). This was believed, at the time of the construction of the test suite, to be Tesseract. Since no test suite existed at this time, a few sample extractions were looked at comparing the two OCR engines’ extraction capabilities manually. Since the test suite was corrected afterwards, the choice of OCR engine on which to create the first preliminary reference extractions did not have an impact of the final result. The Tesseract Extraction System was then used to process all the 115 documents, and output was produced in JSON format.

The test suite was constructed by manually correcting these 115 extractions. This was done by the author, as well as some coworkers at the principal’s office. With large objects and a number of levels of nesting, correcting the JSONs explicitly was deemed too difficult and time-consuming. To produce the output in an easily human-readable format, XLS was instead chosen. XLS is a format for spreadsheet files which can be opened using Microsoft Excel for example [46]. The way the data was structured, using the JSON output from Section 3.4.3 as reference, can be seen in Table 3.6. The third column was not filled in using any data from the JSON, but rather filled in by the author and his coworkers during the manual correction of the extractions. Once all the 115 XLS files had been used to mark incorrect extractions, the incorrect ex-
Table 3.6: Fields, values and a boolean denoting whether the value on the same row is correct as seen in the XLS files.

Potential incorrect corrections (or missing corrections), which are inevitable, since the human factor has been introduced into the corrections, were remedied later on while running the other extraction system (Google Cloud Vision) on the same set of corrections. This is because the likeliness that both the solutions would produce the same faulty output is not very high. As discussed previously, the inherent shortcomings of both of the engines are a bit different.

3.5.2 Implementation of Testing Framework

The framework that tests the extractions automatically uses this reference test suite with 115 manually corrected extractions. The way this testing framework is implemented is as follows; all the 115 documents are run through one of the extraction systems one by one. After the data is extracted and a JSON has been produced, a direct comparison between the JSON object output from the extraction system and the reference object is performed. The correctness is then calculated as follows, considering the two objects $O_r$ and $O_e$. $O_e$ corresponds to the object output by the extraction system (extraction object) and $O_r$ to the reference object. The objects can be seen in Figure 3.7. Taking these two objects into consideration, the accuracy is then calculated as a percentage, where 100% entails a perfect extraction. Every data point that is then included in this calculation are the leaf fields. A leaf field in this context could be considered as a field with a primitive data type as a value, and thus with no nested fields inside it. Looking at the reference object for example—a, b, d, e and f are leaf fields. c is not a leaf field, since its value is an object (a non-primitive data type), and it has nested fields inside it. The percentage is then calculated as

$$\sum_{f \in F_r} E(f, O_r, O_e) \times 100$$
Or = { // Reference object
  a: True
  b: False
  c: {
    d: True
    e: True
    f: False
  }
}

Oe = { // Object from extraction system
  a: True
  b: True
  c: {
    d: True
    f: True
  }
}

Figure 3.7: An extraction object and a reference object to be compared.
where

\[ E(f, O_r, O_e) = \begin{cases} 1, & \text{if } O_r(f) = O_e(f) \\ 0, & \text{otherwise} \end{cases} \]

and \( F_r \) is the set of leaf fields in the reference document. Plugging the two example objects into the above equation, the total percentage is \((1 + 0 + 1 + 0 + 0)/5) \times 100 = 40\%\). Over-extraction, i.e., extractions where there are more fields in the extraction object than in the reference object, are not taken into consideration, since this has not been encountered during the entirety of the testing process. Had this been encountered, precision of extractions would also be included in the accuracy percentage calculation. Under-extraction, i.e., where there are fields that are not present in the extraction object, are taken care of using the above formula as the value for a field that an object does not have is \textit{null} in JavaScript.

### 3.6 Evaluation of Design Choices

Making use of the test suite mentioned previously, a direct evaluation of design choices made throughout the process is easily done. By running the 115 documents against their reference counterparts, it is plain to see if the total accuracy percentage has increased or decreased, which serves as an evaluation of an implemented design choice. It is also possible to look at more specific details regarding the effectiveness of implemented changes, such as to list the documents affected by the implementation(s), and much more. The total accuracy can be seen as the average accuracy over the whole range of documents, where each one is weighed equally. To produce results for each extraction mode implemented in order to evaluate its effectiveness, four evaluations are performed, one for each extraction mode. The extraction modes can be seen in Figure 3.8.

### 3.7 Hypothesis Testing

Using the results from the holistic system, the hypothesis was tested. As stated in Section 1.3, the main hypothesis of the project was that the information extraction system built on top of Google Cloud Vision would perform better than the one built on Tesseract.

Firstly, a Shapiro-Wilk test is used to assert whether the data is normally distributed or not [47]. This is needed since for the t-test, as t-tests are used for
1. Bare-bones extraction mode.

2. Type system on top of bare-bones extraction mode.

3. OCR correctional procedures on top of bare-bones extraction mode.

4. Type system and OCR correctional procedures combined.

Figure 3.8: The four extraction modes of the extraction system, which all can be evaluated separately.

normally distributed test data. Secondly, a one-tailed t-test is used in order to assert whether the hypothesis should be accepted or rejected [48]. For both of the tests, a 95% confidence level will be applied.

3.8 Containerization with Docker

Containerization is, as mentioned previously, handled using Docker. Porting the solution to Docker, as opposed to just running it locally, is a straightforward task. In order to port the solution to Docker, a dockerfile is included in the root of the project repository, stating all the needed dependencies of the project and their respective versions. As the project is completely script-based, no mapping between ports was needed, and to run the project using Docker instead of running it locally, is as simple as building the project with Docker and then running it. In the Dockerfile it is stated that the command to be run when running the container image simply was to execute the main source file of the project (index.js) with Node. Making the project runnable with Docker lets stakeholders of the project test the system for themselves in order to ascertain its accuracy, without a risk of local configurations interfering with test results.

3.9 Summary

The programming language used for implementing the extraction system is Node.js, because of its simplicity and its Google Cloud Vision support. The Google Cloud Vision API can be reached through the Node.js client library. For Tesseract, a shell command is executed that makes use of Tesseract as a command line tool (as opposed to its C++ library). Both tools produce output in a plain text format, either as a long string in a JavaScript object (Google
Cloud Vision) or as a TXT file (Tesseract). This output is then pre-processed where correction of potential misdetections could be performed. For the actual extraction, an ad-hoc type system is implemented and used in order to further increase the accuracy of the solution by making use of the expected values for each field. The output of the extraction process is structured in a JavaScript object, and written to the local file system as a JSON file. The testing is performed comparing two local JSON files—one reference file and one file representing the extracted information from the same document. The test suite consists of 115 documents and was used for evaluating design choices made throughout the project, as well as for hypothesis testing for the final solution.
Chapter 4

Results

In this chapter, results from the different experiments made using the extraction system will be presented. The results are divided by the type of experiment, or the method for extraction.

4.1 Bare-bones Rule-based Extraction

The bare-bones rule-based extraction methodology is described in Section 3.4.2. Here, the results for that extraction mode are presented. A range of metrics regarding the two OCR engines’ accuracies can be seen in Table 4.1. Here, it can be seen that Tesseract outperforms Google Cloud Vision for every category of the comparison. A visualisation of the distribution of accuracies achieved across the test suite range for Tesseract can be seen in Figure 4.1. The Google Cloud Vision counterpart can be seen in Figure 4.2. It can be seen from the distributions that the distribution for Tesseract is more clearly skewed to the right, and that the distribution of accuracies is more normally distributed for Google Cloud Vision than for Tesseract.

4.2 Introduction of Type System

The extraction methodology that makes use of the type system is described in Section 3.4.2. Here, the results for that extraction mode are presented. A range of metrics regarding the two OCR engines’ accuracies can be seen in Table 4.2. Here, it can be seen that Tesseract outperforms Google Cloud Vision for every category of the comparison. A visualisation of the distribution of accuracies achieved across the test suite range for Tesseract can be seen in Figure 4.3. The Google Cloud Vision counterpart can be seen in Figure 4.4. Here it can
CHAPTER 4. RESULTS

Figure 4.1: The distribution of achieved accuracies for documents for the bare-bones extraction method for Tesseract. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where $75 < \text{accuracy} \leq 80$.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Tesseract</th>
<th>GCV</th>
<th>PPD</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Accuracy</td>
<td>91.24%</td>
<td>83.21%</td>
<td>8.03%</td>
<td>9.65%</td>
</tr>
<tr>
<td>Median Accuracy</td>
<td>91.72%</td>
<td>83.05%</td>
<td>8.67%</td>
<td>10.44%</td>
</tr>
<tr>
<td>Maximum Accuracy</td>
<td>100.00%</td>
<td>94.74%</td>
<td>5.26%</td>
<td>5.55%</td>
</tr>
<tr>
<td>Minimum Accuracy</td>
<td>72.73%</td>
<td>52.54%</td>
<td>20.19%</td>
<td>38.43%</td>
</tr>
<tr>
<td>Documents Passing 90% Threshold</td>
<td>66.09%</td>
<td>11.30%</td>
<td>54.79%</td>
<td>484.62%</td>
</tr>
<tr>
<td>Total Correct Fields</td>
<td>91.46%</td>
<td>82.06%</td>
<td>9.40%</td>
<td>11.46%</td>
</tr>
</tbody>
</table>

Table 4.1: A selection of accuracy metrics for the bare-bones mode of the extraction system for the two OCR engines. PPD = percentage-point difference, RD = relative difference.
be seen that the number of documents that lie in the top two accuracy ranges differs greatly between the two OCR engines.

### 4.3 Attempted Correction of OCR Engine Mis-detections

The steps taken to attempt to correct potential misdetections made by the OCR engines are listed specifically in Section 3.4.2. Here, results obtained by implementing these steps on top of the bare-bones extraction system are presented. A range of metrics regarding the two OCR engines’ accuracies can be seen in Table 4.3. Here, it can be seen that Tesseract outperforms Google Cloud Vision for every category of the comparison, and that Google Cloud Vision’s performance is equal to its performance using the bare-bones extraction system. A visualisation of the distribution of accuracies achieved across the test suite range for Tesseract can be seen in Figure 4.5. The Google Cloud Vision counterpart can be seen in Figure 4.6. It can be seen that the number of documents in the top interval for Tesseract has increased comparing to the bare-bones extraction system.
<table>
<thead>
<tr>
<th>Metric</th>
<th>Tesseract</th>
<th>GCV</th>
<th>PPD</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Accuracy</td>
<td>93.99%</td>
<td>89.79%</td>
<td>4.20%</td>
<td>4.68%</td>
</tr>
<tr>
<td>Median Accuracy</td>
<td>94.74%</td>
<td>90.11%</td>
<td>4.63%</td>
<td>5.14%</td>
</tr>
<tr>
<td>Maximum Accuracy</td>
<td>100.00%</td>
<td>97.06%</td>
<td>2.94%</td>
<td>3.03%</td>
</tr>
<tr>
<td>Minimum Accuracy</td>
<td>77.27%</td>
<td>71.19%</td>
<td>6.08%</td>
<td>8.54%</td>
</tr>
<tr>
<td>Documents Passing 90% Threshold</td>
<td>85.22%</td>
<td>50.43%</td>
<td>54.79%</td>
<td>68.97%</td>
</tr>
<tr>
<td>Total Correct Fields</td>
<td>94.63%</td>
<td>88.90%</td>
<td>5.73%</td>
<td>6.45%</td>
</tr>
</tbody>
</table>

Table 4.2: A selection of accuracy metrics for the extraction mode making use of the type system for the two OCR engines. PPD = percentage-point difference, RD = relative difference.

Figure 4.3: The distribution of achieved accuracies for documents for the extraction method using the type system for Tesseract. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where $75 < \text{accuracy} \leq 80$. 
Figure 4.4: The distribution of achieved accuracies for documents for the extraction method using the type system for Google Cloud Vision. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where $75 < \text{accuracy} \leq 80$.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Tesseract</th>
<th>GCV</th>
<th>PPD</th>
<th>RD</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Accuracy</td>
<td>93.20%</td>
<td>83.21%</td>
<td>9.99%</td>
<td>12.01%</td>
</tr>
<tr>
<td>Median Accuracy</td>
<td>94.44%</td>
<td>83.05%</td>
<td>11.39%</td>
<td>13.71%</td>
</tr>
<tr>
<td>Maximum Accuracy</td>
<td>100.00%</td>
<td>94.74%</td>
<td>5.26%</td>
<td>5.55%</td>
</tr>
<tr>
<td>Minimum Accuracy</td>
<td>72.73%</td>
<td>52.54%</td>
<td>20.19%</td>
<td>38.43%</td>
</tr>
<tr>
<td>Documents Passing 90% Threshold</td>
<td>70.43%</td>
<td>11.30%</td>
<td>54.79%</td>
<td>523.08%</td>
</tr>
<tr>
<td>Total Correct Fields</td>
<td>93.20%</td>
<td>82.06%</td>
<td>11.14%</td>
<td>13.58%</td>
</tr>
</tbody>
</table>

Table 4.3: A selection of accuracy metrics for the extraction mode making use of the OCR correctional measures for the two OCR engines. PPD = percentage-point difference, RD = relative difference.
Figure 4.5: The distribution of achieved accuracies for documents for the extraction method using OCR correction for Tesseract. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where $75 < \text{accuracy} \leq 80$.

Figure 4.6: The distribution of achieved accuracies for documents for the extraction method using OCR correction for Google Cloud Vision. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where $75 < \text{accuracy} \leq 80$. 
Table 4.4: A selection of accuracy metrics for the holistic extraction mode for the two OCR engines. PPD = percentage-point difference, RD = relative difference.

4.4 Final Holistic Solution

All the different steps toward producing an optimal extraction engine are described in Section 3.4.2 in detail. In this section, results of combining all these different methods into a holistic system are presented. A range of metrics regarding the two OCR engines’ accuracies can be seen in Table 4.4. Here, it can be seen that Tesseract outperforms Google Cloud Vision for every category of the comparison, and that Google Cloud Vision’s performance is equal to its performance using only the type system. A visualisation of the distribution of accuracies achieved across the test suite range for Tesseract can be seen in Figure 4.7. The Google Cloud Vision counterpart can be seen in Figure 4.8. It can be seen that Tesseract has the greatest portion of documents in the top accuracy range for this extraction mode.

4.5 Accuracy by Field

In this section, the results of the different OCR engines and extraction modes with regards to specific data fields will be presented. The section aims to highlight which fields the extraction systems performed better at and which fields the systems performed worse at. The results are divided by OCR engine.

4.5.1 Tesseract OCR

Here, the results of the extraction modes that made use of the Tesseract OCR engine will be presented. The results are divided based on extraction mode in order to show how the accuracy fluctuates between the different solutions,
Figure 4.7: The distribution of achieved accuracies for documents for the holistic extraction method for Tesseract. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where \( 75 < \text{accuracy} \leq 80 \).

Figure 4.8: The distribution of achieved accuracies for documents for the holistic extraction method for Google Cloud Vision. Each interval represents the number of extractions that fell into that interval; 75-80 denotes the number of extractions where \( 75 < \text{accuracy} \leq 80 \).
Table 4.5: Accuracy based on field for each of the extraction modes using Tesseract OCR. BB = bare-bones, TS = type system, OCRC = OCR correction and TF = total number of fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>BB</th>
<th>TS</th>
<th>OCRC</th>
<th>Holistic</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nome</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>115</td>
</tr>
<tr>
<td>Tipo de Identificacao</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>115</td>
</tr>
<tr>
<td>No de Identificacao</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>115</td>
</tr>
<tr>
<td>Bank</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>257</td>
</tr>
<tr>
<td>Tipo de responsabilidade</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>115</td>
</tr>
<tr>
<td>Produto financeiro</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Tipo de negociacao</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Inicio</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>No devedores no contrato</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Em litigio judicial</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Fim</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Teto</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Valor</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Numero</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Total em dívida</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>do qual, em incumprimento</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Vencido</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Abatido ao ativo</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Potencial</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Prestacio</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Entrada incumpr.</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
<tr>
<td>Periodicidade</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>96.52%</td>
<td>426</td>
</tr>
</tbody>
</table>

and where the difference is most noticeable. The results in their entirety can be seen in Table 4.5.

### 4.5.2 Google Cloud Vision

Here, the results of the extraction modes that made use of Google Cloud Vision will be presented. The results are divided based on extraction mode in order to show how the accuracy fluctuates between the different solutions, and where the difference is most noticeable. The results in their entirety can be seen in Table 4.6.
<table>
<thead>
<tr>
<th>Field</th>
<th>BB</th>
<th>TS</th>
<th>OCRC</th>
<th>Holistic</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nome</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>115</td>
</tr>
<tr>
<td>Tipo de Identificação</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>115</td>
</tr>
<tr>
<td>No de Identificação</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>115</td>
</tr>
<tr>
<td>Bank</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>100.00%</td>
<td>257</td>
</tr>
<tr>
<td>Tipo de responsibilidade</td>
<td>96.01%</td>
<td>96.01%</td>
<td>96.01%</td>
<td>96.01%</td>
<td>426</td>
</tr>
<tr>
<td>Produto financeiro</td>
<td>95.31%</td>
<td>95.31%</td>
<td>95.31%</td>
<td>95.31%</td>
<td>426</td>
</tr>
<tr>
<td>Tipo de negociação</td>
<td>89.44%</td>
<td>89.44%</td>
<td>89.44%</td>
<td>89.44%</td>
<td>426</td>
</tr>
<tr>
<td>Início</td>
<td>57.28%</td>
<td>99.53%</td>
<td>57.28%</td>
<td>99.53%</td>
<td>426</td>
</tr>
<tr>
<td>No devedores no contrato</td>
<td>5.16%</td>
<td>7.51%</td>
<td>5.16%</td>
<td>7.51%</td>
<td>426</td>
</tr>
<tr>
<td>Em litigio judicial</td>
<td>54.69%</td>
<td>86.38%</td>
<td>54.69%</td>
<td>86.38%</td>
<td>426</td>
</tr>
<tr>
<td>Fim</td>
<td>89.44%</td>
<td>100.00%</td>
<td>89.44%</td>
<td>100.00%</td>
<td>426</td>
</tr>
<tr>
<td>Valor</td>
<td>83.16%</td>
<td>96.43%</td>
<td>83.16%</td>
<td>96.43%</td>
<td>196</td>
</tr>
<tr>
<td>Total em divida</td>
<td>98.36%</td>
<td>98.36%</td>
<td>98.36%</td>
<td>98.36%</td>
<td>426</td>
</tr>
<tr>
<td>do qual, em incumprimento</td>
<td>93.43%</td>
<td>99.77%</td>
<td>93.43%</td>
<td>99.77%</td>
<td>426</td>
</tr>
<tr>
<td>Vencido</td>
<td>99.06%</td>
<td>100.00%</td>
<td>99.06%</td>
<td>100.00%</td>
<td>426</td>
</tr>
<tr>
<td>Abatido ao ativo</td>
<td>94.37%</td>
<td>97.18%</td>
<td>94.37%</td>
<td>97.18%</td>
<td>426</td>
</tr>
<tr>
<td>Potencial</td>
<td>94.13%</td>
<td>94.84%</td>
<td>94.13%</td>
<td>94.84%</td>
<td>426</td>
</tr>
<tr>
<td>Prestação</td>
<td>90.14%</td>
<td>92.96%</td>
<td>90.14%</td>
<td>92.96%</td>
<td>426</td>
</tr>
<tr>
<td>Entrada incumpr.</td>
<td>92.96%</td>
<td>99.53%</td>
<td>92.96%</td>
<td>99.53%</td>
<td>426</td>
</tr>
<tr>
<td>Periodicidade</td>
<td>92.96%</td>
<td>95.54%</td>
<td>92.96%</td>
<td>95.54%</td>
<td>426</td>
</tr>
</tbody>
</table>

Table 4.6: Accuracy based on field for each of the extraction modes using Google Cloud Vision. BB = bare-bones, TS = type system, OCRC = OCR correction and TF = total number of fields.
4.6 Accuracy by Data Type

In this section, the results of the different OCR engines and extraction modes with regards to the type of the fields are presented. The section aims to highlight which data types the extraction systems performed better at and which types the systems performed worse at. The results are divided by OCR engine.

4.6.1 Tesseract OCR

Here, the results of the extraction modes that made use of the Tesseract OCR engine will be presented. The results are divided based on extraction mode in order to show how the accuracy fluctuates between the different solutions, and where the difference is most noticeable. The results in their entirety can be seen in Table 4.7.

4.6.2 Google Cloud Vision

Here, the results of the extraction modes that made use of Google Cloud Vision will be presented. The results are divided based on extraction mode in order to show how the accuracy fluctuates between the different solutions, and where the difference is most noticeable. The results in their entirety can be seen in Table 4.8.

4.7 Hypothesis Test

The data from the Tesseract implementation could be confirmed to be normally distributed with a confidence level of 95\% ($p = 3.36 \times 10^{-10}$) using a
Table 4.8: Accuracy based on type for each of the extraction modes using Google Cloud Vision. BB = bare-bones, TS = type system, OCRC = OCR correction and TF = total number of fields.

<table>
<thead>
<tr>
<th>Field</th>
<th>BB</th>
<th>TS</th>
<th>OCRC</th>
<th>holistic</th>
<th>TF</th>
</tr>
</thead>
<tbody>
<tr>
<td>TYPE_DENOMINATION</td>
<td>93.97%</td>
<td>97.02%</td>
<td>93.97%</td>
<td>97.02%</td>
<td>2752</td>
</tr>
<tr>
<td>TYPE_NUM</td>
<td>18.59%</td>
<td>19.95%</td>
<td>18.59%</td>
<td>19.95%</td>
<td>737</td>
</tr>
<tr>
<td>TYPE_ID</td>
<td>83.16%</td>
<td>96.43%</td>
<td>83.16%</td>
<td>96.43%</td>
<td>196</td>
</tr>
<tr>
<td>TYPE_DATE</td>
<td>73.36%</td>
<td>99.77%</td>
<td>73.36%</td>
<td>99.77%</td>
<td>852</td>
</tr>
<tr>
<td>TYPE_TEXT</td>
<td>88.99%</td>
<td>94.71%</td>
<td>88.99%</td>
<td>94.71%</td>
<td>3043</td>
</tr>
</tbody>
</table>

Shapiro-Wilk test. Similarly, the accuracy data from the Google Cloud Vision extractions could also be confirmed to come from a normal distribution \((p = 1.36 \times 10^{-7})\) using the same test. The hypothesis that the extraction system using Google Cloud Vision would perform more accurately than the one using Tesseract can be rejected \((p = 1.00000)\) using a one-tailed t-test. The average and median accuracies across the test suite for Tesseract are, as presented earlier, 96.16% and 97.18%. For Google Cloud Vision, the average accuracy is 89.79% and the median is 90.11%. The standard deviation of the accuracies for the two engines are 4.32 and 4.47 for Tesseract and Google Cloud Vision, respectively.

### 4.8 Summary

For all the extraction modes, Tesseract outperformed Google Cloud Vision. For both OCR engines, the holistic solution was (as expected) the one that performed the best. For Tesseract, both the type system and the OCR correctional version improved the accuracy of the solution, whereas for Google Cloud Vision only the type system managed to improve its accuracy. Some specific fields were extracted more accurately for Google Cloud Vision, but generally Tesseract outperformed Google Cloud Vision in every category. The hypothesis that Google Cloud Vision would outperform Tesseract could thus be rejected.
Chapter 5

Discussion

In this chapter, the information extraction system is discussed in detail with regards to the achieved results. OCR Engines and their feasibility as the backbone of information extraction systems are also discussed in detail based on the experience of implementing and testing the information extraction system and the two OCR engines.

5.1 Design Decisions

This section aims to shed some additional light on the results found when testing the different extraction modes of the system, and also to discuss these results in a general sense. The section is divided based on the particular design choices made, with one subsection discussing the final holistic solution that makes use of both the type system and the OCR correctional procedures.

5.1.1 Divide-and-conquer Methodology

A divide-and-conquer methodology was chosen for building the extraction system. This allowed for development to be performed in a modular manner as opposed to a monolithic design process. Additionally, I/O functions were used exclusively instead of functions that mutated data. These two design decisions made de-bugging of the project during the development stage easier. One problem of using a completely rule-based approach, however, is the situation that arose where two sections got mixed in with each other. Simply using a cutoff point in those situations caused some problems. One example of this was for the value of the Tipo de responsabilidade field, which in numerous cases got extracted before the actual key (which is used as the cutoff point).
This is a common cause of incorrect extractions.

5.1.2 Bare-bones Rule-based Extraction

The bare-bones extraction mode was designed for two reasons: test the robustness of the type system as well as the OCR correctional methods. It was also useful for evaluating the output predictability of Google Cloud Vision and Tesseract. This extraction mode, as discussed previously, only deals with indices and keywords and does not evaluate values to be extracted explicitly. Because of this, irregularities in the extraction from the OCR engines will be impossible to predict and remedy, and this is something that impacts the achieved accuracy as well. Additionally, the possibility of correcting possible misdetections made by the OCR engines is not possible in this extraction mode either, which means that there will be problems with zeros getting recognized as O’s and other similar issues.

Looking at the achieved results for the two OCR engines from the bare-bones extraction mode, it can be seen that the lowest accuracy for both solutions was achieved here—as expected. The accuracy when using Tesseract is better for every metric—the average accuracy is higher, the median accuracy is higher and the maximum and minimum accuracies are both higher. Since no OCR correctional methods or type system are used here, it can be deduced that the structure in which Tesseract outputs its results is more standardized between documents and easier to predict.

The metric that most clearly displays the big difference between the two OCR engines when used together with the bare-bones extraction system is the number of documents passing the 90% threshold. For Tesseract, 76 documents passed the threshold while only 13 of the Google Cloud Vision extractions did. It is also clear from the figures of the distributions that the greatest chunk of extractions for the Google Cloud Vision version lies in the 80–85% interval, which is slightly under the threshold, whereas the most frequent interval for the Tesseract solution is 90–95%, which is just above the principal’s threshold.

When looking at the accuracy for specific fields of the solution, there are a few fields where the performance is severely lacking when looking at the results for Tesseract. These are mainly the Guarantee fields (Tipo, Valor and Numero). It seems that the order of these values often come shuffled, and without type-checking these values, the accuracy is low. This is a good ex-
ample of where a type-checker may improve the accuracy significantly for an information extraction system based on OCR engines that may produce values that are out of order. When looking at the values for Google Cloud Vision, it can be seen that there are several fields that reduce the total accuracy severely. The *Numero* field, as discussed previously, is totally undetectable when it comes to Google Cloud Vision. Additionally, the value for *No devedores no contrato* is extremely rare even though it is encountered occasionally. One explanation for this could be that these one-character words get removed by the noise removal procedure of Google Cloud Vision mentioned in Section 2.2.1. It can be shown that there are some fields that get extracted more accurately than their Tesseract counterparts for the bare-bones version; these are generally fields that are misrecognized by the Tesseract. That Google Cloud Vision performs better when it comes to pure OCR-capabilities (disregarding structure) has been previously shown [33, 35]. An example of this is *No de Identificacao*. This field generally consists of nine numbers, and as been discussed before Tesseract have problems recognizing certain symbols. These shortcomings are then magnified for large numbers that may contain several symbols that are hard to recognize. For the first four fields, which are easily separable, Google Cloud Vision also achieves a 100% accuracy, which is another testament to the pure OCR capabilities of the OCR engine.

When looking more generally at the type of the data, it is clear from the results that Google Cloud Vision’s overall accuracy suffers from its poor performance at recognizing number types. However, Google Cloud Vision outperforms Tesseract for two of the five data types; TYPE_DENOMINATION and TYPE_ID.

### 5.1.3 Type System

On top of the bare-bones extraction methodology, the type system was implemented. The type system was implemented after a first review of the document suite, where it was clear that the data fields were often somewhat predictable—at least when it came to the type of value that they correlated with. This is a fact that can be used to greatly increase accuracy when it comes to situations where fields of different types are scattered throughout the same section. For this particular document, this is very useful, since all the modular sections of the document contain values of at least two different types. We showed in Section 3.4.2 that the structure in which the output is formatted from both the OCR engines still have a lot to wish for, so a solution that can remedy this to
any extent will increase the total accuracy of the solution.

The use of the type system without involving any OCR correctional measures, however, may result in incorrect extractions, where one value might get classified as being of another type than what it actually is. For example, when it comes to numbers, the string "200/" would get classified as a text value even though it probably is meant as "2007" but misrecognized by Tesseract. When looking at exact accuracy, however, as in the fact that two strings must be exactly identical to be deemed correct, this does not affect the final result. This is due to the fact that even if "200/" would get classified as a number, and put into the right place, it would still be deemed incorrect by the testing framework, since "200/" != "2007". Additionally, the type system still cannot remedy the situation in which two values of the same type get mixed up—in the resulting section for example, the four first values are all text values—if these get mixed up by either of the OCR engines, this will not be able to be corrected by the extraction system.

When it comes to the actual results from testing of the extraction modes supported by the type system for the two engines, it is obvious that Tesseract outperforms Google Cloud Vision here as well. For every category of the comparison in Section 4.2, Tesseract comes out as the stronger alternative. Looking at the number of documents passing threshold, the type system proved to provide a big improvement for the Google Cloud Vision implementation. Still, the number of documents that are extracted sufficiently accurately for the principal in a practical application are few in relation to the Tesseract implementation. It can be seen from the distributions that a lot of the extractions from the lower
ranges (≤ 85) have migrated into the two top intervals for Tesseract (90–95% and 95–100%). For Google Cloud Vision, the most frequent interval has gone from 80–85% to 90–95%.

Looking at specific fields, a clear jump in accuracy for the guarantee fields (Tipo, Valor and Numero) can be seen for the Tesseract solution. It seems that these values often come shuffled, but, since they are all of different types, this can be remedied to a great extent by the type system. One interesting point about the reduction in accuracy between the bare-bones solution and the solution that makes use of the type system for the Tesseract engine was discovered. Albeit minor, for the fields Potencial, Prestacao and Periodicidade, a worse accuracy using the type system is achieved compared to the one without it. The drawback of the type system, as mentioned previously, is that it is entirely rule-based, offering no flexibility when it comes to certain misrecognitions. An example of this was found in the extraction of an amounts section, where the values got recognized as in Figure 5.1. The problem with this extraction is that the value that got extracted as ‘0’ actually was meant to be ‘0 €’. What happened then was that, during the type-checking, the ‘0’ (rightfully so) got typed as a TYPE_NUM, and thus not held on to, since there are not supposed to be any TYPE_NUM values in that particular section. This resulted in the array used for keeping TYPE_DENOMINATION values containing five elements (instead of the usual six). Additionally, the fourth and fifth element got pushed up one index. This resulted in two extraction errors, one for the fifth key (0 -> 419) and one for the sixth key (419 -> “n/a”). With the bare-bones extraction system, this would only result in one error (0 € -> 0). It can also be seen that this problem was not encountered with Google Cloud Vision, due to the fact that misrecognitions of that nature are much more rare.

Looking at the type system implementation more generally with regards to types, it can be shown that for Tesseract the introduction of types increased accuracy for all types but one, TYPE_TEXT. As previously mentioned, due to misdetections, the type system may actually end up worsening accuracy in some special cases—but the overall benefits of the system outweighed the negatives, as shown by the increase in overall accuracy between the solutions. For Google Cloud Vision, the accuracy was increased between the solutions for all data types.
5.1.4 OCR Engine Correction

The OCR correctional methods were developed after the initial testing of the Tesseract version of the extraction system. Some of the misdetections done by Tesseract can, if sufficiently predictable, be remedied by the extraction system. This can be seen as an additional post-processing step for the holistic solution, where some known misdetections can be used together with background knowledge of the domain to correct these shortcomings. Misdetections between characters of the same origin (such as numbers -> numbers, characters -> characters) are by nature hard to remedy, since it is hard to know what the original value was. However, when misdetections are done turning a number into a symbol or a character, background knowledge of the document can be used in order to correct this. This is described in depth in Section 3.4.2.

Obviously, shortcomings of the OCR correctional methods do exist and with Tesseract there will always be misdetections made that cannot be remedied during the extraction process. However, the approach chosen during the development of the OCR correctional methods was to only correct potential misdetections in situations where it is completely certain that there has been a misdetection. This defensive methodology was chosen over a more intrusive one, where OCR corrections are done even if certainty of misdetection cannot be guaranteed. An example of this could be that the value for No devedores no contrato are, across the whole test suite of 115 documents, always either 1 or 2. Using this fact, and the fact that the number one in some cases get recognized as the number seven, one action could be to always turn a seven into a one. Even though this may increase the total accuracy, choices like this were turned down in order to obtain a system that, in theory, could handle every potential (correct) value.

When it comes to the achieved results of the extraction mode, there is no need to look at the results for Google Cloud Vision. This is due to the fact that they are exactly identical to the ones for the bare-bones extraction mode. The reason as to why they are completely identical is that the OCR correctional methods were developed for Tesseract in the first place, and none of the misdetections experienced with Tesseract were encountered at all using Google Cloud Vision. For Tesseract, however, it can be seen that for all the general metrics, an increase in accuracy was achieved between the OCR correcting system and the bare-bones extraction system. The distribution of achieved accuracies has shifted to the right when compared to the bare-bones extraction
• Value 2
• 100€

Figure 5.2: Two shuffled values to be processed by the extraction system.

mode, and for the OCR correctional extraction mode, the most frequent interval is now the top interval. As for specific fields, it can be shown that the accuracy for all fields was either the same as in the bare-bones implementation or greater. The bulk of the increased accuracy appeared in the amounts section, where all the denomination type values experienced a dramatic increase in accuracy. Looking at Table 4.7, it can also be seen that the greatest increase in accuracy was encountered for the denomination type, both relatively and percentage-point-wise.

5.1.5 Holistic Solution

When looking at the results for the holistic solution, it can be examined whether the holistic solution successfully can incorporate the best elements of every extraction mode or falls with the worst one. As discussed previously, the holistic solution simply consists of combining the OCR correctional methodology with the implemented type system in order to achieve a total accuracy that surpasses all the others’. Looking at it theoretically, there certainly are situations in which a combination of the two methodologies (at least for the Tesseract implementation) can solve problems that neither of the two methodologies can solve separately. As an example, consider the values in Figure 5.2, where their order is shuffled. As pre-processing of all the lines is performed, 1OO € would get corrected to 100 €, placed on the same index, and then, when sorting by type, the values would get extracted correctly.

The major shortcomings of either of these two methodologies still exist in the holistic solution, thus expecting a 100% correct extraction using a combination of these two is unrealistic. Some situations can be remedied, such as correctly typing values that suffered minor misdetections by the OCR engines, but major misdetections and formatting inconsistencies between values of the same type are still unrepairable.

Looking at the accuracy in general terms, it can be seen that the average as well as the median accuracy across the test suite are at their highest point point
using the holistic system for both Tesseract and Google Cloud Vision. This is also true for the number of documents passing the 90% threshold as well as the percentage of fields across the whole suite correctly extracted. These are successful results, since an unwanted effect of the two methodologies would be that they somehow contradict or negate each other’s positive effects.

As for the accuracy distribution for the Tesseract OCR engine, the greatest bulk of extractions lie in the 90–95% interval by a large margin. Additionally, accuracies below the threshold have been reduced immensely compared to the bare-bones extraction method, which is a testament to its success.

When it comes to the accuracy for specific fields, the results are similar. For Tesseract, there are three fields’ accuracy (Potencial, Prestacao and Periodicidade) that are not at their peaks for the holistic extraction mode. This is due to the fact also mentioned in Section 5.1.3, that poorly extracted values can lead to a reduction of accuracy by the use of the type system for Tesseract. Using the OCR correctional methods does not help in this scenario, since there are no OCR correctional methods present for turning numeric values into denomination values. Other than that, the accuracy for the rest of the values is higher than or equal to their type system or OCR correctional counterparts. As can be seen in Table 4.5, there are only five fields’ accuracies that are strictly higher than any of the other solutions. As for fields that are equal to the most accurate one of the three previous extraction modes, there are 14 of them. It can be deduced that for Tesseract, the holistic solution seems to bring out the best of each of the extraction modes but does not really succeed in bringing them together into a system that is greater than the sum of its parts. It seems that examples like the one mentioned previously in this subsection are rare, and that (solvable) errors seem to either lie in the domain of OCR misdetection or the domain of structural shortcomings, and not both simultaneously. Looking at the results for the accuracy based on type, Table 4.7 shows similar results. One of the types, TYPE_TEXT, achieves a lower total accuracy than for its OCR correctional counterpart, but for the remaining four types their accuracies are all equal to or greater than for any other extraction mode. More precisely, three out of four remaining types are extracted at maximum accuracy by the holistic solution. The accuracy for ID types are identical in the holistic solution to the type system extraction mode.

As mentioned in Section 5.1.4, since the OCR correctional methodology is not improving the results for the Google Cloud Vision implementation, it comes
as no surprise that the results of the holistic solution are identical to the ones of the type system implementation.

5.2 Hypothesis

As for the hypothesis stated in Section 1.3, namely that the Google Cloud Vision version would outperform the Tesseract version, it comes as no surprise, looking at the results, that the hypothesis was rejected. As for the secondary research question; both the average and the median accuracy for the Tesseract solution was over 90%, so it can be concluded that the best one of the two solutions surely is accurate enough to be used in practice by the principal.

5.3 OCR Engines and Their Feasability

This section aims to, as opposed to the previous one, discuss the actual OCR engines instead of the extraction system built on top of them. Different shortcomings and weaknesses with the chosen OCR engines will be discussed both generally and with regards to the test data, and finally the optimal OCR engine for information extraction purposes will be described.

5.3.1 Misdetections and Post-processing

Misdetections are inevitable for any OCR engine. Since misdetections happen, to varying extent, with the use of any OCR processor, steps need to be taken in order to try and remedy or at least detect these potential errors whenever possible. The frequency of misdetections of an OCR engine is generally a good indicator for how suitable that specific tool is for information extraction systems. However, as shown in this thesis, the OCR engine that produces the most accurate detections for a given document does not necessarily have to be the one that is most viable for an information extraction system.

In some cases, it seems that it may be an OCR engines’ post-processing procedures that turn a correct extraction into an incorrect one. This is based on the fact that Tesseract sometimes added apostrophes seemingly at random to customers’ names and bank names that did not originally contain them, as well as turning regular c’s into c-cedilla (ç). This was a minor cause of incorrect extractions for the Tesseract engine, whereas the Google Cloud Vision handled these names without a problem. Again, these misdetections are difficult
to handle in the extraction engine, as there is no way to tell what the original value was.

5.3.2 Optimal Use Cases for Tesseract and Google Cloud Vision

Based on the two OCR engines’ strengths and weaknesses, it is possible to theorize about what kind of documents may fit an information extraction system built on one OCR engine or the other. It has been concluded that Tesseract delivers output in a slightly more predictable format, while at the same time having trouble to recognize text 100% correctly, unless background information about the document can be used. Using this information, it seems like the situation in which Tesseract would outperform Google Cloud Vision the most would be for a document that is fairly complex when it comes to structure of the document, but with easily recognizable text—optimally with the text having a format that is pre-determined to some extent that can be used to correct potential incorrect detections made by Tesseract. When it comes to Google Cloud Vision, for it to achieve at its peak performance the document’s structure needs to be fairly straightforward and not overly cluttered. Optimally this document would only contain data that is read from top-to-bottom, without any sections that are divided horizontally and values that are close to each other along the y-axis (such as Mapa CRC). On top of that, to make Google Cloud Vision outperform Tesseract as much as possible these fields would consist of long text strings with many words, and without any clear format that could be used to correct incorrect detections.

5.3.3 The Optimal OCR Engine

When evaluating the flaws of the two tested OCR engines, it is interesting to theorize about how the optimal OCR engine for information extraction would look. When it comes to correctness of extractions there is not a lot to say—an information extraction system built on top of an OCR engine is, as any other OCR application, heavily dependant on accurate detections. When it comes to the structure of the output, however, it is much more vital for an information extraction application that the OCR engine handles the formats of different documents in a robust and predictable manner. For adaptability to different documents’ formats, the optimal OCR engine would somehow be customizable when it comes to the order in which the text on the image is read. For the sake of the Mapa CRC, the easiest way to handle the format would be to
read it from top-to-bottom and then left-to-right. This would let us handle the only element of the document that is not identical between credits, which is the guarantees section. The guarantees section grows downwards along the y-axis depending on how many guarantees are present for the specific credit, but if the document would be read top-to-bottom, this would not cause any problems, as there are no values for any other field below it. Currently, if the guarantees section is 3 elements or longer (which is rare, but still a possibility), the segmentation of the pages is a lot harder to do if the documents are read left-to-right, top-to-bottom, which is the current order in which they are read.

A really good addition to what was mentioned above, and another feature which would be extremely useful for the Mapa CRC, would be an OCR engine that can understand simpler formatting structures and thus handle some of the segmentation automatically. Taking Figure 1.1 on page 2 as an example, it can be seen that the credits are divided by the use of thin grey borders around them. If an OCR engine itself could handle the subdivision of this document, the information extraction would be easier and problems with values overlapping between sections would be completely non-existent. Looking at Mapa CRC, there are even nested sections. An OCR engine that could correctly nest these sections as well would be extremely useful for information extraction purposes.

5.4 Development of Information Extraction Systems

This section looks at general aspects of developing systems for information extraction. It includes pitfalls encountered and lessons learned during the development process, which may serve as a pointer for software developers in related fields. It is also worth noting that often information extraction systems deal with personal or classified information, and proper care must be taken when it comes to storage and transmission of this information. For this specific project, work was carried out in a closed GitHub repository that was only shared with the co-workers at the principal’s office. A non-disclosure agreement was signed with regards to client data that was used in the development and testing process.
5.4.1 Test Suite

Initially, a complete test suite was not used. This was due to the fact that preliminary, initial extractions had to be made in order to have something to manually correct. However, as was learned during the development process, this should have been completed earlier. A lot of design decisions had been made when only considering two or three initial sample documents, which led to a lot of ad-hoc solutions and a naïve belief that the structure of the OCR output would more or less be identical over the 100+ documents of the final test suite. This proved to be very incorrect, and a lot of problems were found once the test suite had been built that either were handled incorrectly or not at all. Problems not being handled at all is of course not as big of a problem as problems handled in the wrong way when it comes to resources put in to the project, as problems that were not handled at all could just as easily be handled once the test suite had been completed. But when it comes to problems handled in an incorrect way, the system required a substantial amount of rebuilding once it was clear that the output could vary a lot more between documents throughout the suite than initially thought. This came to some surprise as structurally, the documents are very similar, yet the order in which fields were read could vary greatly between documents. During this process, a lot of time would have been saved should the test suite had been constructed previously—however, in this particular case a Catch-22 situation arose. This was because a test suite was not present initially, and for the test suite to be built some initial, preliminary extractions would need to be made that were then to be manually corrected by the author and other personnel at the principal. Because of this, and the fact that the manual correction was scheduled beforehand, a larger portion than needed of the development process had to be done prior to the construction of the test suite. In an optimal world, previous extractions would already exist that could be made use of or a test suite could have been constructed much earlier than for this project.

5.4.2 Containerization with Docker

Containerization was used for this project, and as previously discussed this was done using Docker. Containerization was of help during the development stage since it allowed for coworkers at the principal’s office to evaluate the system continuously and provide feedback. Docker was useful for this since software developers not actively participating in the project could test the solution without having to install the needed dependencies locally. This also reduced the resource necessities for new developers in order to start up and
continue to develop the system. Should the principal chose to deploy the application this could be done in the Docker environment as well as opposed to some server machine where dependencies would need to be installed.

5.5 Summary

For the bare-bones extraction method, there was a big difference between the solutions when it comes to the number of documents passing the 90% threshold. For this method, some fields are more error-prone than others because of their tendency to be shuffled by the OCR engines.

As for the type system, the accuracy for both solutions is improved. Some situations can arise where the accuracy is decreased by making use of the type system—though seen as a whole, the system is successful. It can be noted that the most frequent accuracy interval for Google Cloud Vision has moved from 80–85% (under accuracy threshold) to 90–95% (above accuracy threshold).

For the OCR correctional methods, the results as compared to the bare-bones extraction mode were only improved for Tesseract OCR. For Google Cloud Vision, these measures did not improve accuracy at all. It can clearly be deduced that misdetections were much more common for Tesseract than for Google Cloud Vision, which earlier research also has shown. The denomination type values were the values that benefited the most from the implementation of the type system for the Tesseract OCR engine.

For the holistic solution, the question was whether the two extraction modes could be combined into one that outperformed them both or not. Some of the shortcomings of both of the extraction modes could not be remedied even with the combination of the two methodologies, but still Tesseract experienced its best accuracy using this extraction mode. Coincidentally, Google Cloud Vision showed the same accuracy as the extraction mode using only the type system.

It can be seen from the results that for an information extraction system, it is not always the OCR engine that produces the most accurate extractions word for word that is the most suitable for a given application. For this specific application, it can be seen that structure proved more important than accuracy in the end, but that the optimal OCR engine must be one that can achieve both accuracy and a consistent structure between documents. When developing a
system for information extraction, it is also very important to construct a sufficiently large test suite as soon as possible. Obviously, the size of the suite depends based on the variance between documents, but the test suite consisting of 115 documents was enough to be able to see extraction patterns and to properly be able to evaluate design choices made. Docker also proved very helpful in order to demonstrate the application on different devices for different stakeholders as it ensured identical results for identical input between devices.
Chapter 6

Conclusions

It can be seen as an extremely hard or in some cases impossible task to develop a 100% accurate system for information extraction with the use of OCR engines. This depends on a lot of factors, which may be boiled down to the choice of OCR engine as well as the document due to be processed. Depending on the combination of OCR engine and the document, certain actions can be taken in order to remedy and counteract the weaknesses of an OCR engine using information from the document at hand. These greatly depend on the nature of the document to be processed. Generally, the degree of standardization and the range of values for the different fields of the document are an indicator of to what extent correcting OCR engine output is possible. It has been noted that even though a document may be completely digital and in a clear format, the format of the OCR output may still be very inconsistent across a large document suite and incorrect extractions still happen to a certain extent, depending on the OCR engine chosen.

To connect back to the research question, it can be said that the characteristics needed of an OCR engine to achieve maximal accuracy for automatic information extraction of financial documents are correctness and clearly predictable output. The most important characteristic of an OCR engine in an information extraction system depends on the document to be processed. In this thesis project the Mapa CRC was the document that was processed. For this particular document, predictable and structured output proved more important than correctness of extracted words. This was also discussed in Chapter 5. For a closer look at this numerically, see Chapter 4, where the test data is displayed for both the OCR engines tested in this project, Tesseract and Google Cloud Vision. For the secondary research question of the thesis, namely Is the best
one of these two implemented systems accurate enough to be used in practice?, it can be seen that the best one of the two solutions, which is the Tesseract version, meets the accuracy requirements set by the principal.
Chapter 7

Future Work

Initially, further building on the research of this paper could be made by incorporating even more OCR engines available on the market today. The research presented in this thesis report could be extended by comparing even more OCR engines to Google Cloud Vision and Tesseract, to find potential candidates for different kinds of documents. An extensive list of available OCR engines, both proprietary and free-to-use/open-source, are listed in an article on Wikipedia [17].

Something else to look at for comparative reasons would be to test an information extraction system built on top of an OCR engine alongside a proprietary tool for PDF parsing, such as Docparser [40]. This could give hints as to whether the same accuracy can be reached with an information extraction system built on top of a generic OCR engine as opposed to Docparser’s solution, which is a holistic system that handles everything from OCR processing to extracting the actual data. It could also be interesting to perform a comparison between the method chosen here (information extraction system on top of OCR engine) and a tool for PDF parsing such as PDFminer [49]. This has to be done for unprotected PDF documents, as PDF parsers cannot handle owner-protected PDF documents, like the ones processed in this thesis project.

A completely different approach altogether for information extraction of PDF documents using a machine learning model would also be interesting to evaluate. The model could be trained on the set of correct document extractions and then evaluated on another set of documents in order to evaluate how good such a system could perform. An ML model, as opposed to the rule-based extraction model used in this project, should hopefully prove more adaptable.
to variations in the document than a rule-based extraction system.

It has been discussed previously how a modular solution was chosen as the means of constructing the extraction system for this project. However, as discussed in the same section, a monolithic solution could potentially be able to redeem some of the drawbacks that come with developing a system in a divide-and-conquer manner. It would be interesting to see how such a system would perform when compared to its modular counterpart for a given document, both in terms of accuracy but also regarding ease of development in order to give aspiring developers some guidance as to which approach that might be the most fruitful or efficient for their situation.
Bibliography


[32] Ling-Hong Hung et al. “GUIdock: using Docker containers with a common graphics user interface to address the reproducibility of research”. In: PloS one 11.4 (2016), e0152686.


[40] Docparser - Extract Data From PDF to Excel, JSON and Webhooks. [URL](https://docparser.com/).


[47] Shapiro-Wilk Test: What it is and How to Run it - Statistics How To. [URL](https://www.statisticshowto.datasciencecentral.com/shapiro-wilk-test/).

[48] Paired Samples t Test - SPSS Tutorials - LibGuides at Kent State University. [URL](https://libguides.library.kent.edu/spss/pairedsamplesttest).

Appendix A

JSON Examples

A.1 Extraction System Output

```json
{
  "name": "JOAO SERGIO NOBRE SALEIRO",
  "type_of_id_document": "NIF/NIPC",
  "number_of_id": "228400600",
  "banks": [{
    "bank": "BANKINTER, SA - SUCURSAL EM PORTUGAL (0269)",
    "credits": [{
      "guarantees": [{
        "amount": "0 EUR",
        "type": "0100",
        "number": "1"},
      {"amount": 91 222 EUR,
        "type": "1410",
        "number": "1"}
      ],
      "type_of_responsability": "Devedor",
      "product": "Credito a habitacao",
      "type_of_negotiation": "Totalmente nova",
      "judicial_litigation": "Nao",
      "number_of_debtors": "1",
      "debt": "66 509 EUR",
      "default_debt": "0 EUR",
      "overdue": "0 EUR",
      "credited_to_assets": "0 EUR",
      "potential": "0 EUR",
      "installment": "138 EUR",
    }
  ]
}
```
where the ... denotes that there may be additional credits and banks in the complete object, but the full structure can be seen in the JSON-object above.