UCLouvain

epl

École polytechnique de Louvain

Exploring Open Source AI-OCR for Efficient Personnel Invoice Verification

Author: **Narjis LARAKI**
Supervisors: **Yves DEVILLE, Pierre SCHAUS**
Readers: **Benoit DUHOUX, Xavier GILLARD**
Academic year 2023–2024
Master [60] in Computer Science

Abstract

This dissertation presents the development and evaluation of Docspotter, an open-source Python package designed to facilitate personnel invoice verification using Optical Character Recognition (OCR) technology. The motivation behind this study is rooted in the need to enhance the efficiency and accuracy of invoice processing at academic institutions, where traditional manual methods are labor-intensive and error-prone. Using a combination of CRAFT for text detection and Pytesseract for text extraction—both prominent open-source tools—this work addresses the challenge of skewed text in diverse document formats that typically hinder OCR performance. Through the implementation of Docspotter, followed by a user-friendly GUI application, this work demonstrates the potential of leveraging deep learning and OCR techniques to streamline administrative processes. The evaluation conducted across various document types—scanned, pictures, and electronic—highlights the system's precision and identifies areas for further improvement compared to industry standards, such as Azure AI Vision. The findings suggest that while Docspotter provides a robust platform for OCR tasks, incorporating adaptive processing techniques and advanced machine learning models could further enhance its accuracy and reliability. This work contributes to the field of document processing by demonstrating the practical application of open-source AI-OCR technologies in improving institutional administrative efficiency.

Contents

Acronyms

CCA Connected Component Analysis.

CNN Convolutional Neural Network.

CRAFT Character Region Awareness for Text detection.

FSF Free Software Foundation.

GPL GNU General Public License.

GRU Gated Recurrent Unit.

IP Intellectual Property.

k-NN k-Nearest Neighbors.

LSTM Long Short-Term Memory.

NLP Natural Language Processing.

OCR Optical Character Recognition.

OSI Open Source Initiative.

ReLU Rectified Linear Unit.

RNN Recurrent Neural Network.

ROI Region of Interest.

SVM Support Vector Machine.

TanH Hyperbolic Tangent.

Introduction

The increasing volume of personnel invoices in modern organizations such as UCLouvain has led to a labor-intensive and time-consuming manual processing system. This traditional approach is not only prone to errors, but is also inefficient. This method involves multiple steps that require significant human intervention, going from data entry to validation and approval. Each step can lead to delays, discrepancies in financial reporting, and possibly errors in the amount refunded to employees filing those invoices.

The motivation for this thesis is driven by the critical need to enhance both the efficiency and accuracy of personnel invoice processing. Traditional manual processing is highly prone to errors. Typical errors involve mistakes in data entry and misinterpretation of invoice details. The subsequent delays in rectifying these errors can lead to a cascade of operational inefficiencies which can drain the organization's resources but also affect employee satisfaction due to delayed or incorrect reimbursements. Additionally, the system's inability to adapt and scale with the organization's growth exacerbates these challenges, rendering it unsustainable in the long term. Moreover, the project is intrinsically worthwhile as it addresses real-world challenges faced by modern organizations. Through the automation of these processes, organizations can redirect their workforce towards more strategic and creative endeavors, thereby enhancing overall productivity and fostering innovation.

The objective of this work is to develop a system that automates the text extraction process, thereby improving the usability and reliability for the university's accounting staff. This task involves an exploration of Optical Character Recognition (OCR) and using deep learning for text detection and extraction. With this, the project seeks to overcome the limitations of manual processing by offering a more efficient, accurate and streamlined approach for handling documents within academic institutions. The goal is to:

- Handle various image formats, including *jpeg, jpg, png, bmp, webp* and *pdf.*
- Use a deep-learning-based text detector which is involved in the correction of text skewness.
- Use OCR technology to convert invoices into a searchable text format.

- Create a user-friendly interface that can be easily used and navigated through.

This document is divided into the following:

Chapter 1 is dedicated to establishing a theoretical foundation by defining key concepts such as open source and invoices. **Chapter 2** provides a comprehensive introduction to neural networks, including some of their architectures, the intricacies of training, and the fundamentals of OCR systems.

Chapter 3 builds on this theoretical knowledge by delving into the practical contributions of the project. It outlines the development process of the "Docspotter" Python package and an accompanying application, highlighting the techniques and technologies employed for text detection and extraction, particularly focusing on Character Region Awareness for Text detection (CRAFT) and Tesseract OCR.

Chapter 4 evaluates the project's performance, detailing the methodology used to test the package, the data utilized for evaluation, and the criteria for success. It discusses the results in relation to various document types and examines the project's limitations and potential improvements. The chapter concludes with a comparative analysis with Azure AI, benchmarking the package against industry standards.

This dissertation will not cover the complete integration of the developed system with the university's existing systems or databases, nor will it explore adapting the system for other types of documents beyond personnel invoices.

Part I

Theoretical Background

Chapter 1

Foundations of Open Source and Invoices

1.1 What is Open Source?

Open source refers to a type of software development and distribution model where the source code of a software is made available to the public, meaning that anyone can view, use, modify and distribute the code, as opposed to proprietary software, where the code is kept secret and users are typically not allowed to modify or share the software. Open source projects celebrate principles of transparency, collaboration, flexibility and community-driven development. [1]

1.1.1 Historical Background

While the concept of sharing software is as old as computing itself, the movement truly began to gain momentum in the 1980s with Richard Stallman's founding of the GNU Project. This pivotal event led to the launch of the Free Software Foundation (FSF), which emphasizes the importance of user freedoms and advocates for the adoption of free software while rejecting proprietary software that restricts these freedoms. The FSF promotes

four essential freedoms that define free software: running the program for any purpose, studying and modifying the program, redistributing copies, and sharing modifications [2].

Following this, Stallman created the GNU General Public License (GPL), under which all components of the GNU operating system were released. This license was revolutionary as it legally enshrined the above freedoms, ensuring that all derivatives of GNU software would remain free and open.

During its formative years, the FSF faced challenges in effectively communicating its message. The term "free" led to confusion, as it pertained not to the price but to liberty—best summarized by the maxim "free as in speech, not as in beer" [2]. This misunderstanding underscored the need for clearer communication regarding the ideological foundation of free software.

The term "Open Source" emerged in the late 1990s with the creation of the Open Source Initiative (OSI), which appealed to a broader audience, including businesses. This was partly because the OSI focused on the pragmatic aspects of software development and distribution, highlighting the economic and developmental advantages of open collaboration without emphasizing the ethical dimensions as heavily as the FSF. The OSI eventually standardized this approach by releasing its own OSI-approved licenses, which fostered a more business-friendly interpretation of software freedom.

1.1.2 Key Differences between Free Software and Open Source

While the terms "open source" and "free software" are often used interchangeably, there are nuanced differences between them. Free software emphasizes user freedoms and ethical considerations, focusing on the philosophical aspects of software freedom. Conversely, open source dissociates itself from these ethics, operating under the belief that software is just software and that ethical considerations should be associated with the people, not the software itself [3].

1.1.3 Alternative Approaches to Intellectual Property: Open Source vs. Proprietary

"Open source" and "proprietary" software represent two distinct methods of managing Intellectual Property (IP). Open source IP focuses on public benefit without pursuing profit, whereas proprietary software aims to generate revenue through subscription or license fees.

"Copyright" is a legal mechanism that grants the creator of original work exclusive rights to its use and distribution, usually for a limited time, with the intention of enabling

the creator to receive compensation. In contrast, "copyleft" is a variation of copyright that allows public use, modification, and redistribution of the source code while preventing its conversion into proprietary software.

Open source software isn't inherently anti-profit; it's based on the idea that software thrives when shared, providing greater value to a wider audience. The internet, the largest open source project, started as a means to share academic papers and has grown into a vast network due to contributions from many envisioning new possibilities.

Although open source software is free to the public, it isn't part of the public domain. Today, there are over 100 open source licenses, some of which allow derivative works that can be copyrighted and sold, thus creating commercial opportunities for open source creators [4].

1.1.4 Overview of Software Licenses

Open source licenses are "software licenses that allow content to be used, modified, and shared" [5]. They are crucial for facilitating the open source software movement, promoting collaboration, and ensuring the software remains free and accessible. Licenses can only be recognized as open source if they are OSI-approved. They also vary in permissiveness and restrictiveness. In the family of permissive licenses, we can find [6, 7]:

- **MIT License:** One of the most popular permissive licenses, allowing virtually unrestricted use, modification, and distribution. It only requires that the original copyright notice and disclaimer be retained.
- **Apache License 2.0:** Adds a grant of patent rights from contributors to users, providing additional legal protection. It also allows for the distribution of modified versions under different terms.
- **BSD Licenses:** Similar to the MIT License, with variations in clauses that may impose additional restrictions. They permit extensive reuse and modification of code.

On the other hand, popular copyleft licenses include [6, 7]:

- **GNU General Public License (GPL)**: A strong copyleft license that requires any derivative works to be released under the same GPL terms, ensuring that all modifications remain free and open.
- **GNU Affero General Public License (AGPL)**: Extends the GPL by requiring that the source code of any software running over a network also be made available to users.

- **GNU Lesser General Public License (LGPL)**: Allows linking with non-GPL licensed software, providing more flexibility for libraries.

1.1.5 Advantages of Open Source

Now that we have considered the fundamentals of open source, we can dive into the advantages, which include, but are not limited to [8, 9]:

- **Security:** Open source projects benefit from the scrutiny of a large community of developers. This collective oversight often leads to quick identification and resolution of vulnerabilities, enhancing the security of the software.
- **Cost Savings:** Open source software is generally free, significantly reducing the costs of software acquisition and maintenance. Even when there is a licensing fee, it is typically much lower than the fees associated with proprietary software. Moreover, the collaborative nature of open source development often relies on voluntary contributions, further reducing costs.
- **Flexibility:** Open source licenses offer the freedom to modify and adapt the software to specific needs. Users can add features, rectify bugs, and customize the software to their requirements without depending on a vendor's schedule.
- **Longevity and Independence:** Unlike proprietary software, which is vulnerable to the fortunes of its providers, open source software can profit from robust community support. This ensures it remains available and frequently updated, securing its long-term viability.

The combination of cost savings, enhanced security, flexibility, along with the benefits of community collaboration and long-term viability, motivated me to choose open source technologies for my project. These elements ensure that a robust, adaptable, and sustainable solution can be developed while leveraging the collective expertise and innovation of the global open source community. The goal is to create a software that is not only effective but also continuously improved and supported by a diverse and engaged community.

1.2 Invoice Fundamentals and Processing Systems

1.2.1 Definitions and Requirements

Invoices have always played a crucial role in business transactions, evolving from handwritten bills of sale to sophisticated digital documents. In fact, an invoice is defined as "a time-stamped commercial document that itemizes and records a transaction between a buyer and a seller" [10].

A valid invoice should detail the vendor's name, transaction date, the goods or services purchased, the amount paid, and the form of payment used [11]. In cases where an invoice does not contain all necessary information in a single document, it can be supported with additional documents like order summaries and proof of payment, such as bank statements. These complementary documents together provide a complete overview of the transaction and fulfill the requirements for reimbursement. Figure 1.1 illustrates a typical example used for expense reporting, providing clear information on the transaction details needed for reimbursement purposes.

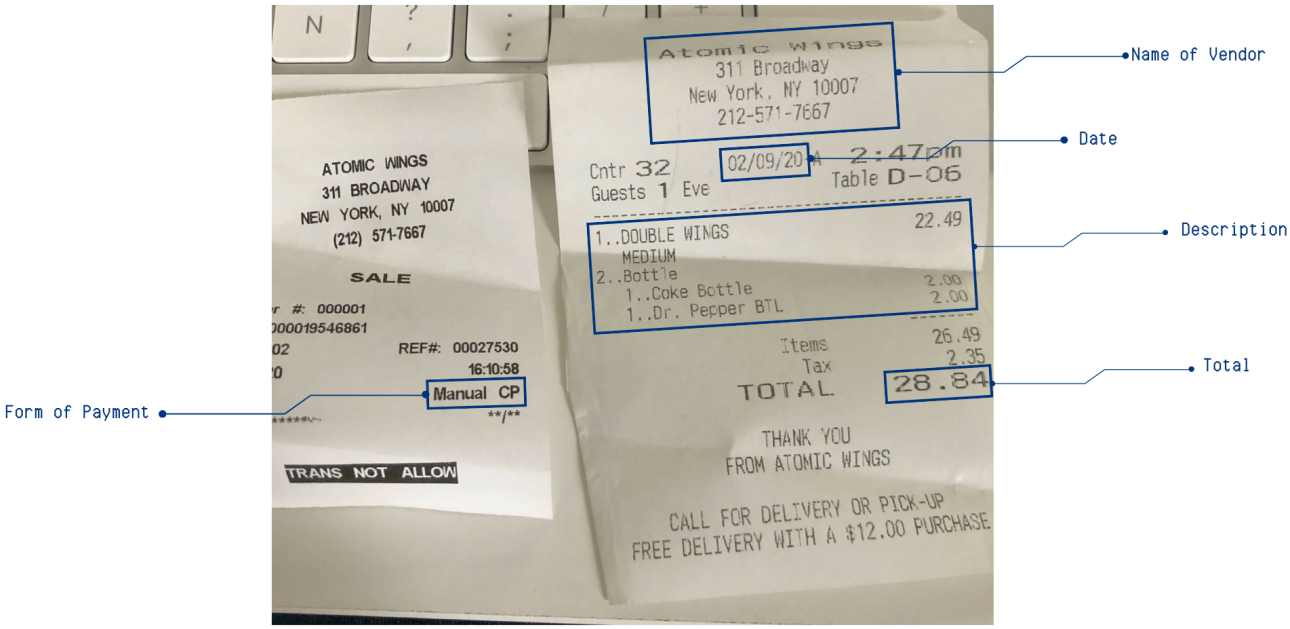

Figure 1.1: Example of receipt details

For business-related out-of-pocket expenses, employees complete an **Expense Reimbursement Invoice** to claim reimbursements. This type of document breaks down the expenses incurred, detailing the total cost of each item. It is supported by the invoices for each expense, which include receipts, bank statements, and other documents containing details necessary to verify the payment of these expenses [12]. This approach not only facilitates the reimbursement process but also helps in managing and tracking expenses to ensure all costs are accurately documented and compensated.

1.2.2 Invoice Processing Systems

Invoice processing systems are categorized into two types: Manual and Automatic. Each approach has its distinct advantages and challenges, which we will explore in this section.

Manual Processing

Manual invoice processing begins with the receipt of an expense reimbursement invoice and continues through to the final clearance of that reimbursement. Every component of this sequence is handled manually, involving a detailed comparison and matching of each item on the expense reimbursement invoice with corresponding documentation. This step is necessary for the approval of reimbursements.

Although methodical, the traditional manual method is time-consuming and labor-intensive. It requires significant manpower to manually enter data, verify accuracy, and organize paperwork. As the volume of transactions increases, manual systems can quickly become overwhelmed, leading to procedural delays and escalating operational costs. Furthermore, this approach is prone to errors, such as missed payments or inaccuracies in data entry, further compounding its inefficiencies.

Despite these drawbacks, manual processing provides direct human oversight [13], which is valuable for identifying and resolving discrepancies or inconsistencies within invoices. Additionally, this method offers flexibility as it can handle any invoice format, regardless of complexity or non-standardization. Such adaptability ensures that manual invoice processing remains a viable option for businesses that prioritize having more control over their financial operations.

Automatic Processing

In contrast, automatic invoice processing uses software technologies to manage invoices with minimal human intervention. These systems streamline the entire invoicing process, from receipt to payment approval, significantly reducing the time required to handle invoices by automating routine tasks and minimizing the risk of human errors.

Although the initial setup costs for automated systems might be higher, they can lead to significant cost savings by reducing labor costs and preventing payment errors. However, these systems also introduce complexities related to their implementation and ongoing maintenance.

Chapter 2

Deep Learning and Optical Character Recognition

2.1 Deep Learning for Document Processing

Before going further, it is important to clarify some fundamental concepts that are essential for understanding the application of deep learning in enhancing document processing systems. We will briefly explore neural networks, including some of their architectures and their ability to mimic human brain functions. Additionally, an overview of how these architectures are trained will be provided.

According to IBM, deep learning is defined as a "subset of machine learning that uses multi-layered neural networks, called deep neural networks, to simulate the complex decision-making power of the human brain" [14]. This approach is renowned for its capacity to autonomously learn and evolve from experience without the need for explicit programming. Deep learning is applied across a wide range of industries and domains, including image and video recognition, and plays an integral role in modern Natural Language Processing (NLP). This includes powering technologies behind *language translation, sentiment analysis,* and *text generation.* In our context, deep learning significantly improves text detection and extraction, especially from images and unstructured documents. These enhancements are vital for OCR technologies, which will be further explored in section 2.2.

2.1.1 Neural Networks

A neural network is a set of algorithms inspired by the human brain. The idea is to simulate the brain's activity, particularly in tasks such as pattern recognition and the transmission of information across various neural connection layers. A neuron serves as the fundamental computational unit within a neural network, organized into layers.

It's important to acknowledge that neural networks can assume diverse configurations contingent on the nature of the data being analyzed.

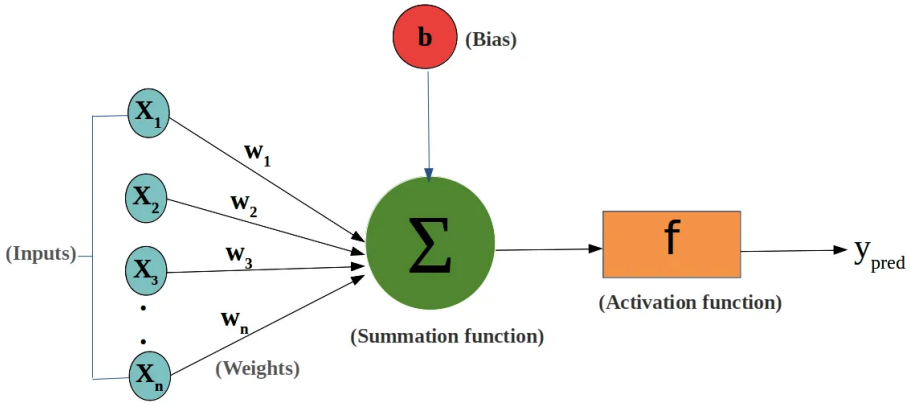

Figure 2.1: Neural Network

In neural networks, data flows through each neuron via a connection. Each connection has a specific weight that regulates the data flow. In Figure 2.1, X_1 and X_2, which could be integers, floats, etc., are adjusted based on the weights of these connections, which is similar to the dendrites in an artificial neuron.

To process information, an activation function is required. Activation functions help the network learn complex patterns. In a human brain, an activation function, also known as an activation potential, ultimately decides what gets passed to the next neuron. These functions are crucial because they enable the nonlinear modification of data, allowing the neural network to learn and perform more complex tasks. The most common types of activation functions include the *Rectified Linear Unit (ReLU), The Sigmoid function* and the *Hyperbolic Tangent (TanH) function* [15].

2.1.2 Neural Network Architectures

A neural network can take various forms depending on the data being processed, its complexity and the method of processing used. Neural network architectures are essentially the blueprints that define the structure and functionality of these computational models. Each architecture is designed to address specific types of problems, from image recognition to sequential data processing, by strategically layering and connecting neurons in various patterns. In this section, we will delve into the most relevant neural network architectures for document processing, particularly in the context of enhancing OCR systems used in invoice processing. These include Convolutional Neural Networks (CNNs) for their superior image analytical abilities, Recurrent Neural Networks (RNNs) and Long Short-

Term Memory (LSTM) networks for their adeptness at handling sequential and contextual data.

Convolutional Neural Network

As previously stated, a Convolutional Neural Network (CNN) is an architecture well suited for analyzing visual data. Here, information enters through an input layer and exits through an output layer, similar to a traditional feed-forward neural network.[1] Between these, CNNs contain several specialized layers including convolutional layers, pooling layers, and often one or more fully connected layers towards the end. The convolutional layers are specifically designed to automatically and adaptively learn important features from the input data, making them particularly effective for handling complex and varied information such as images [16].

In tasks that require processing detailed and hierarchical features from visual data, it is common to employ multiple distinct convolutional networks, each tailored to focus on different aspects of the information. This is particularly evident in applications like image classification, object detection, and image recognition. The mechanism of convolutional layers, which perform filtering operations that capture spatial hierarchies and patterns, is central to the effectiveness of CNNs in these domains.

Recurrent Neural Networks

Recurrent Neural Network (RNN) architectures are more common than one might think. These networks are used for various tasks such as speech recognition and translation. Currently, RNNs are the most popular architecture for the extraction of relevant information in unstructured documents [17]. Indeed, they are particularly useful when dealing with sequential information, which is a specific order in which one piece of information follows another. An RNN uses previous information to perform a task on the current information.

However, this type of architecture has a short-term memory. In order to compensate for this problem, two specific types of RNNs have been created:

[1] A convolution neural network is a special type of feed-forward neural network designed to handle data with a grid-like structure, such as images

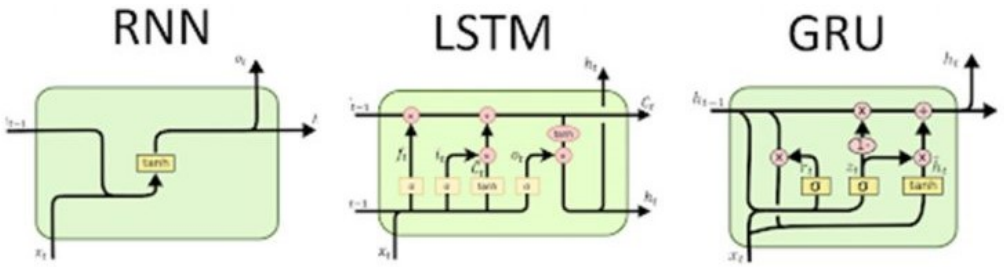

Figure 2.2: Architectures of LSTM and GRU cells compared to RNN

RNNs can either be based on Long Short-Term Memory (LSTM) cells or Gated Recurrent Unit (GRU) cells.

LSTM cells and GRU cells generally function similarly to RNNs, but these cells have the capability to maintain state over extended periods through "gate" mechanisms. These gates can learn to identify which pieces of information should be retained or discarded within a sequence.

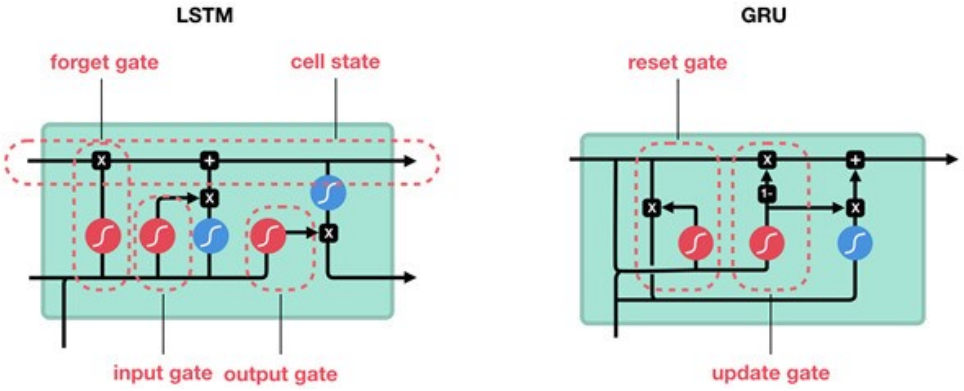

Figure 2.3: Left: Diagram of an LSTM cell; Right: Diagram of a GRU cell

An LSTM neuron includes an internal memory managed by three control gates (see Figure 2.3, left side) [18]:

- **Input gate:** Determines whether the input should alter the cell's content.
- **Forget gate:** Decides which information requires more focus and which can be ignored.
- **Output gate:** Decides whether the content of the cell should influence the output of the neuron.

This gating mechanism enables LSTM cells to enhance their memory capacity. glsgru cells on the other hand are considered part of the newer generation of RNNs and are quite similar to LSTMs but with some distinct differences. GRUs do away with the internal state and instead use the hidden state to transfer information. They feature two gates instead of three (see Figure 2.3, right side) [18]:

- **Reset gate:** Allows the model to decide how much of the past information to forget.
- **Update gate:** Helps the model determine how much information from previous steps should be carried forward or retained for future steps.

2.1.3 Training a Neural Network

Training a neural network involves several steps. The first step is to initialize the parameters, including the weights and biases (as seen in Figure 2.1). A bias is a constant that helps the model adjust more easily to the presented data, while the weight indicates the degree of influence an input will have on the output.

The training then consists of an iterative process involving an optimization algorithm aimed at minimizing the difference between the actual value and the network's prediction. The most commonly used algorithm for this purpose is *gradient descent*. A gradient represents the value used to adjust the network's parameters; the larger the gradient, the more significant the adjustments, and the smaller the gradient, the smaller the adjustments. There are several versions of the gradient descent algorithm, with mini-batch gradient descent being the most popular.

Mini-Batch Gradient Descent

In this method, N random elements from a labeled dataset (training set) are introduced at each iteration. This balances training time and the randomness of the inputs. An iteration with this algorithm proceeds as follows:

- A mini-batch containing N random samples from our dataset is introduced.
- Calculations are performed on each layer of the network, resulting in a prediction. This step is also known as forward propagation or forward pass.
- The network then compares and evaluates the prediction against the actual value using a loss function. This function calculates the error value, indicating the accuracy of the network's predictions. The smaller the value, the more efficient the network.
- The network uses the error value to perform backpropagation, which calculates the gradient for each node in the network. Mathematically, this is a vector that indicates

the direction in which the loss function increases most rapidly, hence ideally, we should move in the opposite direction to minimize it.

- Once the gradient vector is obtained, the network's parameters are updated by subtracting the corresponding gradient value from the current value, multiplied by the learning rate.

These steps are repeated as long as the loss value and output metrics do not deteriorate.

2.2 Optical Character Recognition Technologies

OCR technology represents a key element in the extraction of data from documents. It is used to convert different types of documents into machine-readable text. This technology has made it possible for companies to digitize vast amounts of information, simplifying the processes of searching, storing, and analyzing data.

2.2.1 OCR Workflow

This section will explore how OCR operates, detailing the algorithms and methods employed to identify characters and transform them into digital text. [19]

Pre-processing

The effectiveness of OCR first starts with the quality of the input image. OCR softwares use several techniques to ensure a better recognition accuracy. Common pre-processing techniques include:

- **Deskewing:** This process refers to correcting any tilt a few degrees clockwise or counterclockwise to align the text properly for analysis.
- **Noise Reduction:** Removing random variations of brightness or color in images to enhance text clarity.
- **Grayscale Conversion:** Converting the image to grayscale to reduce computational complexity.
- **Binarization:** Converting an image to black-and-white (binary), where the text is typically black and the background is white. This is an effective way to distinguish text from the background.
- **Line removal:** Detecting and removing horizontal and vertical lines that do not form part of the text characters.
- **Normalization:** Normalize aspect ratio and scale.

Text Detection

Once the image is processed, the OCR software needs to locate the text within the image, this involves:

- **Layout Analysis:** Identifying columns, paragraphs, captions and blocks of text. This helps in understanding the structure of the document, particularly multi-column layouts and tables.
- **Script Recognition:** In documents containing multiple languages, the script can change at the word level, making it crucial to identify the script accurately before applying the appropriate OCR to process the specific script effectively.

Character Segmentation and Recognition

This is the core phase of OCR, where actual text recognition occurs. Character segmentation is a key component of OCR, which is focused on identifying and converting individual characters within an image into machine-encoded text. Once the text regions are identified, **character segmentation** comes into play to segment the text into smaller units. This might involve breaking down a page into lines, lines into words, and words into individual characters. Segmentation involves multiple levels. We first have page segmentation where we would isolate areas that contain text from other non-text elements. Once those are isolated, the OCR detects the horizontal alignment of text to distinguish between separate lines. These lines are then broken down into words. This involves detecting spaces between groups of characters. Finally, the last step of segmentation involves isolating individual characters. [20]

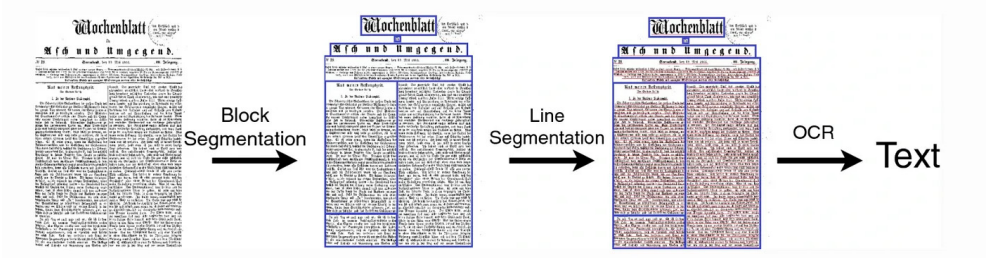

Figure 2.4: Page and line segmentation example

In order to achieve that, different techniques can be used. For example, **Histogram Analysis** creates histograms of pixel distribution along the horizontal and vertical axes of the text image. Peaks and valleys in these histograms help identify where lines and spaces exist.

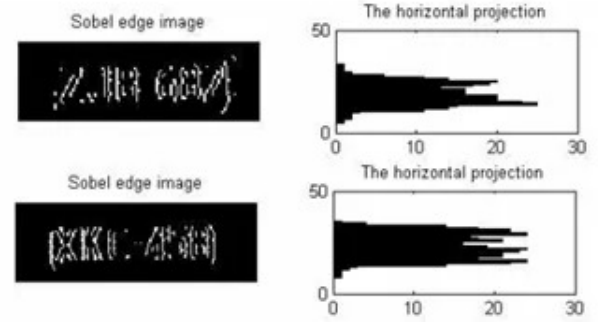

Figure 2.5: Horizontal projection

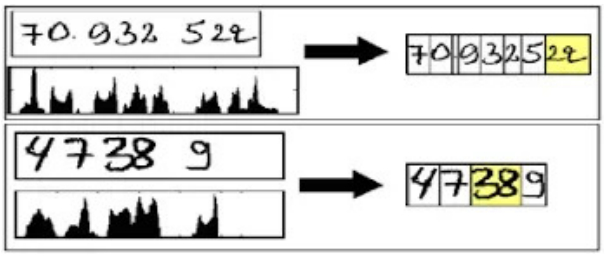

Figure 2.6: Vertical Projection

Other algorithms like **Connected Component Analysis (CCA)** involve identifying regions of connected pixels that are likely to form characters or symbols.

Following character segmentation, the next step is the recognition or classification of the segmented characters into readable text. This stage typically employs various machine learning techniques to accurately identify characters. Traditional methods include algorithms such as **Support Vector Machine (SVM)** and **k-Nearest Neighbors (k-NN)**. However, more sophisticated approaches often involve deep learning technologies, including **CNNs** and **RNNs** [21], which have been seen in section 2.1.2. CNNs excel in picking up patterns in spatial data, which helps recognize stylistic and structural patterns in character images. RNNs on the other hand are useful for handling sequences of data, which is beneficial for understanding the context within strings of characters and helps enhance the accuracy of the character recognition process, especially in cursive or connected scripts.

Once the characters are recognized, the next task is to combine them logically to form words and sentences. They are first grouped based on their proximity and alignment. Characters that are close together and aligned in a straight line are likely to form a word. This is often guided by the spacing between characters, which can vary depending on the font and size of the text.

Post-processing

This final step tries to improve the OCR accuracy with the help of different elements:

- **Lexical Analysis**: Some OCR perform lexical analysis to match groupings of characters against known words in a language dictionary. This could be words in English or a more technical lexicon for a specific field for instance. Some advanced OCR systems also use context to predict words. For instance, the word "the" in a sentence is likely to be followed by a noun or adjective.

- **Error Correction**: Once words are formed, OCR systems often apply error correction algorithms to fix common mistakes. This may involve spell checking based on a standard dictionary and a grammar analysis checking to correct structural errors in sentences.

2.2.2 Challenges in OCR Recognition

Unfortunately, OCR is not perfect. While it has made significant improvements in digitizing text, it still faces numerous challenges that can affect its performance and accuracy [22]. Some of the challenges include:

- **Variability in Text**: While OCR systems must be able to recognize different fonts and styles, each font has unique characteristics that can confuse OCR if it hasn't been trained on that specific font.
- **Image Quality**: Text captured at a low resolution may not have enough detail which makes it difficult for the OCR to distinguish between similar characters like 'O' and '0'. Lighting conditions may have an impact on the image as well, obscuring characters which ends up leading to inaccurate recognition. Text that is not perfectly aligned or have inconsistent text orientations can also complicate the recognition process.
- **Handwritten Text**: Handwriting varies from person to person. OCR struggles with the inconsistency in handwritten texts, including cursive handwriting where letters are connected. Poor handwriting can also be indecipherable.
- **Contextual Interpretation**: Even if individual characters are recognized accurately, assembling them into meaningful words and sentences can be problematic without a strong understanding of the language and context.

Part II

Contributions to Personnel Invoice Verification

Chapter 3

Text Detection and Extraction

3.1 Dataset Analysis

In order to fully understand the nuances of a typical case of invoice verification input, it is essential to examine real-life examples provided by my promoters. In this section, we will conduct a comprehensive analysis of the 187 documents received, which plays a crucial role in identifying the key aspects that require attention during the implementation phase. These documents, ranging from receipts and transportation tickets, to bank statements, offer a rich source of data. We can categorize them into three main types:

- **Electronic Documents**: E-documents are any type of electronic media content that is intended to be used in an electronic form. They represent a digital alternative to traditional paper copies or printouts. [23] Making up **77.01%** of the dataset, these documents include various types of invoices.

  E-documents do not encounter the same problems associated with OCR because they are consistently clear, evenly lit and lack skewness, ensuring reliable readability and processing without the typical issues found in other types of documents.

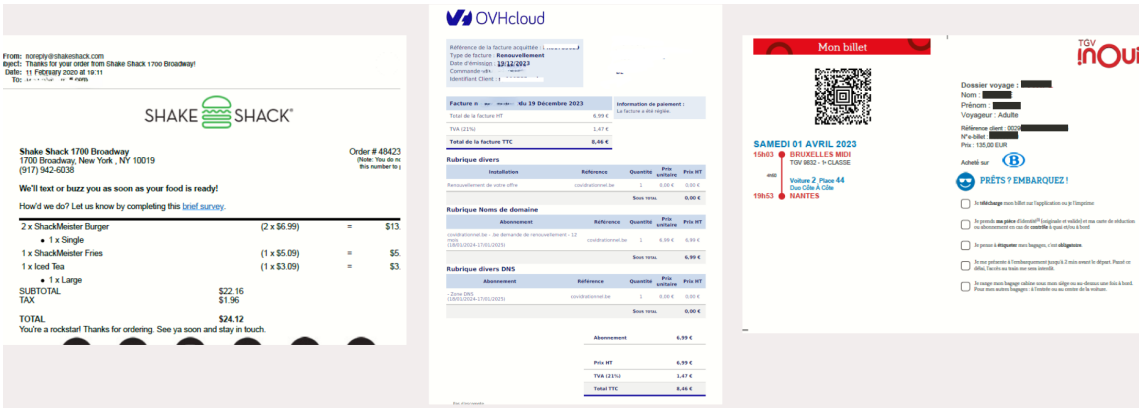

Figure 3.1: Examples of electronic documents

- **Scanned Documents**: These are physical papers that have been digitized using a scanner. Unlike documents that are originally created in digital formats, scanned documents are image-based, which means that they do not contain selectable text. A single scanned document can contain multiple items (e.g., more than one receipt) and represent the same types of elements previously mentioned. They account for **12.3%** of the documents.

  Unlike electronic documents, scanning can sometimes introduce issues that hinder OCR performance. Such issues include poor image quality, skewed text and inconsistent lighting.

Figure 3.2: Examples of scanned documents

- **Pictures of Documents**: Pictures are images taken with a camera. Like scanned documents, pictures can include multiple documents, and even other real-life elements, which adds complexity to the image.

In pictures, lighting and shadows are often uneven, particularly because the documents are not always completely flat and may have creases. This introduces skewness, angles and shadows that affect OCR detection and accuracy. While they only constitute **10.69%** of the dataset, they represent a significant challenge, which the application Docspotter tries to address.

Figure 3.3: Examples of pictures of documents

3.2 Implementation of the Docspotter Python Package

Docspotter[1] is a Python library designed to extract information from specific document images by combining text detection and extraction technologies.

One of the primary challenges in processing document images is the presence of skewness, where text is not aligned horizontally or vertically, which can hinder accurate text recognition and extraction. To address this, Docspotter uses two tools in sequence: CRAFT and Tesseract OCR. This method was initially observed in a Medium article by Bibek Thapa [24].

CRAFT is initially employed to detect text regions within the images. Unlike many text detection systems, CRAFT is good at recognizing text lines and blocks even when they are presented at angles or skewed. This capability is important because it ensures that all textual content (or atleast most of it) is accurately identified and prepared for the next stage of processing.

[1] `https://github.com/narjislaraki/docspotter`

Once text regions are detected and their orientations are determined, the identified text areas are adjusted with OpenCV to correct any skewness. This preprocessing step standardizes the text orientation, making it more amenable for extraction. Here, Tesseract OCR comes into play. While Tesseract is a powerful OCR tool know for its accuracy in converting images to machine-encoded text, its performance can degrade when handling skewed texts. By preprocessing the images with CRAFT and OpenCV to rectify any alignment issues, we enhance Tesseract's ability to perform its role more effectively.

This section will explore in detail the technologies implemented for text detection and extraction, detailing how they collectively improve the efficiency and accuracy of the Docspotter library.

3.2.1 Text Detection: CRAFT

CRAFT is a deep neural network that uses a dual scoring system [25] to detect and label text regions within complex backgrounds. The system consists of a **region score** used to identify individual character within the image and an **affinity score** which is the extent that a substance is likely to combine with another, means that it groups the characters into a single unit. These elements combined give a bounding box for each word [26].

Architecture

CRAFT [27] is founded on a CNN architecture based on VGG-16[2] which extracts feature maps from input images. These feature maps encode detailed spatial information about text regions at various scales, which is important for detecting text with different font sizes and shapes. The decoding part of the model uses skip connections similarly to U-Net[3].

Training

This model has been trained using a combination of synthetic data that provides labeled examples, and real-world data which introduces the model to practical scenarios. The use of synthetic data is particularly important in order to recognize a wide variety of text styles and distortions without the need for extensive manual labeling.

[2]CNN model supporting 16 layers with 13 convolutional layers and 3 fully connected layers. It is renowned for its capacity to deliver strong results when it comes to computer vision tasks, like image classification and object recognition. [28]

[3]CNN designed for image segmentation tasks, especially in the biomedical field.

The training is based on weakly-supervised learning. With this method, the training data is typically labeled but the labels are imprecise or incomplete. This approach is useful when it's too impractical to obtain a fully labeled training dataset, or if it's too costly to do so. It's also commonly used in this kind of computer vision tasks where bounding box annotations are hard to obtain.

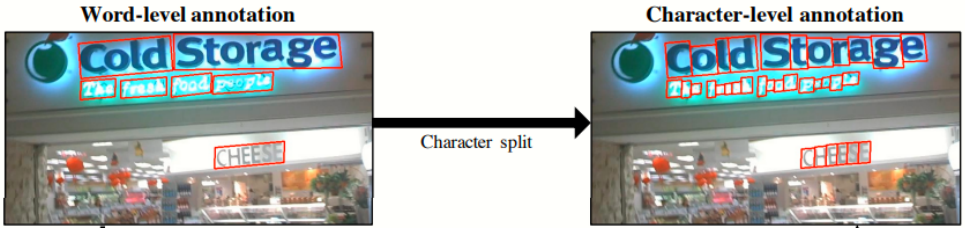

Figure 3.4: Transformation from world-level annotation to character-level

In CRAFT's case, real images in a dataset usually have word-level annotations and weakly-supervised learning allows to generate character-level bounding boxes instead, as seen in Figure 3.4. The model uses a confidence map to evaluate the reliability of its predictions during training. This is done by comparing the number of characters detected by the model to the ground truth which is the actual number of characters.

In a training dataset with word-level annotations, for any given sample w, let $R(w)$ represent the region of the bounding box, and $l(w)$ the length of the word within sample w. During the process of character splitting, the derived character bounding boxes and their respective character lengths, denoted as $l_c(w)$, are obtained. The confidence score $s_{conf}(w)$ is then calculated for the sample w using the following formula:

$$s_{\text{conf}}(w) = \frac{l(w) - \min(l(w), ||l(w) - l_c(w)||)}{l(w)} \tag{3.1}$$

A higher confidence score means that the example has a greater impact on training.

Pytorch Implemetation for CRAFT Text Detector

For the Docspotter project, we have employed the Pytorch implementation of the CRAFT text detector. As previously mentioned, OCR struggles with skewed text. Bringing CRAFT into this project allows us to detect the text regions and skew them accordingly in order to improve OCR performance.

License

CRAFT-pytorch is distributed under the MIT License which means it can be used for private, educational or commercial purposes without restriction.

3.2.2 Text Extraction: Tesseract OCR

Tesseract OCR is an open-source OCR engine that can extract text from images and convert it into a machine readable form. It was originally developed by Hewlett-Packard in the 1980s and later sponsored by Google. It can be used with many programming languages through wrappers. For Docspotter we use Pytesseract[4].

Processing follows a step-by-step pipeline [29]. It begins with preprocessing the input image during which adaptive binarization is applied to outline the different elements of the image. OCR performance is enhanced on black-on-white text. By applying binarization, connected components are more easily identified; these are simply set of pixels of the same color with a path between any two pixels. These outlines define the shape and size of each component, which is important for character recognition. The outlines are then compiled into blobs. Blobs are a fundamental data structure used during the text recognition process and represent potential characters or parts of characters.

These blobs are then organized into lines of text which are analyzed to determine whether they consists of fixed pitch or proportional text [30]. **Fixed pitch text** refers to text where each character occupies the same amount of horizontal space (similar to text produced on a typewriter), while **proportional text** involves characters that occupy varying amounts of space.

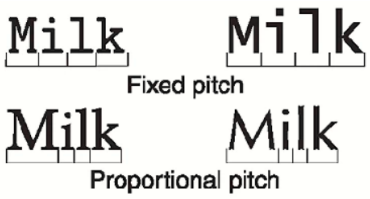

Figure 3.5: Fixed pitch vs proportional pitch

Categorizing into these two types allows the use of different methods for word segmentation. Fixed pitch is straightforward, as each character cell can be sliced without ambiguity. Proportional text requires more nuanced processing to distinguish between definite spaces (separations between words) and fuzzy spaces (smaller separations that might not always represent a break between words).

Character recognition then goes through a two-pass process. First, Tesseract attempts to recognize each word. If the word is satisfactorily recognized, it is passed to an adaptive classifier as training data. The second pass allows the classifier the re-evaluate words that were not recognized well in the first pass.

Licenses

Tesseract is distributed under the Apache License 2.0, This permissive license does not require derivative works to be distributed under the same terms and explicitly grants patent rights from contributors to users. Conditions include the requirement to include

[4]Tesseract for Python

a copy of the license and a notice providing attribution to the original authors in any redistributed program.

Pytesseract on the other hand is distributed under the GNU General Public License version 3 (GPL-3.0), which is a copyleft license.

3.2.3 Implementation Details

Docspotter includes several functional components designed to preprocess images, detect regions, skew them appropriately, and extract readable text. Here's a breakdown of the different functions:

Helper Functions

These helper functions are designed for preprocessing images in a pipeline (more specifically, in the function *'skew_and_extract_text'* which will be described later). It includes:

- **__process__image**: This function is desgined to preprocess the image by converting it to grayscale and applying a dilation morphological operation. This allows to enhance the visibility of the text from the background, which is beneficial for OCR processes. More specifically, grayscale conversion simplifies the image by reducing it to a single intensity channel, this is done with OpenCV's *'cvtColor'* function.

  After conversion to grayscale, the function applies a dilation operation using a kernel of elliptical shape, defined by *'getStructuringElement'* with a size of $3 * 3$. It allows to expand the areas of high intensity of the image which helps in closing small gaps within the characters and can make disjoint parts of a character connect better.

- **__resize__and__process__image**: This function resizes an image to a specific width while maintaining the aspect ratio. Since we process small parts of an image at a time, it is important to resize, ensuring that they are normalized to a standard size for consistent processing, thus making them clearer for the OCR. It then calls the previous function for further processing.

Main Functions

- **create__craft__detector**: This function configures an instance of the craft detector with parameters such as:
  - *crop_type*: Refers to the type of cropping method used by CRAFT detector, it is by default "poly". Choosing poly allows the detector to more accurately

follow the contours of irregular text shapes, which is beneficial for texts that are not strictly horizontal or are shape unusually.

- *cuda*: Determines whether the model should use GPU acceleration with CUDA or not. Setting this to 'True' enables the detector to leverage the GPU which speeds up the text detection process. This is great for performance efficiency but requires a compatible GPU. It is set to "false" by default.
- *text_threshold*: This is a confidence threshold that determines what confidence score the detected text regions must achieve in order to be considered valid. It typically ranges from 0 to 1. The higher it is, the more conservative the detector will be and while it might lead to fewer detections, they'll have a higher reliability. The default value is 0.8.
- *link_threshold*: Similar to text_threshold, this parameter sets the confidence level required to consider links between different parts of text as valid. This allows to identify characters that are part of the same word or text block. The default value is 0.4.
- *low_text*: Finally, this parameter controls the detection of low-confidence text regions. A text region with a lower confidence score will be considered less likely to be text and thus filtered out. The default value is 0.25.

The default values have been chosen to balance detection accuracy and computational efficiency.

- **detect_text**: This function encapsulates the process of detecting text within an image using the CRAFT instance. It first invokes CRAFT's *'detect_text'*; the model then processes the image and identifiers areas likely to contain text. It returns a list of detected text boxes.

- **extract_text** and **extract_data**: Both these functions are integral to the text extraction process using Pytesseract engine. *extract_text* is designed to directly extract readable text from an input image using Pytesseract's built-in function *'image_to_string'*. This function is particularly useful when the goal is simply to obtain the textual content of an image without needing detailed information about the text bounding boxes, formatting or confidence scores. *extract_data* extends the power of the OCR process by not only recognizing the text but also providing a detailed breakdown of each textual component of the image. In this function, we use Pytesseract's image_to_data which returns a dictionary with detailed metadata about each text segment, including its location, size, confidence level and other relevant attributes.

- **skew_and_extract_text**: This is the main function of this package. It addresses the common challenge in OCR applications of recognizing text that is skewed or

oriented in other ways than the standard horizontal alignment. This function has a dual purpose: it first adjusts the orientation of detected text regions to a standard alignment, then processes these regions for better OCR accuracy and extracts the text. We can break it down like:

1. *Input Parameters*: The function receives two parameters which include the input image and a list of Region of Interest (ROI) that have been generated by *detect_text*.
2. *Processing each Text Region*: For each ROI, the function checks if it contains at least three points (in order to form a polygon). It then calculates the width and height of the ROI based on the Euclidean distance between the respective points.
3. *Perspective Transformation*: New destination points for the perspective transformation are calculated, representing a rectangle. This part used OpenCV's function *'getPerspectiveTransform'* which performs a perspective transformation on an image, making it appear as though the photo had been taken directly from the front.
4. *Image Processing*: The new warped image is further processed by *_resize_and_process_image* to enhance its features for the OCR.
5. *Text Extraction*: We then extract the text using the *extract_text* function and split it into individual words in case multiple blocks of text were identified. This part of the function also calculates the minimum bounding rectangle around the original ROI which is then stored alongside the found value.
6. *Output*: Finally, the function returns 'values' and 'rois' as a tuple. 'values' contains the words extracted from each ROI, and 'rois' contains the bounding boxes corresponding to these words.

Figure 3.6: Example output using the Docspotter package, showing from left to right: original image - skewed image - extracted text

3.2.4 Deployment to Pypi

In order to allow developers to easily install and integrate Docspotter into their projects, the package has been deployed to PyPi. PyPi, or The Python Package Index, is a repository of software for the Python programming language. Packages deployed on PyPi can easily be downloaded using pip.

The deployment implies the preparation of the package for public distribution. For this purpose, two configuration files are used: *'setup.py'* and *'setup.cfg'*. These files contain metadata and configuration details necessary for the package distribution.

3.2.5 License

Docspotter uses the GNU General Public license version 3.0 (GPL-3.0) since it has integrated Pytesseract, which is also distributed under GPL.

3.3 Application of Docspotter

To properly showcase Docspotter's capacities, a small application[5] has been implemented using PysimpleGUI[6]. The user manual can found in appendix A in which a real-life example is used.

3.3.1 Implementation Details

Interface

PysimpleGUI is a python package that allows developers to create desktop application very easily. The API is simple and intuitive. By default, it uses Tkinter as the underlying technology but also supports Qt, WxPython and Remi (for web-based applications).

The interface is centralized in the file *gui.py* which contains a series of helper functions that are defined to handle different aspects of the GUI, such as displaying popups, images, file explorers and search results. The main event loops of the application listens for user interactions, such as button clicks for browsing files or search for numerical values within documents, this part dictactes the flow of the application based on the user input. The first interaction with the Docspotter package begins here as we create a craft detector instance that will be used throughout the execution. In case the user input is invalid or if no files are selected, the application gives out a feedback.

Processing Pipeline

When the user has chosen files to process, the files go through a processing pipeline for extracting text from images and PDF documents. When the *'process_files'* function is invoked, the following steps occur in order:

- **Directory Setup**: If it hasn't been done already, two directories are created, one for cached results and another for temporary files.
- **File Collection**: If the user has chosen multiple files, they're grouped into a single list. If a directory is specified, all the files within it and its sub-directories are included.
- **Hash Calculation and Cache Check**: A hash value is computed for the collected files. Since the information about files is cached, it can determine whether the set of files has already been processed or not, thus making the process much faster as it only has to fetch the cached file.

[5] `https://github.com/narjislaraki/docspotter_app`
[6] `https://github.com/PySimpleGUI/PySimpleGUI`

- **Multithreading and File Processing**: If the files have never been processed before, the function creates a thread pool. Each file is then handed off to the *'_process_single_file'* function within its own thread. The use of parallel processing allows to use the CPU more efficiently, since we might deal with a large number of files and the OCR tasks can be computationally intensive.
- **Single File Processing and Text Detection and Extraction:** if we're dealing with a PDF, each page is converted into an image using the Python package pdf2image[7] and saved temporarily. Other files are then processed with the *'_extract_and_save_information* function. This function calls Docspotter package to find text regions then correct their orientation and extract the text. The bounding boxes and text values are returned and made serializable.
- **JSON Data Compilation and File Creation**: A JSON entry is then created for each file, having the following structure:

```
{
"index": file_path,
"values": values,
"bounding_boxes": bboxes
}
```

  Finally, the entries are then in a JSON file under the unique hash as a filename, marking the end of pipeline execution.

Searching for values

Now that the files have been processed, we only need to find and rank the extracted text based on their similarity to the user-provided input. For that, we first begin by opening and reading the JSON file which contains the previously extracted values, and for each value, the Levenshtein distance between the user input and the extract text is calculated. In this context, the use of Levenshtein distance is effective for many reasons:

- OCR isn't perfect and can introduce errors in recognition. For example, if OCR reads 0 as the letter O, the Levenshtein distance can still identify the characters as similar.
- Levenshtein allows the identification of values that are partially correct. For example, we could search for '29' instead of '29.01'.

We then filter the values based on the threshold and the result is stored in a dictionary alongside their bounding box. The search results are then displayed from smallest to biggest distance.

[7] https://github.com/Belval/pdf2image

Chapter 4

Results and Analysis

In this chapter, we will evaluate the accuracy of the Docspotter application in locating values across various invoice formats. Additionally, we will assess its precision and recall in extracting and aligning numeric values from these invoices. The goal is to pinpoint prevalent OCR errors and trends in value extraction discrepancies across different document formats (scanned, electronic, and images). Naturally, certain formats demonstrate superior performance compared to others. How do these different types of documents impact the performance of OCR technologies?

4.1 Methodology

4.1.1 Data Description

For this evaluation, a dataset containing the three types of documents outlined in section 3.1 was used. The ground truth for each document was manually established by meticulously reviewing each image to accurately identify and document the sought values. The dataset includes various challenges, including multiple documents on a single page, skewed text, handwritten annotations, and uneven lighting, to test the tool's robustness and adaptability across different document conditions. The dataset includes the following:

Document Type	**Nb of Documents**	**Nb of Values Sought**
Scanned	16	24
Pictures	14	24
Electronic	12	24

Table 4.1: Overview of dataset content

The values can be found in Appendix B.

4.1.2 Success Criteria

Values identified by the OCR are categorized as follows:

- **Exact Match:** Values that precisely match the expected figures, such as 28.84 found as 28.84. This category also includes scenarios where a comma replaces a dot, e.g., 28,84, and instances including currency symbols, which do not alter the numeric value.
- **Near Match:** Values that approximate the expected number by matching the integer part but excluding some or all decimals. For example, 1131 would be accepted as a near match for 1131.23.
- **Linked Match:** Values that are associated with the correct item (with the bounding box) but inaccurately extracted due to OCR inaccuracies (e.g., 28.84 linked to 278.84).
- **No Match:** Indicates that no value has been identified.

4.1.3 Evaluation Metrics

For a comprehensive evaluation of the OCR system, precision and recall have been selected as the primary metrics due to their relevance in assessing both the accuracy and completeness of text extraction processes. Given the system's nuanced approach to identifying matches, traditional concepts of True positives (TP), False Positives (FP), False Negatives (FN) and True Negatives (TN) are not entirely applicable. Therefore, we will adjust these metrics to better suit the evaluation framework. Additionally, and given the use of Levenshtein distance to quantify the closeness of matches, we introduce a weighting scheme into the metric calculations. This method effectively captures the varied correctness of OCR results and provides a nuanced evaluation of the system's performance. Weights are assigned to each category of matches as follows

- *Exact Match Weight (w_1):* 1.0, since they are perfectly correct.
- *Near Match Weight (w_2):* 0.5, assuming these are half as valuable as exact matches.
- *Linked Match Weight (w_3):* 0.25, assuming these are less valuable than near matches.

These weights help integrate different levels of match correctness into the evaluation metrics:

- **Precision**: Measures the correctness of the positives that the system has predicted:

$$\text{Precision} = \frac{w_1 \times \text{Exact} + w_2 \times \text{Near} + w_3 \times \text{Linked}}{\text{Total Predicted Positives}} \tag{4.1}$$

Here, *Exact*, *Near*, and *Linked* represent the counts of each type of match, with the *Total Predicted Positives* being the sum of all identified values by the OCR.

- **Recall**: Assesses the system's ability to capture all relevant instances, calculated by:

$$\text{Recall} = \frac{w_1 \times \text{Exact} + w_2 \times \text{Near} + w_3 \times \text{Linked}}{\text{Total Actual Positives}} \tag{4.2}$$

Total Actual Positives includes all types of matches plus the cases where no matches were found, representing every instance where a value should have been identified.

4.2 Results

4.2.1 Performance by Document Type

Document Type	Nb of Exact Matches	Nb of Near Matches	Nb of Linked Matches	Nb of No Matches	Precision	Recall
Scanned	17	1	1	5	93.42%	73.96%
Pictures	10	0	2	12	87.5%	43.75%
Electronic	20	1	1	2	94.32%	86.46%

Table 4.2: Comparative effectiveness across different document formats

To clearly demonstrate how the values for precision and recall were computed, consider the following details for scanned documents:

Scanned documents in the dataset consisted of 17 *exact matches*, 1 *near match*, and 1 *linked match*, while 5 instances resulted in *no matches*. Despite finding 19 values (representing the *total number of predicted values*), the *total number of actual positives* we aim to identify is 24. As outlined in subsection 4.1.3, weights are assigned as 1.0 for exact matches, 0.5 for near matches, and 0.25 for linked matches. These weights help in calculating the adjusted values for both precision and recall:

$$\text{Precision} = \frac{1 \times 17 + 0.5 \times 1 + 0.25 \times 1}{19} = \frac{17.75}{19} \approx 0.9342 \text{ or } 93.42\% \tag{4.3}$$

$$\text{Recall} = \frac{1 \times 17 + 0.5 \times 1 + 0.25 \times 1}{24} = \frac{17.75}{24} \approx 0.7396 \text{ or } 73.96\% \tag{4.4}$$

These calculations provide a basis for understanding the tool's performance in terms of precision and recall. Let's now explore the detailed results for each document type:

Scanned Documents

The precision is quite high, which means that the OCR is accurate when it identifies a value as a match, most identified matched are correct. However, the recall is somewhat lower, meaning that there are several matches that the OCR fails to detect.

Pictures of Documents

Here, the precision is moderately high, suggesting that when the OCR identifies matches in images of documents, they are likely to be correct. However, the recall is significantly lower, indicating that the system misses a lot of matches in picture-based documents. This might be because pictures can vary greatly in terms of lighting, angles, and clarity, affecting the OCR's ability to accurately capture text.

Electronic Documents

Electronic documents show the best performance in terms of precision and recall. Ideally, these results should be perfect given that electronic documents are generally free from many of the common issues that affect other document types during OCR processing. Nevertheless, there were instances of discrepancies: a near match where '27.33' was misread as '27:35', and a linked match where '13.71' was incorrectly identified as 'L341'. However, employing Pytesseract alone achieves 100% in both precision and recall. These observations underscore the potential benefits of adaptive processing techniques, which are further explored in subsection 4.2.3.

4.2.2 Limitations and Challenges

In order to understand the performance of each document type, let's dissect the challenges that hinder efficient processing.

Handwritten text

Handwritten text presents a recurrent challenge due to the variability in handwriting styles, cursive connections between letters and inconsistent spacing and alignment. Unlike typed text, which has uniform fonts and layouts, handwritten elements show significant diversity, even within the same page.

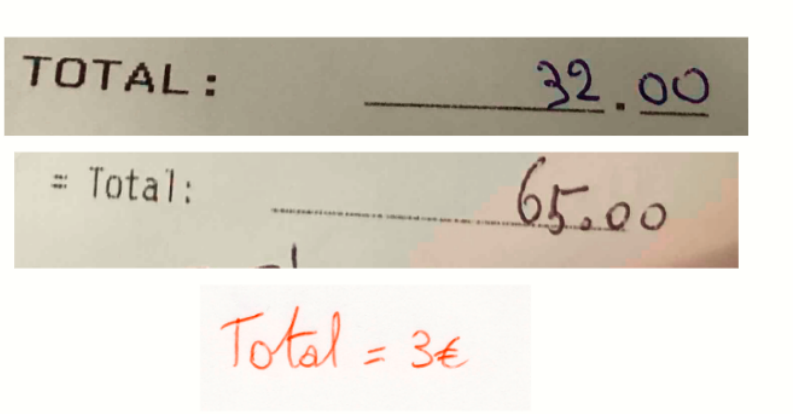

Figure 4.1: Examples of handwritten text that the OCR failed to capture

Preprocessing

Another significant hurdle lies in the absence of a universally effective preprocessing method. Every type of document goes through the same preprocessing pipeline, as described in section 3.2.3. Dilation aims to enhance the visibility of text by expanding the regions of high pixel values, which typically makes the characters appear bolder and more distinct. This morphological operation, while beneficial in many scenarios, does not uniformly enhance OCR performance across all types of documents. The variability in document characteristics (text density, font styles, and overall quality) influences the how effective this process is.

For instance, documents with dense layouts or elaborate font styles often present challenges when subjected to dilation. In this case, dilation may cause neighboring characters to merge, thereby creating connections that are not originally present. Moreover, documents with inherent blurriness or low resolution may not respond well to dilation either; rather than clarifying text, the process can make the blur worse, leading to poorer recognition rates. On the other hand, high-quality documents might see an improvement in character separability post-dilation. The effectiveness of dilation also depends on the contrast between the text and the background. Documents with significant background noise or uneven backgrounds can see an increase in noise post-dilation, further complicating text recognition.

Moreover, electronic documents, which typically require minimal preprocessing, are still processed in the same way due to the application's inability to differentiate between the three document types. Unfortunately, this means that preprocessing may do more harm than good for these documents.

Quality of pictures

Among the three document types, pictures exhibit the poorest performance due to several factors.

First of all, many documents are presented in crumpled forms. These folds introduce shadows and distorted text lines, leading to misalignment and occlusions of text characters. This makes it difficult for OCR algorithms to correctly identify and segment individual characters. In addition to that, lighting conditions are important for capturing high-quality images suitable for OCR. Underexposed images resulting from poor lighting can lead to low contrast between the text and the backgrounds, while overexposure can cause glare, which may wash out parts of the text. Uneven lighting creates shadows and highlights that can confuse the OCR, potentially mistaking them for text or obscuring actual text.

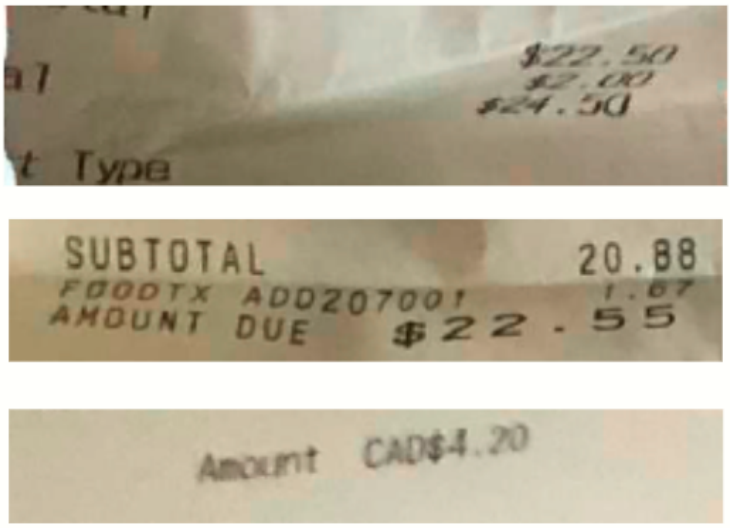

Figure 4.2: Examples of image documents with no matches

4.2.3 Possible Improvements

Optimizing the system's performance requires addressing several challenges. Let's outline potential improvements that could enhance OCR accuracy.

Adaptive Processing

One approach that could be promising is implementing a classifier that can automatically distinguish between scanned documents, pictures, and electronic documents. This classification would enable the application to apply tailored preprocessing strategies

optimized for each document type. For instance, scanned documents could require skew correction and noise reduction, while pictures would benefit from advanced lighting correction and perspective adjustment. On the other hand, electronic document wouldn't need any preprocessing.

Quality Control Before Processing

Before going through processing, we could implement some filters which would refuse pictures. For example, a blur detection method that could assess the blur at a micro-level, perhaps focusing on the clarity of individual characters or small groups of text, rather the whole image. Noise and clutter assessment could be beneficial.

Machine Learning

Using deep learning models, particularly CNNs, could significantly boost the OCR's ability to deal with complex document background and layouts. However, these models would need a diverse and large dataset that includes various types of noise, distortions and document qualities.

4.3 Comparison with Azure AI

While the core focus on this thesis has been the development and evaluation of the Docspotter package and application, it is essential to benchmark the results against established solutions in the industry. One such solution is Azure AI's OCR tool, which is part of Microsoft's Cognitive Services.

Azure AI OCR represents a mature, commercially available solution that leverages advanced machine learning techniques to offer robust text extraction capabilities. Although it is not a free service, its comprehensive feature set and scalability make it a valuable point of comparison. Unfortunately, the specific details of the architecture are not fully disclosed by Microsoft, as they are proprietary.

This section aims to contextualize the performance and features of the Docspotter application by comparing it with Azure AI OCR. We will explore the following aspects:

1. **Accuracy and Reliability:** How does the accuracy of text recognition in our application compare to that provided by Azure AI OCR across various document types ?
2. **Features and Capabilities:** Can Azure AI OCR support multiple languages, handwriting recognition and variable document quality ?

4.3.1 Performance by Document Type

Document Type	Nb of Exact Matches	Nb of Near Matches	Nb of Linked Matches	Nb of No Matches	Precision	Recall
Scanned	24	0	0	0	100%	100%
Pictures	23	0	0	1	100%	95.83%
Electronic	24	0	0	0	100%	100%

Table 4.3: Comparative effectiveness across document formats using Azure AI

	Docspotter		Azure AI	
	Precision	Recall	Precision	Recall
Scanned	93.42%	73.96%	100%	100%
Pictures	87.5 %	43.75%	100%	95.83%
Electronic	94.32 %	86.46%	100%	100%

Table 4.4: Comparison of OCR Performance: Docspotter vs Azure AI

Based on the table 4.3, we can conclude the following about Azure AI's performance across different document types:

Scanned Documents

Azure AI achieved perfect precision and recall. This means that every value the OCR identified was correct and it successfully recognized all values present in the scanned documents.

Pictures of Documents

The results for pictures are also impressive, particularely in terms of precision, suggesting that when the OCR system identified a match, it was always correct. The recall is slightly less than perfect due to one instance where the OCR failed to identify a value. This is due to the physical condition of the document that was severely folded, which obscured part of the document, as seen in figure 4.3. The OCR found *.4.3 instead of $24.50. This issue is more related to document preparation and handling than the OCR technology itself.

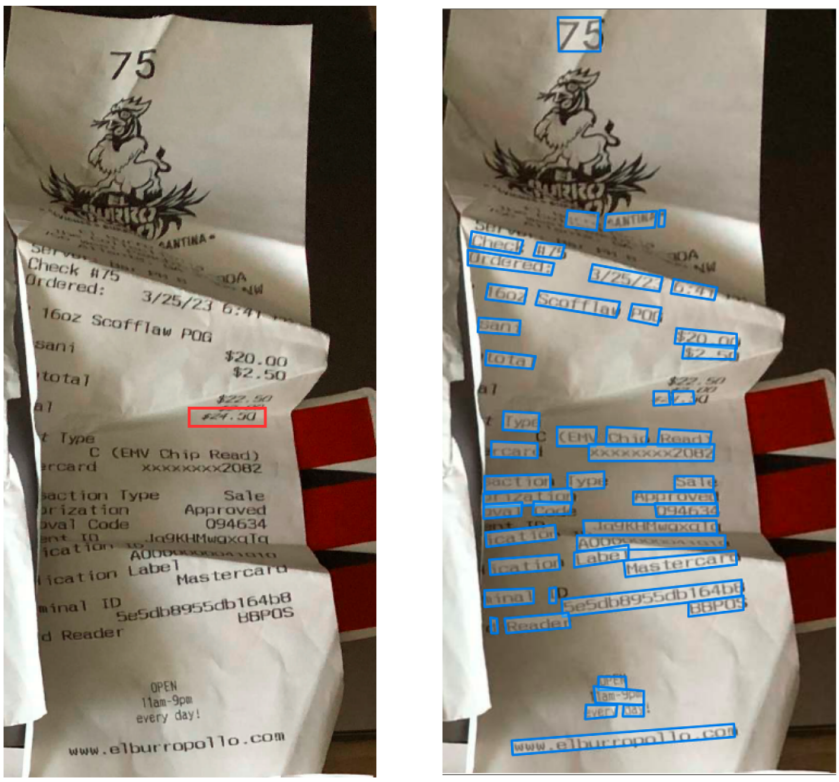

Figure 4.3: (Right) Value we are looking for. (Left) Values found by Azure AI

Electronic Documents

For electronic documents, we can also notice a perfect precision and recall. This is expected as electronic documents typically do not suffer from the same physical problems that can affect scanned documents and pictures, such as folds or creases.

4.4 Overall Assessment

Azure AI's OCR technology demonstrated excellent performance across all document types with perfect precision in every category and nearly perfect recall. The single "no match" instance emphasizes the importance of proper document preparation, including flattening and ensuring sufficient lighting, to maintain high accuracy, particularly when scanning physical documents.

4.5 Potential Enhancements for Docspotter

Given the performance gap highlighted in the comparisons in Table 4.4, several areas for enhancement in Docspotter become apparent. For example, incorporating more sophisticated machine learning models could bridge the performance gap. Azure AI also likely uses advanced preprocessing techniques to handle different types of documents, including those with poor quality or adverse conditions such as variable lighting and angles. This could also include adjusting the confidence thresholds dynamically based on the type of document being processed and employing more context-aware recognition strategies, making Docspotter more adaptable to various OCR challenges.

Conclusion

This dissertation addressed the need to improve the accuracy and efficiency of invoice processing at academic institutions like UCLouvain, where traditional manual methods are both error-prone and labor-intensive. It culminated in the development of a Python package, Docspotter, whose purpose is to detect and extract text from various document types. Docspotter leverages open source technologies to handle skewed text and diverse documents formats, in hopes of improving the efficiency of OCR processes and facilitate invoice processing. Additionally, a user-friendly application was created to demonstrate the capabilities of this package, allowing users to search and verify numerical values within scanned documents, images, and electronic files.

In the first part, which includes Chapters 1 and 2, we provided a comprehensive review of the foundational concepts in deep learning and OCR technologies. We discussed essential aspects such as neural networks, their architectures, training methods and the workflow of OCR systems. Additionally, this chapter covered key definitions related to invoices and the general principles of open-source projects, which were important for understanding the context and objectives of this work. This theoretical grounding was relevant for understanding the subsequent practical implementations.

The core contribution of this project, as detailed in Chapter 3, was the development of the Docspotter Python package. This package leverages CRAFT for text detection and Pytesseract for text extraction. The integration of these tools aimed to enhance the accuracy of OCR, particularly in dealing with skewed text and diverse document types. We also demonstrated the practical application of Docspotter through a GUI-based tool. This application enables users to search and verify numerical values in various invoice formats. The implementation details highlighted the use of text detection, skew-correction and preprocessing techniques, followed by text extraction, to handle all types of documents.

Chapter 4 evaluated the system's performance using custom precision and recall metrics across three documents types: scanned documents, pictures of documents, and electronic documents. The results indicated high precision across all types, with some variability in recall, particularly for picture-based documents, due to issues ranging from lighting and document quality, to an inadequate processing technique. Additionally, a

comparative analysis with Azure AI OCR, a commercially available OCR solution, revealed that while Docspotter performed well in terms of precision, Azure AI OCR exhibited superior performance overall. This comparison underscored the potential benefits of integrating advanced machine learning models and adaptive preprocessing techniques to further enhance OCR accuracy.

Throughout the dissertation, we identified several challenges, including the variability in document quality, the limitations of preprocessing techniques, and the difficulty in accurately recognizing handwritten text. To address these issues, potential improvements such as implementing adaptive preprocessing strategies, enhancing document quality control mechanisms, and exploring the use of advanced deep learning models were proposed.

The development of Docspotter represents a significant step towards automating personnel invoice verification. While it achieved substantial improvements in processing efficiency and accuracy, the comparison with Azure AI OCR highlighted areas for further enhancement. The proposed improvements offer a roadmap for future development, aiming to achieve even higher levels of performance and reliability.

In conclusion, this project contributes to the broader field of document processing by demonstrating the viability of open-source AI-OCR solutions in real-world applications. The findings and proposed enhancements provide a good foundation for future research and development, in hopes of further advancements in the automation of administrative tasks within academic and other institutional settings.

Bibliography

[1] What is open source? https://opensource.com/resources/what-open-source.

[2] What is free software? https://www.gnu.org/philosophy/free-sw.en.html.

[3] shubh_89. Difference between free software and open source software. https://geeksforgeeks.org/difference-between-free-software-and-open-source-software/.

[4] What is open source software? https://www.ibm.com/topics/open-source.

[5] Wikipedia contributors. Open-source license — Wikipedia, the free encyclopedia, 2024. [Online; accessed 1-June-2024].

[6] Open source licenses: Types and comparison. https://snyk.io/learn/open-source-licenses/.

[7] Open source licenses explained: A comparison. https://osssoftware.org/blog/open-source-licenses-explained-a-comparison/, 2024.

[8] Developer.Com Staff. 7 key benefits of open source software. https://www.developer.com/project-management/open-source-software-benefits/, 2017.

[9] Frank Amissah. Open source vs. commercial software license: What do you need? https://www.turing.com/blog/open-source-vs-commercial-software-license/, 2023.

[10] Adam Hayes. What is an invoice? its parts and why they are important. https://www.investopedia.com/terms/i/invoice.asp, 2023.

[11] Receipts and proofs of payment. https://www.law.berkeley.edu/business-services/purchasing-and-reimbursement/getting-reimbursed/receipts-and-proofs-of-payment/.

[12] What is expense reimbursement invoice and how to create one? https://www.reliabills.com/blog/expense-reimbursement-invoice/, 2021.

[13] Manual invoice processing vs. automated invoice processing: A comprehensive comparison. https://www.artsyltech.com/manual-invoice-processing-vs-automated-invoice-processing.

[14] What is deep learning? https://www.ibm.com/topics/deep-learning.

[15] Sadaf Saleem. Neural networks in 10mins. simply explained! https://medium.com/@sadafsaleem5815/neural-networks-in-10mins-simply-explained-9ec2ad9ea815, 2023.

[16] Manav Mandal. Introduction to convolutional neural networks (cnn). https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/, 2024.

[17] Chi Nhan Nguyen. Deep learning for information extraction. https://blogs.itemis.com/en/deep-learning-for-information-extraction, 2018.

[18] Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation. https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21, 2018.

[19] Forough Karandish. The comprehensive guide to optical character recognition (ocr). https://moov.ai/en/blog/optical-character-recognition-ocr.

[20] Sana'a Zahra. Ocr segmentation with python code. https://sanaazahra.medium.com/ocr-segmentation-with-python-code-f3251114ee48, 2023.

[21] Moniele Kunrath. Understanding ocr: Your (entertaining) guide to optical character recognition. https://medium.com/poatek/understanding-ocr-your-entertaining-guide-to-optical-character-recognition-5891d3
2023.

[22] Ocr – definition, benefits, challenges, and use cases. https://www.shaip.com/blog/ocr-definition-benefits-challenges-and-use-cases-infographic, 2022.

[23] What are e-docs? https://www.adobe.com/acrobat/business/resources/edoc.html.

[24] Bibek Thapa. How to extract texts from rotated(skewed) text-images using craft, opencv and pytesseract. https://medium.com/@bibekthapa.works/how-to-extract-texts-from-rotated-skewed-text-images-using-craft-opencv-and-pytes
2023.

[25] Nikita Saxena. Pytorch: Scene text detection and recognition by craft and a four-stage network. https://towardsdatascience.com/pytorch-scene-text-detection-and-recognition-by-craft-and-a-four-stage-network-ec 2020.

[26] Jasmin Kurtanović. Deep learning – ocr text detection and text recognition. https://serengetitech.com/tech/deep-learning-ocr-text-detection-and-text-recognition/, 2021.

[27] Youngmin Baek, Bado Lee, Dongyoon Han, Sangdoo Yun, and Hwalsuk Lee. Character region awareness for text detection, 2019.

[28] pawangfg. Vgg-16 cnn model. https://www.geeksforgeeks.org/vgg-16-cnn-model/, 2024.

[29] Pavly Salah. Ocr from concept to deployment part i. https://medium.com/optomatica/ocr-from-concept-to-deployment-part-i-c47156b8ade0, 2021.

[30] R. Smith. An overview of the tesseract ocr engine. volume 2, pages 629 – 633, 10 2007.

[31] sbatta. What are the different types of receipt options on the device? https://discuss.poynt.net/t/what-are-the-different-types-of-receipt-options-on-the-device/2908, 2019.

Appendix A

Docspotter App: User Manual

In the following section, we will explore the practical application using a using a receipt as seen in Figure A.1. This is a simple example of a restaurant receipt which provides a detailed summary of a transaction. With the receipt's layout that includes distinct separated sections and clear typography, it provides a good test case for the OCR.

Figure A.1: Example receipt [31]

Home Screen

The first screen, as seen in Figure A.2 is the home screen. It contains one field that needs to be filled, in which the user will put the numerical value they want to find in a file. The user can then choose a distance threshold, which represents Levenshtein distance[1], meaning that the application will show all results that respect the chosen distance.

The user can then choose which files need to be scanned. They can either choose folders or specific files. All types of files are handled, including '.jpg', '.jpeg', '.png', '.bmp' and '.webp'.

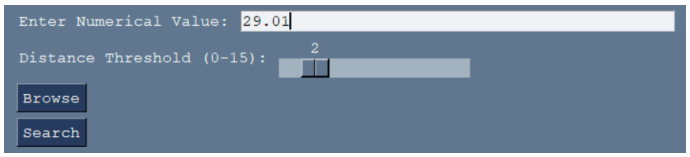

Figure A.2: Home Interface

Search Results

After clicking on the search button, a list of values that are closest to the initial value based on the distance threshold are displayed for the user, as seen in Figure A.3. It shows *"Value: {the value found} (Distance: {The Levenshtein distance it has with the initial word})"*.

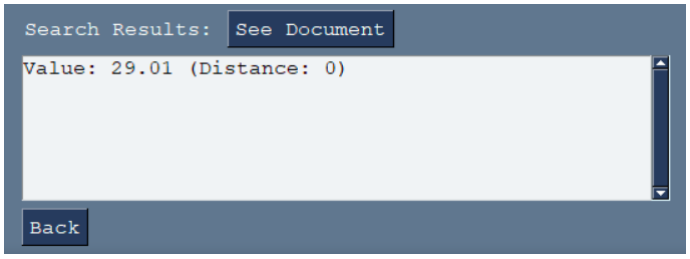

Figure A.3: Results interface

When a user clicks on a value from the search results, the image from which it has been taken is displayed, with a rectangle around the value so that it's easier to spot. The

[1]The Levenshtein distance, or edit distance, is a measure of the similarity between two strings. It calculates how many single-character edits (insertions, deletions, or subtitutions) are needed to change one word into the other.

user can then close the last two screens and search for other values from the same files, or redo the process with other files.

Figure A.4: Example receipt after finding the value

Appendix B

Evaluation Dataset Values

Scanned Documents

Values	Handwritten?	Found by Docspotter	Found by Azure
365,20	No	Exact match	Exact match
54,00	No	Exact match	Exact match
16,80	No	Exact match	Exact match
13,30	No	Exact match	Exact match
132.7	No	Exact match	Exact match
42	No	Exact match	Exact match
21.3	No	Exact match	Exact match
28,52	No	Exact match	Exact match
16,00	No	Exact match	Exact match
671,60	No	Exact match	Exact match
11,60	No	Exact match	Exact match
30.08	No	Exact match	Exact match
89.6	No	Exact match	Exact match
1068,40	No	Exact match	Exact match
16.8	No	Exact match	Exact match
12,80	No	Exact match	Exact match
68,60	No	Exact match	Exact match
4,30	No	Linked match	Exact match
469,00	No	Near match	Exact match
3	Yes	No match	Exact match
11,80	No	No match	Exact match
51,50	No	No match	Exact match
27,00	No	No match	Exact match
52,50	No	No match	Exact match

Pictures of Documents

Values	Handwritten?	Found	Found by Azure
116.88	No	Exact match	Exact match
7.89	No	Exact match	Exact match
29	No	Exact match	Exact match
18.5	No	Exact match	Exact match
14.63	No	Exact match	Exact match
16.02	No	Exact match	Exact match
16.22	No	Exact match	Exact match
551.73	No	Exact match	Exact match
52.00	No	Exact match	Exact match
298.64	No	Exact match	Exact match
28.84	No	Linked match	Exact match
28.84	No	Linked match	Exact match
6.65	No	No match	Exact match
64.98	No	No match	Exact match
4.2	No	No match	Exact match
32	Yes	No match	Exact match
65	Yes	No match	Exact match
34	No	No match	Exact match
22.55	No	No match	Exact match
30	No	No match	Exact match
25.04	No	No match	Exact match
24.5	No	No match	No match
9.9	No	No match	Exact match
4.7	No	No match	Exact match

Electronic Documents

Values	Handwritten?	Found	Found by Azure
7	No	Exact match	Exact match
886.29	No	Exact match	Exact match
1131.23	No	Exact match	Exact match
171.18	No	Exact match	Exact match
8.46	No	Exact match	Exact match
8.46	No	Exact match	Exact match
78	No	Exact match	Exact match
50	No	Exact match	Exact match
96.62	No	Exact match	Exact match
31.85	No	Exact match	Exact match
15	No	Exact match	Exact match
60.89	No	Exact match	Exact match
22.74	No	Exact match	Exact match
15.22	No	Exact match	Exact match
30.26	No	Exact match	Exact match
1,024.63	No	Exact match	Exact match
56.49	No	Exact match	Exact match
17.49	No	Exact match	Exact match
24.12	No	Exact match	Exact match
24.2	No	Exact match	Exact match
13.71	No	Linked match	Exact match
27.33	No	Near match	Exact match
1,101.75	No	No match	Exact match
610.58	No	No match	Exact match

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
École polytechnique de Louvain
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**