Development of an End-to-End Pipeline for Custom Key-Value Extraction from Commercial Invoices

by

Abhishek Mohan

S.B. Computer Science and Engineering
Massachusetts Institute of Technology, 2022

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2023

Author .....
Department of Electrical Engineering and Computer Science
January 18, 2023

Certified by .....
Amar Gupta
Research Scientist
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Development of an End-to-End Pipeline for Custom Key-Value Extraction from Commercial Invoices

by

Abhishek Mohan

Submitted to the Department of Electrical Engineering and Computer Science
on January 18, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Inefficiencies in manual extraction of information from business documents have resulted in the development of automated processing solutions. Within the scope of business documents, commercial invoices present additional complexities due to the diversity of document layouts and the variation in quality of scanned documents. Commercially available solutions have been built to perform invoice extraction, yet they do not provide flexibility in accomplishing tasks unique to a particular dataset and its associated complications. Using sample documents provided by a leading electronic component distributor, we researched different approaches capable of extracting key-value information from a complex dataset of invoices. The thesis provides a detailed look into the development of a highly accurate, end-to-end data pipeline accomplishing this task. A multi-module approach integrating image processing, optical character recognition, custom algorithms, and machine learning-based matching was built and compartmentalized into continuous stages - allowing for effective and efficient key-value extraction of information from invoice documents. In conjunction with an intuitive web interface, the custom pipeline provides a solution with strong performance and the flexibility to be generalized for extraction of additional business documents in future efforts.

Thesis Supervisor: Amar Gupta
Title: Research Scientist

Acknowledgments

First and foremost, I would like to thank my thesis advisor, Dr. Amar Gupta, for providing me with the opportunity to join his research group and work within such an interesting space. His support and mentorship throughout working on this project has been an invaluable part of my MEng experience, and I am looking forward to taking what I have learned from him into my future endeavors.

Next, I would like to thank all of the different student members from the Gupta Research Group who have supported the project in some capacity from the beginning: Pierce Lai, Steve Kim, Samuel Lee, Victor Chu, Prabhakar Kafle, and Haimoshri Dali. Their contributions all helped move the project forward, and I am very grateful for the experience of leading our project team.

I would also like to thank Justin Mintz and Bert Love, who were the main representatives from Arrow Electronics that my project team worked closely with. Their correspondence and cooperation ensured the rapid progress that was made on the project.

Last but not least, I would like to thank my family and friends, who have been a critical part in my successful completion of the MEng program. Their love and support continues to be a great source of my inspiration.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Document Processing Overview

A growing trend exists among companies to outsource parts of their manual business tasks, one of which is data extraction. Data extraction outsourcing is a multi-billion dollar industry and is only growing rapidly year to year, with estimations showing it to grow by $504 million into 2025 [1]. Data extraction specifically regarding business documents is a significant portion of this outsourcing. However, this comes at a price as significant human effort is still required, its manual nature does not improve speed, and many ethical issues around labor exploitation and data privacy remain. Thus, there has been a continued need for automating these processes to require minimal manual intervention [2, 3].

Over the past decade, there has been significant work in the development of automated data extraction pipelines [4, 5]. A significant portion of these research efforts have come, unsurprisingly, from the corporate world. Many major cloud service providers, including Google Cloud, AWS, Microsoft Azure, and Alibaba Cloud, have some form of service available that helps automate the document extraction pipeline. While these services work well for simple optical character recognition (OCR) and other general document extraction tasks, they do not allow much flexibility to fine-tune the pipeline for custom purposes.

Additionally, even though automation of document processing can lead to much

higher efficiency compared to traditional hand processing, many difficulties make automation challenging [6, 7]; this includes noise, varying locations of important information from document to document, and unclearly printed letters. Hence, there is an existing need for custom pipeline solutions to be built around particular datasets on an as-needed basis.

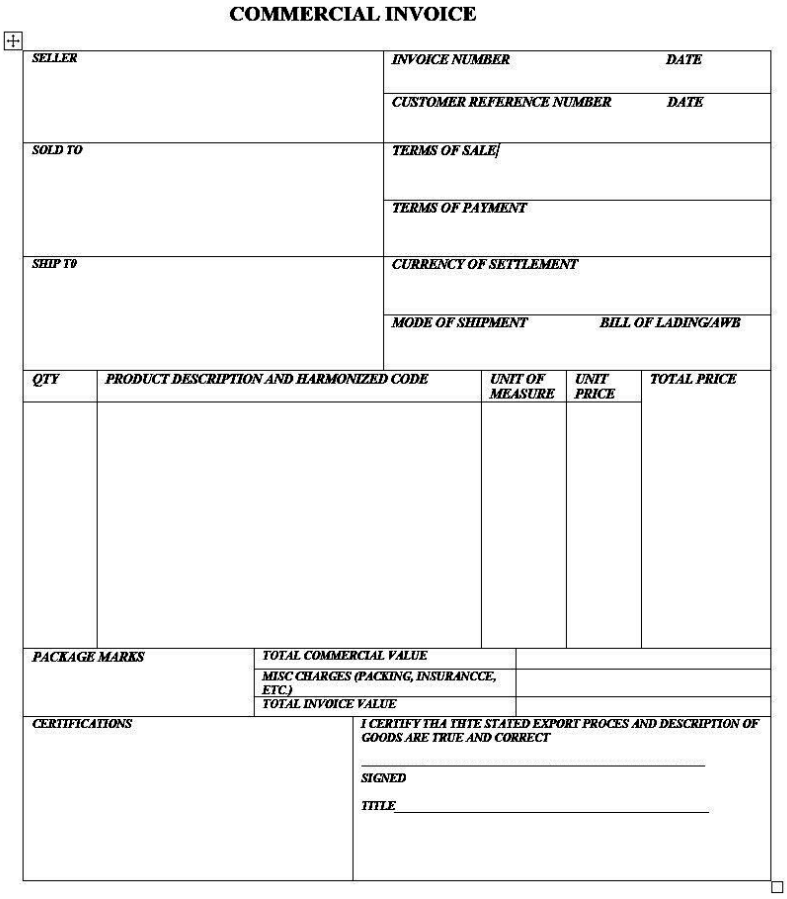
1.2 Invoice Documents Overview

Commercial invoice documents record the internal and external transactions made by a company, and are crucial to the company's proper functioning. An example of an empty commercial invoice document as provided by the International Trade Administration is shown in Figure 1-1. It is important to note that invoices are not required to follow this exact structure, and the company providing the invoice has full jurisdiction of its formatting. Extracting data from these documents is an important process, yet the extraction process for invoices often requires significant human effort and intervention, making it inefficient and expensive [8, 9]. Automatically understanding information from invoice documents is a challenging task additionally due to the diversity of invoice document layouts and the wide variation in quality of the scanned documents - presenting further complexities.

If the information a business hopes to extract differs from the standard information that commercial extraction services can extract, the business must develop a custom solution. For example, the AWS invoice analysis service known as Textract can extract items such as number, quantity, total, and company, but certain other important keys in work, like the waybill number, country of origin, and PO number, cannot be extracted [10]. Custom solutions allow for the elimination of such discrepancies, and ensure the solution being developed is able to achieve exact desired specifications.

Our lab has worked on problems of extracting important information from documents such as bank checks, IDs, and other miscellaneous documents [11, 12]. With the issue of invoice processing in mind, Arrow Electronics, a leading electronic component and computer product distributor, provided research funding to our lab for the

development of a pipeline that can process and extract key-value information from their provided collection of invoices; in other words, identify structured pairs from their unstructured data.



COMMERCIAL INVOICE

SELLER		*INVOICE NUMBER*		*DATE*
		CUSTOMER REFERENCE NUMBER		*DATE*
SOLD TO		*TERMS OF SALE]*		
		TERMS OF PAYMENT		
SHIP T0		*CURRENCY OF SETTLEMENT*		
		MODE OF SHIPMENT		*BILL OF LADING/AWB*

QTY	*PRODUCT DESCRIPTION AND HARMONIZED CODE*	*UNIT OF MEASURE*	*UNIT PRICE*	*TOTAL PRICE*

PACKAGE MARKS	*TOTAL COMMERCIAL VALUE*	
	MISC CHARGES (PACKING, INSURANCCE, ETC.)	
	TOTAL INVOICE VALUE	

CERTIFICATIONS	*I CERTIFY THA THTE STATED EXPORT PROCES AND DESCRIPTION OF GOODS ARE TRUE AND CORRECT*
	SIGNED
	TITLE

Figure 1-1: Example of an empty commercial invoice document as provided by the International Trade Administration.

1.3 A Need for Customized Extraction

The identified goal was to develop a multimodal pipeline that could efficiently parse through the Arrow Electronics invoice documents and convert them into a structured data output with minimal human effort. The specific problem at hand did not have a ready-made solution that could be easily applied, and so we aimed to build a custom solution that could accomplish tasks unique to the dataset and handle its associated complications.

Since many large companies handle many document types, the desired solution to extract relevant information would require compatibility with Arrow's diverse dataset. Most processing techniques require document customization and the algorithms themselves must be tuned for each document format [13]. In the domain of tables as found in invoices, while creating a system to extract key-value pairs from a specific format usually does not pose significant difficulty, creating a single approach that achieves a high accuracy on many different tabular layouts is quite challenging [14].

Despite the existence of commercially available pipelines to extract key-value based information, most approaches are usually tailored to a specific type of document or a certain document format [15]. Components such as the use of dark background colors and light foreground colors, as well as shading and background images, increase document complexity [16]. This further elevates the difficulty of using ready-made techniques to read, process, and match extracted information. The task of building a custom pipeline for key-value extraction, both efficiently and accurately, was accordingly shown to be one that would require deliberate preparation.

1.4 Identified Objectives

Having determined the project's overarching goal, we hoped to compartmentalize the pipeline into multiple modules such that each module would be responsible for a different objective, which together would accomplish the task of key-value extraction from the Arrow invoice documents. Below, planned objectives corresponding to the general focus of each module are presented; a detailed breakdown of each module is presented in following sections.

1. Create a preprocessing and OCR approach to read and extract information from commercial invoice documents

We intended to construct a module that would first prepare different kinds of technical documents for OCR. This preprocessing could involve techniques such as

binarization, thresholding, deskewing, denoising, etc. The challenge with this process is knowing which combination of techniques to use for each document, and the degree to which they should be applied. Then, an OCR engine for the pipeline would also need to be identified. The OCR component would allow for the extraction of texts and their relative locations within each of the documents.

2. Develop postprocessing methods to eliminate errors and further prepare extracted information for key-value matching

We intended to produce postprocessing methods which would serve as an additional step following preprocessing and OCR extraction. Complementing the previous module and handling errors that remain after it or other intermediate issues, postprocessing would aim to further correct the extracted texts before passing them on to the matching module.

3. Determine key-value pairs from extracted information by applying algorithmic and machine learning-driven approaches

We intended to develop an algorithmic and machine learning-based module that would allow for accurate matching of key-value pairs from the documents' extracted information. This would support the final step of determining which extracted words (values) matched with which of the specified keys, in turn accomplishing the goal of identifying the specified key-value pairs from the input invoice documents.

4. Build additional features to supplement the pipeline's usability

We intended to complete additional work to make the pipeline easily usable, primarily designing and creating a web interface that would allow users to run the pipeline on invoice documents from their local system. This work could also include developing a tabulation method that could convert the matched key-value results into

a format that can be visually identified on a processed invoice document, after which it could be displayed on the developed web interface.

Chapter 2

Related Work

Each of the following topics will be discussed in more detail throughout the thesis, but a variety of previous research studies and background information adjacent to the topics are presented here. By obtaining an initial understanding of these items, we are better prepared as to how the pipeline can be built, in addition to unique approaches that can be utilized.

2.1 Document-based Datasets

Document-based datasets are commonly recognized as being complex due to their inclusion of information such as text, figures, and tables [17]. The document dataset used for training and testing purposes in this project, consisting solely of Arrow-provided commercial invoice documents, would serve as the primary source of data for pipeline training, testing, and evaluation. Other studies and similar datasets can provide some insight into how the data might be structured and different potential ways of processing the input.

Receipts hold some resemblance to invoices due to their information also being able to be classified as key-value pairs [18]. With this, we looked at a dataset known as "Consolidated Receipt Dataset for Post-OCR Parsing" (CORD) that held some similarities to our dataset [19]. We noted that newer models such as DONUT and LayoutLMv3 produced results of over 90% in accuracy on CORD. This particular

dataset could help explore fine-tuning model(s) applied in the pipeline, with advanced models having shown effectiveness. Additionally, use of such a dataset could improve the selected model's ability to handle dynamic tabular data in the future - key-value pairs in which certain keys may not have been seen within the current Arrow dataset.

The "Scanned Receipts OCR and Information Extraction" (SROIE) dataset provides scanned receipts that are generally low quality [20]. It consists of six hundred receipts in the training set and four hundred receipts in the test set, with four possible keys: company, address, date, total. The dataset was used for three competition tasks in the study (text localization, OCR, key-information extraction), indicating it had good comparative quality to the Arrow dataset.

2.2 Image Processing

Unstructured documents in many cases lack a deterministic approach that would allow them to be accurately processed. Datasets with such documents have been been processed using different techniques with varying characteristics, as illustrated in Figure 2-1 [21, 22]. These documents are found in a wide variety of business-related activities, from which invoices are commonly seen. The Arrow dataset is one in which the input is entirely raw, and in many cases may not be fully compatible for key-value extraction. With the project's task to build a custom solution, specific processing approaches can be applied to improve the dataset's extractability.

Processing of document images plays a critical role in ensuring that the extracted information being passed into the key-value matching stage has been minimized for errors. Multiple libraries exist with already developed preprocessing methods that can be applied and evaluated to understand what configuration of the Arrow dataset is best for subsequent OCR extraction [23, 24]. Postprocessing of images has the ability to further reduce propagated errors as shown in previous studies [25, 26]. Its usage in our pipeline would ultimately depend on if there is an identified need for further correction upon OCR extraction.

OCR refers to the automated process of extracting information from printed or

written text within an image file or scanned document, after which the extracted material is converted into a machine-readable format [27]. The extracted results can be used for a variety of data processing applications, and is primarily used to improve efficiency of processing documents - eliminating the need for manual human entry.

We accordingly investigated some studies discussing previously applied OCR engines. Tesseract is an engine shown to take less than a second to extract information from an image [28]. On a dataset of license plate numbers, Tesseract had an accuracy of about 70% and performed better on grayscale compared to color images - important to note as the Arrow dataset primarily consisted of images with no color. Google Cloud Vision is another noteworthy engine that is mainly used for image classification, for which a previous study found that it is not robust to noise; adding random noise to images (e.g. about 20%, random colored dots) altered image classifications [29]. However, it is important to note that the type of noise added is not the same as the typical noise in the Arrow dataset, which contained less random distributed noise, more dots and random lines, and skewing.

Parameter	**Optical Character Recognition**	**Robotics Process Automation**	**Artificial Intelligence**
Ability to handle input data type	Works well on structured and semi-structured good quality scanned images.	Works well on structured data and some semi-structured data (Spreadsheets, RFID tags, GPS data.)	Works well on all data types, including unstructured data (word files, emails, images.)
Approach	Rule or Template-based.	Rule-based.	Learning-based : learns from data collected.
Processing approach	-	Deterministic.	Probabilistic.
Technology	Text detection and recognition.	Software robots configured to perform the repetitive task and complete routine.	AI-based on Deep Learning, Machine Learning, Computer Vision, Text Analytics, NLP.
Automation scope	Allows converting diverse categories of documents, like scanned documents, PDFs, or images, into editable and searchable data.	Minimize manual, repetitive, and rule-based task.	Can automate tasks that require decision making and increase the scope of automation.
Benefits	Editable and searchable data, which can be further utilized for storing in databases.	Business benefits are quick and immediate, returns can be realized quickly and cost-effectively.	Scalable and flexible.
Output	The key tool for digitizing documents.	Completed task or process, exception or alerts.	Structured output for the advanced integration.

Figure 2-1: Comparison of general image processing approaches.

2.3 Document Layout Models

When matching extracted words (values) to their corresponding keys, there are two approaches: the use of deterministic algorithms or the application of a model trained on a dataset [30]. Custom algorithms have shown applicability for text extraction purposes, and can be built for the pipeline based upon keys that we know exactly what to expect for [31, 32]. However, more advanced methods must be used for the majority of keys in the Arrow dataset. Machine learning models, specifically document layout models, provide the ability to match key-value pairs from a complex set of documents, and hence should also be incorporated into the pipeline [33].

LayoutLM is an example of a model that can jointly learn text with document layout information, and achieves state of the art results on multiple datasets such as the SROIE and FUNSD [33]. Built off of the BERT model, which uses text and position embeddings, LayoutLM integrates 2D position embeddings and image embeddings. The model outperforms other powerful models for objectives such as Masked Visual-Language Modeling (MVLM) and Multi-label Document Classification (MDC). Next generation models such as BROS and Docformer also indicate some potential for use within the pipeline [34, 35].

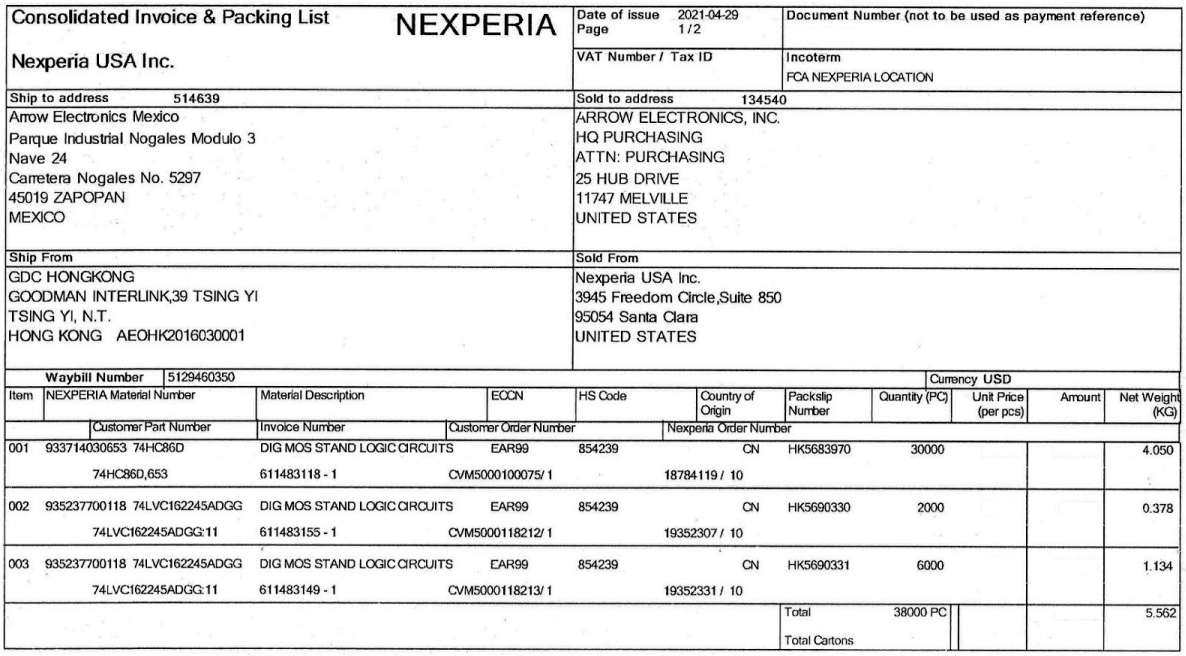
Some available solutions provide an existing end-to-end model that would eliminate the need for building individual modules for the pipeline. Many methods outsource the job of OCR to off-the-shelf engines, which can be costly, inflexible, and propagate OCR errors throughout the pipeline [36]. Models such as DONUT eliminate the need for this outsourcing while achieving strong results on many datasets such as CORD and DocVQA while also being faster - demonstrating potential in its use within the pipeline.

Chapter 3

General Methodology

3.1 Invoice Dataset

The dataset provided by Arrow Electronics contains image-based invoice documents (in PDF format) from various companies, as well as a number of spreadsheets containing the ground-truth data of those documents. This ground-truth data was used in the evaluation of individual modules and the comprehensive pipeline.



Consolidated Invoice & Packing List NEXPERIA

Nexperia USA Inc.

Date of issue 2021-04-29 Page 1/2

Document Number (not to be used as payment reference)

VAT Number / Tax ID

Incoterm: FCA NEXPERIA LOCATION

Ship to address 514639

Arrow Electronics Mexico
Parque Industrial Nogales Modulo 3
Nave 24
Carretera Nogales No. 5297
45019 ZAPOPAN
MEXICO

Sold to address 134540

ARROW ELECTRONICS, INC.
HQ PURCHASING
ATTN: PURCHASING
25 HUB DRIVE
11747 MELVILLE
UNITED STATES

Ship From

GDC HONGKONG
GOODMAN INTERLINK,39 TSING YI
TSING YI, N.T.
HONG KONG AEOHK2016030001

Sold From

Nexperia USA Inc.
3945 Freedom Circle,Suite 850
95054 Santa Clara
UNITED STATES

Waybill Number 5129460350 — Currency USD

Item	NEXPERIA Material Number	Material Description	ECCN	HS Code	Country of Origin	Packslip Number	Quantity (PC)	Unit Price (per pcs)	Amount	Net Weight (KG)
	Customer Part Number	Invoice Number	Customer Order Number		Nexperia Order Number					
001	933714030653 74HC86D	DIG MOS STAND LOGIC CIRCUITS	EAR99	854239	CN	HK5683970	30000			4.050
	74HC86D,653	611483118 - 1	CVM5000100075/ 1		18784119 / 10					
002	935237700118 74LVC162245ADGG	DIG MOS STAND LOGIC CIRCUITS	EAR99	854239	CN	HK5690330	2000			0.378
	74LVC162245ADGG:11	611483155 - 1	CVM5000118212/ 1		19352307 / 10					
003	935237700118 74LVC162245ADGG	DIG MOS STAND LOGIC CIRCUITS	EAR99	854239	CN	HK5690331	6000			1.134
	74LVC162245ADGG:11	611483149 - 1	CVM5000118213/ 1		19352331 / 10					
						Total	38000 PC			5.562
						Total Cartons				

Figure 3-1: Example of an invoice document from the Arrow Electronics dataset.

An example of a document is shown in Figure 3-1 (some information is redacted for confidentiality reasons). From the tabular format, visual assessment of the example can determine the presence of desired keys such as invoice date, company providing the invoice, and waybill number. The pipeline, in its final form, should be able to automatically and efficiently extract such keys and their corresponding values. Each of the documents includes a variety of possible keys, from which a comprehensive set of keys for extraction was identified:

1. **Waybill number** (identified as "GUIA")
2. **Invoice date** (identified as "FECHA")
3. **Shipping method** (identified as "SHIP_METHOD")
4. **Company for the invoice** (identified as "PDC")
5. **Part number** (identified as "NoPARTEARROW")
6. **Client part number** (identified as "NoPARTECLIENTE")
7. **Unit price** (identified as "PUNIT")
8. **Code** (identified as "FRACCION")
9. **PO number** (identified as "FACTURA")
10. **Country of origin** (identified as "PORIGEN")
11. **Quantity** (identified as "QTY")
12. **Total** (identified as "TOTAL")

Unlike standardized datasets such as the SROIE dataset, however, the Arrow dataset lacked quality control [20]. Hence, potential techniques to improve data quality for the pipeline's preprocessing component such as skewing and denoising were deemed necessary [42]. Additionally when provided to us, less than 1/3 of the initial dataset corresponded to any lines in the master ground-truth spreadsheet, and

of all the lines in the spreadsheets, only about 6.6% of them corresponded to any invoice PDFs. Since training via the pipeline would require being able to link invoice documents with ground-truth data, the data on which the pipeline could be trained on was only a small fraction of the full dataset.

To ensure that the dataset would be compatible with the pipeline, instances of error with respect to matching with the ground-truth were corrected with correspondence from Arrow Electronics. The corrected dataset consisted of ∼440 invoice documents from ∼50 distinct companies, corresponding to ∼2,200 lines within the ground-truth spreadsheet. This dataset continued to grow throughout the development process, with similar issues of some inconsistencies between the invoice documents and ground-truth spreadsheets being seen and corrected as the dataset grew. Ultimately, the comprehensive dataset used during the development process contained ∼18,000 identified key-value pairs.

3.2 Pipeline Architecture

There is a multitude of ways to develop a robust pipeline, with there being various stages and dependencies for it to properly function end-to-end. Previous pipeline developments provided guidance on how to go about deciding the structure of the pipeline, with the processing stage (including preprocessing and postprocessing) commonly being independent of stages involving other functionalities [37, 38, 39].

Using this, the pipeline consisted of two primary halves: the first half was responsible for loading the document images, preprocessing the images, extracting the words and their corresponding bounding boxes, and then postprocessing the extracted words (in other words, the preprocessing/OCR and postprocessing modules); the second half of the pipeline identified the words corresponding to each of the desired keys, which was accomplished by custom algorithms and a trained machine learning model, after which the output was converted into a proper table that could be easily read and understood (in other words, the key-value matching module and tabulation algorithm).

This structure is visually detailed within Figure 3-2, where each small box represents an internal component, script, or data file: black boxes represent data files; blue boxes represent regular document processing components; green boxes represent components associated with key-value matching; purple boxes represent pipeline evaluation scripts.

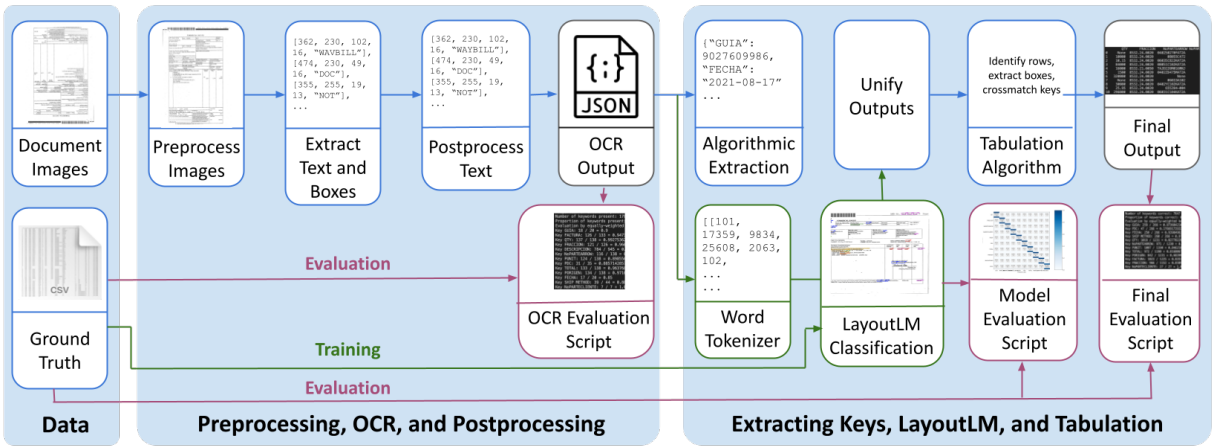


Figure 3-2: General architecture of the pipeline.

The architecture of the proposed pipeline is not fully linear. The second half of the pipeline containing the machine learning and algorithmic extraction components form two distinct branches, after which their outputs are combined. This ensured that the pipeline would not have any circular dependencies, and hence no cycles would form and cause disruption - a challenge found with managing data pipelines [41].

3.3 Web Application Interface

A web application that allows users to run the extraction process from end-to-end was also built to accompany the pipeline, meaning users could select invoice documents from their local system for key-value pair extraction. By providing an easy-to-use and easy-to-understand user interface, users would have simple use of the pipeline for their key-value extraction purposes. Figure 3-3 provides an example of what the interface displayed after a document was run through the pipeline, with its bounding boxes and corresponding keys identified on the document itself on the left and the

extracted key-value pairs presented on the right (some information is redacted for confidentiality reasons). Some of the UI's features were:

- The web application could run both halves of the pipeline on one or multiple input PDF(s)
- Just as a progress bar can be seen for progress within a terminal, we integrated a progress bar into the web application so users could be aware of approximately how close the pipeline was to being completed for its run
- Hotkeys provided a shortcut to different options/actions within the pipeline for the user's ease of use (i.e. save output/images)

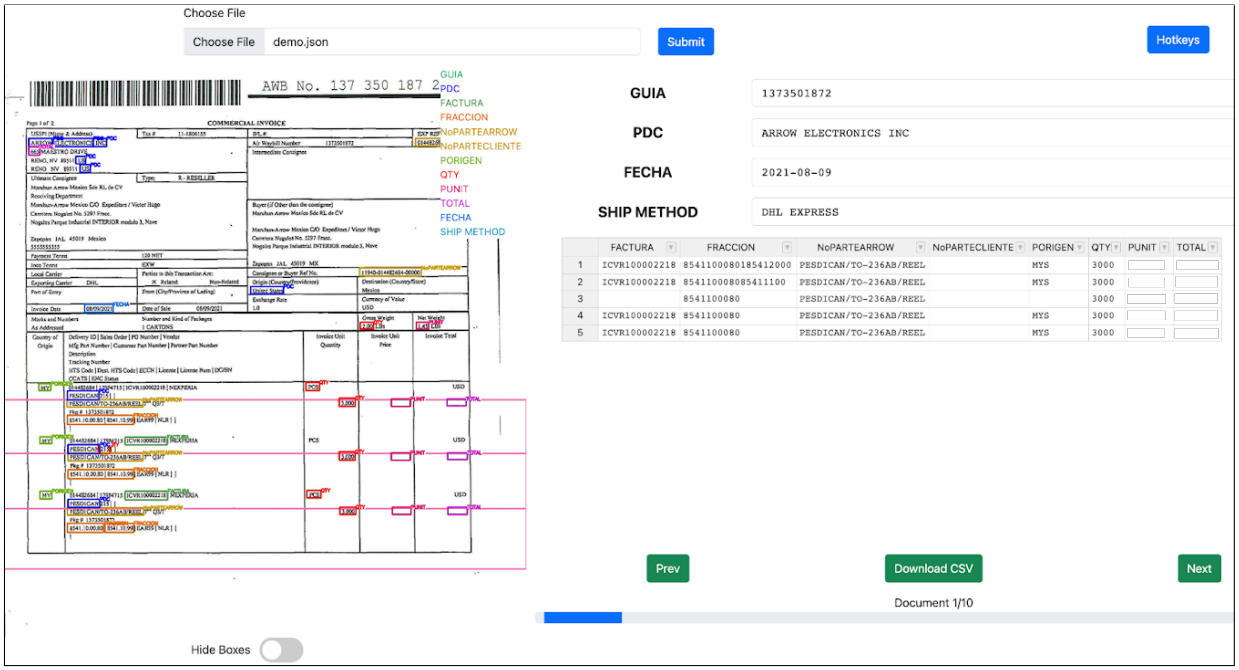


Figure 3-3: General outline of the pipeline's web application.

3.4 Internal Dependencies

Generalizing the current pipeline so that it could be used for more key-value extraction purposes in the future would allow for new keys to be extracted. This would in turn

allow us to automatically extract key-value pairs from commercial documents other than specifically invoices.

To potentially allow such changes in the future, we broke down the pipeline's internal dependency structure by developing architecture diagrams and an extensive documentation report. This report is not included in the thesis, but essentially provides internal implementation details of the many individual modules, scripts, and files forming the pipeline. Figures 3-4, 3-5, and 3-6 are diagrams illustrating the internal dependency relationships responsible for image processing, machine learning-related components, and the web application. Using these diagrams, we can easily and visually identify which file/component to work on to add new features as desired in the future, and they simply provide greater clarity as to what the internal dependencies of the pipeline look like.

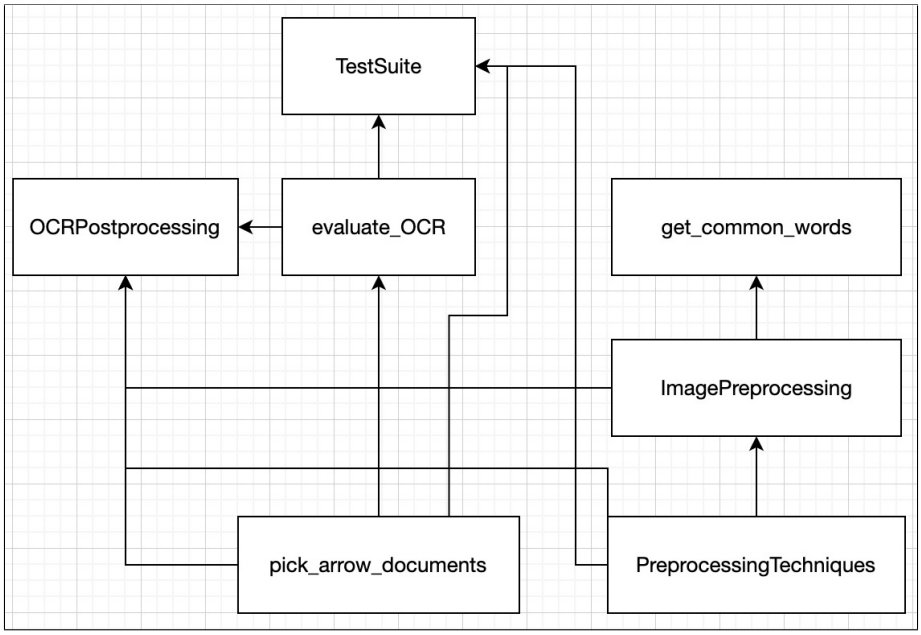


Figure 3-4: Diagram illustrating the dependencies of internal files responsible for the image processing of invoice documents.

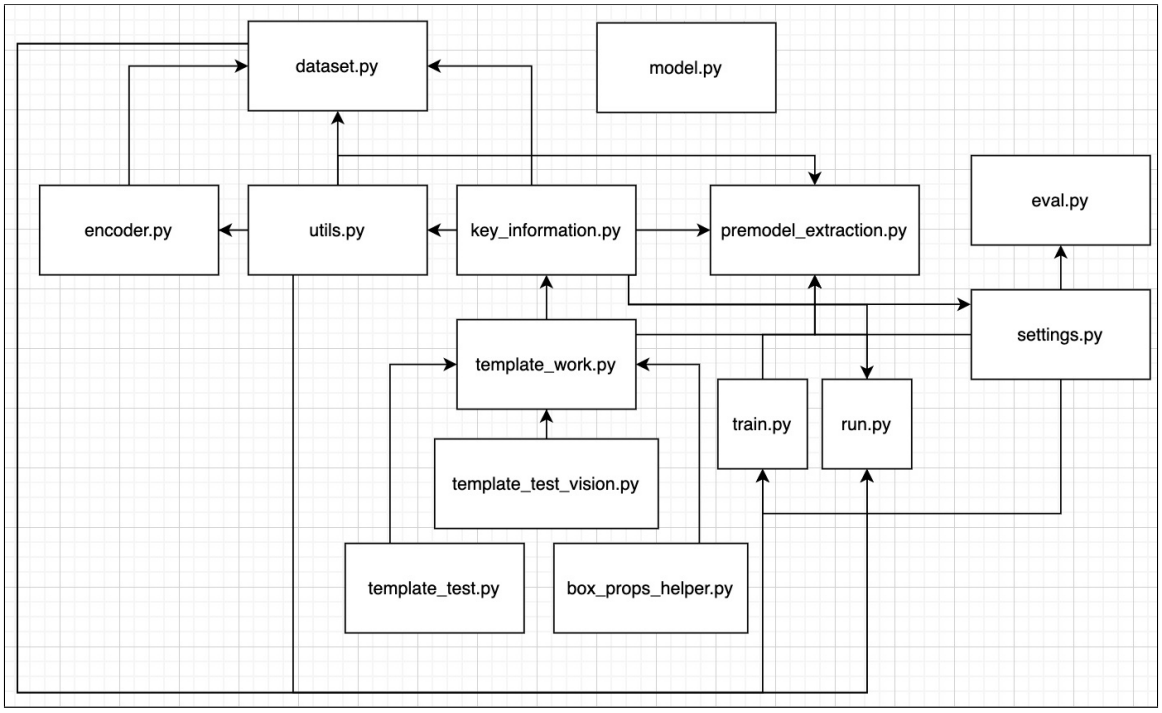


Figure 3-5: Diagram illustrating the dependencies of internal files responsible for the machine learning components.

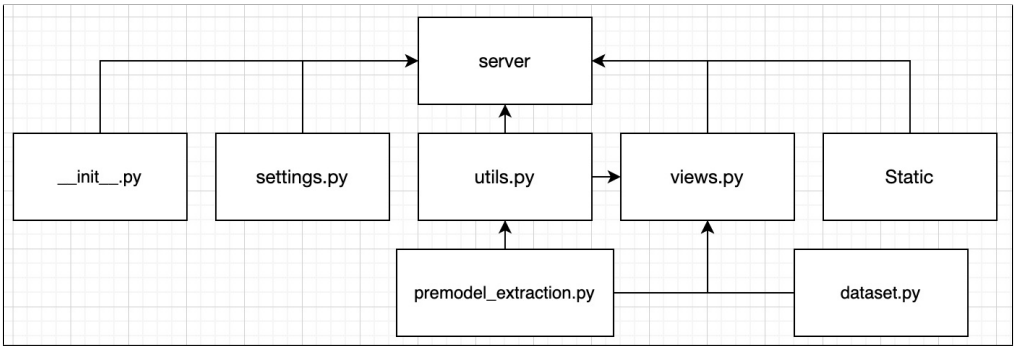


Figure 3-6: Diagram illustrating the dependencies of internal files responsible for the web application.

Chapter 4

Preprocessing and OCR Module

4.1 Preprocessing Overview

Preprocessing serves as a common step in OCR applications as scanned documents tend to contain a variety of flaws that limit OCR accuracy, such as noise, skewed pages, watermarks, and degraded text [42]. Functions provided by OpenCV, a popular Python computer vision library, were utilized to develop a suite of potential preprocessing techniques to include in the module: resize, binarize, denoise, sharpen, dilate, and deskew [43].

- **Resize:** Rescales an image, as images with a low DPI (dots per inch) tend to result in decreased readability
- **Binarize:** Converts an image to consist of only black and white pixels, increasing contrast within images and allowing for text to better stand out from the background
- **Denoise:** Removes minor pepper noise (specks and small impurities from the scanning process)
- **Sharpen:** Sharpens edges and text within an image, which allows for text to better stand out

- **Dilate:** Increases the area taken up by elements within an image, which can help fill in degraded or missing parts of individual characters
- **Deskew:** Fixes an image's orientation/skew

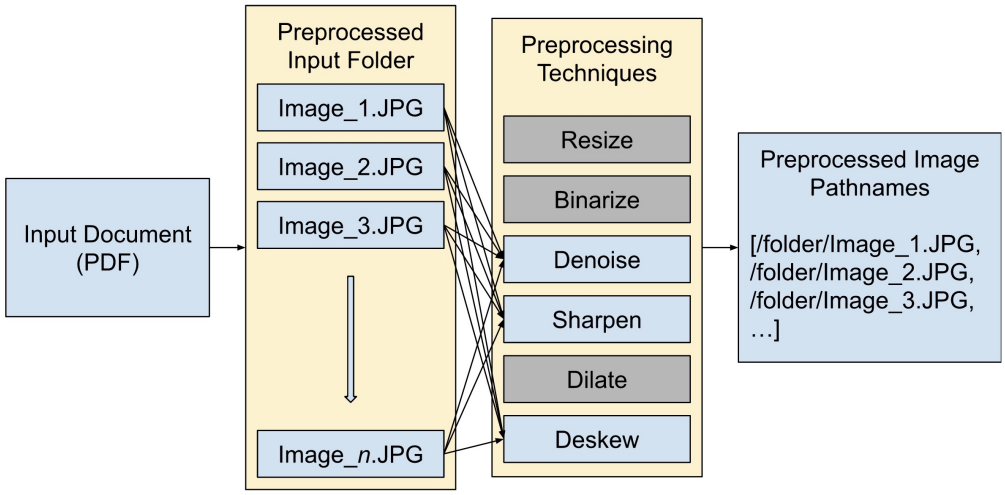


Figure 4-1: Overview of the Preprocessing component.

4.2 OCR Overview

After applying preprocessing techniques to the invoice documents, the next step in the module included utilizing an OCR engine to extract and output texts from the documents. In general, upon calling an OCR engine's API, the response returns the detected words stored as description keys and their location on the image - what we considered as the text's "bounding box." An example of bounding boxes within an input document is shown in Figure 4-2 [44]. Each of the texts includes a green rectangle enclosing it, with four corners corresponding to coordinates being returned. This bounding box is essentially four pairs of (x, y) pixel values that correspond to the corners of the tightest "box" in the image encompassing each of the texts, also known as position embeddings, which could be used within later steps of the pipeline.

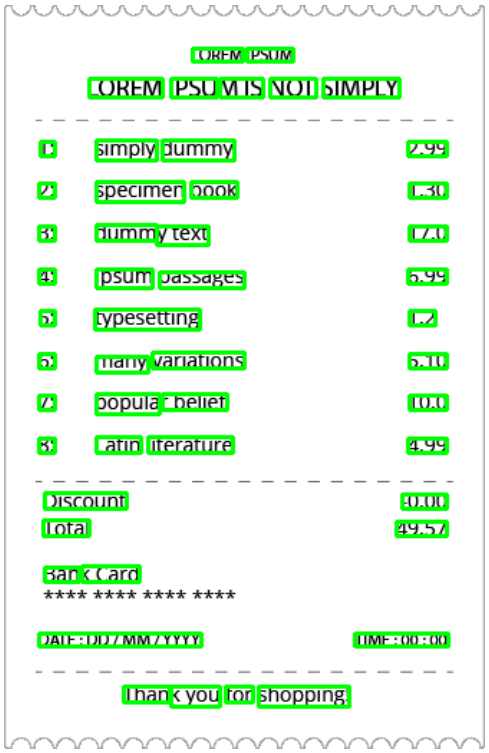


Figure 4-2: Example of bounding boxes returned from an OCR API.

4.3 OCR Engine Selection

Much work in this field has used Tesseract OCR as the primary OCR engine as is illustrated in a previous research study [45]. The module was initially constructed with Tesseract as the OCR engine through the Python wrapper Pytesseract, but Google Cloud Vision (GCV) was soon considered as an option for the pipeline's OCR engine - with previous studies indicating GCV as a powerful alternative [46, 47].

GCV OCR is part of the Google Cloud Vision API, and could be used for its "Document Text Annotation" service to read and parse text within the invoice documents [44]. With the invoice documents having an unstructured format (a lack of specificity as to how exactly each document looked), GCV would allow for the identification and extraction of texts regardless of where they might be on the document and what they might look like. Most, if not all, of the text within the documents was expected not to have any extra type of formatting (such as being bolded or italicized), which would allow for effective use of GCV.

Hence, we decided to test between our two considered options. Pytesseract was quite slow when we ran it on a large number of documents, as shown in Figure 4-3. Even with multithreading, it took ∼60 minutes to run Pytesseract on a subset of ∼230 invoice PDFs. Running the same dataset using GCV's service with multithreading took ∼9 minutes, providing a ∼6x increase in speed.

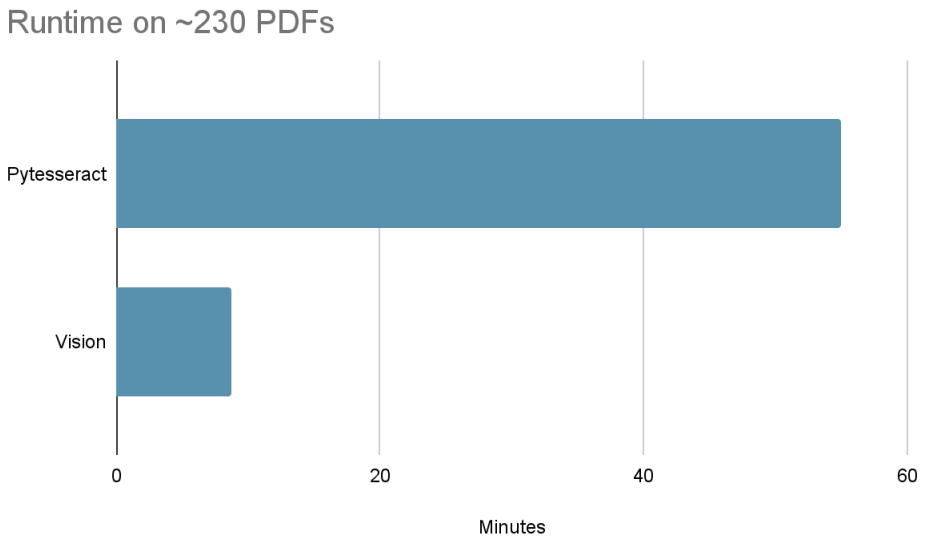


Figure 4-3: Runtime comparison between Pytesseract and GCV OCR engines.

The original preprocessing module with Pytesseract was constructed with three of the six potential preprocessing techniques: denoising (removing impurities from an image), sharpening (sharpening an image's edges to improve "reading" of text), and deskewing (fixing an image's orientation); other preprocessing methods were also tested, such as binarization and dilation, but this set of three worked best in the pipeline. GCV was much more robust than Pytesseract meaning that some of these preprocessing techniques could be removed, simplifying the preprocessing step.

With GCV, the pipeline's preprocessing module consisted of only the deskewing technique, as denoising and sharpening did not improve GCV's overall performance and would hence be unnecessary. Deskewing, however, remained necessary, as the provided documents varied in orientation and skew. Additionally, we found that GCV was overall more accurate than Pytesseract for our dataset, as it detected more

words and made fewer mistakes. Based on these observations, and it being recognized as one of the most accurate and robust options available, GCV was selected as the pipeline's OCR engine [48].

With regards to the use of multithreading within the pipeline, integration of the Python package "multiprocessing" into the pipeline improved performance. This package allowed the pipeline to utilize multiple cores and improved processing speed by $\sim$5x (depending on the number of cores used). As of now only the first half of the pipeline is multithreaded, while the second half is not.

Chapter 5

Postprocessing Module

5.1 General Overview

Postprocessing allows for the improvement of text information extracted from OCR, and is accomplished via combinations of automated and/or manual techniques [49, 50]. Specific errors can be identified and corrections be made, which in our pipeline ensured that the extracted text outputs were in a proper, revised format for use in the subsequent Key-Value Matching Module.

5.2 Levenshtein Distance Overview

A commonly used postprocessing technique is to utilize Levenshtein distance, a metric that enumerates how different two words are [51]. The standard Levenshtein distance between two words is the number of character edits required to convert one word to another, including insertions, deletions, and replacements. For example, the Levenshtein distance between "stone" and "tin" is three: two removals ("s" and "e") and one replacement ("o" to "i"). The recursive Levenshtein distance equation is shown in Figure 5-1, where "a" is a word, "b" is a word, "i" is the terminal character position of "a," and "j" is the terminal character position of "b."

We implemented a semi-automatic Levenshtein distance-based method in conjunction with a ground truth dictionary, or word bank. With inspiration taken from a

$$\mathrm{lev}_{a,b}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} \mathrm{lev}_{a,b}(i-1,j) + 1 \\ \mathrm{lev}_{a,b}(i,j-1) + 1 \\ \mathrm{lev}_{a,b}(i-1,j-1) + 1_{(a_i \neq b_j)} \end{cases} & \text{otherwise.} \end{cases}$$

Figure 5-1: Complete equation for calculating the true Levenshtein distance.

previous work, this method accepted input words which were each then compared to their closest neighbor within the word bank [52]. The method would swap a word with its word bank counterpart if its calculated Levenshtein distance was below a set threshold. This was due to the fact that the words of interest in the dataset consisted of more than just words within the English lexicon, as commercial invoices could also contain foreign and industry-specific words. Levenshtein distance does not discriminate between English and non-English words, and hence was generalizable for our objectives.

5.3 Levenshtein Distance Customization

We included several customized features so that the extracted texts could better fit the Key-Value Matching module. First, we did not consider any extracted words containing numbers for postprocessing. This was due to the fact that any text containing numbers was likely to represent some sort of value or identification number, which was usually unique to the product and accordingly should not have been fixed.

Next, the module utilized custom weights when calculating a pair's Levenshtein distance to allow for improved word comparisons [53]. The default Levenshtein distance algorithm treats all single-character edits the same, meaning that swaps between pairs of characters similar in appearance (1, I) are treated the same as pairs that are very different (X, 0). However, as the OCR engine was much more likely to incorrectly identify characters similar in appearance, such instances needed to be given a lower distance, which could also be considered as a weight. Because of this, we used the Python package "weighted-levenshtein," which had a set of custom weights for insertions, deletions, and character substitutions [54]. Given the need for custom

weights and with inspiration from the results of a prior study, the distance thresholds for "fixing" words were accordingly set as follows: 0.99 for words of length 3 or less, 1.99 for length 4 to 8, and 2.99 for all other word lengths [55].

5.4 Word Bank Construction

As our Levenshtein distance approach required a bank of words to be compared to, we implemented a self-growing word bank with the ability to adapt and grow with new input documents. Due to the sheer number of invoices processed by Arrow Electronics, manually constructing a list of relevant words would be inefficient. Hence, we initialized the word bank to contain a small subset of crucial keywords that would be used later in the Key-Value Matching Module.

Each time a document was processed through the pipeline for its first time, if the word bank construction feature was enabled, the module would record the OCR word outputs that had a confidence score greater than a defined threshold. After a word appeared a set number of times with a high enough score, the module would add the word to the word bank, which would be saved across runs to a specified JSON output file. We settled upon a confidence of 0.95 with a count threshold of 5, which ultimately allowed for the word bank to be adaptable and effective in its purpose.

5.5 Additional Methods

Besides using the word bank, additional custom postprocessing methods specific to this dataset were implemented. Some of these methods include:

- To remove redundant inputs to the word bank, we checked if a word did not include numbers and stripped special characters
- Dates were converted into the ISO format, which would make dates easier to detect (for example, "July 17, 2021" would become "2021-07-17")

- Unit prices would often have "/M" attached to it, which meant that the unit price was for a thousand units; the module would detect this and divide the price by 1000 to identify the correct price per unit
- "FACTURA" numbers occasionally had "-1" or "/1" attached at the end, which would be removed
- To maintain consistency, the country of origin would be converted into predefined three character codes, such as "TAIWAN" to "TWN" or "KOREA" to "KOR"
- PO numbers often started with "CVM5," so GCV would read it in as "CVMS" instead - this would be corrected
- The given list of bounding boxes would sometimes be modified in-place to combine words surrounding hyphens and slashes
- Two "PORIGEN" codes that were separated by a slash would be identified as independent from one another

Chapter 6

Key-Value Matching Module

6.1 General Overview

After passing through the Postprocessing Module, the pipeline produced a JSON file containing the words and bounding boxes for all pages in the input. Now, the pipeline's objective was to identify which words (values) corresponded to which of the specified keys. For some of the keys, custom algorithms were used as a deterministic approach could be followed. For the remaining keys, the pipeline used a trained machine learning model to classify the given words and bounding boxes. An example of some matched key-value pairs can be seen in Figure 6-1, where classifications from the module are color coded and written adjacent to each bounding box (some information is redacted for confidentiality reasons).

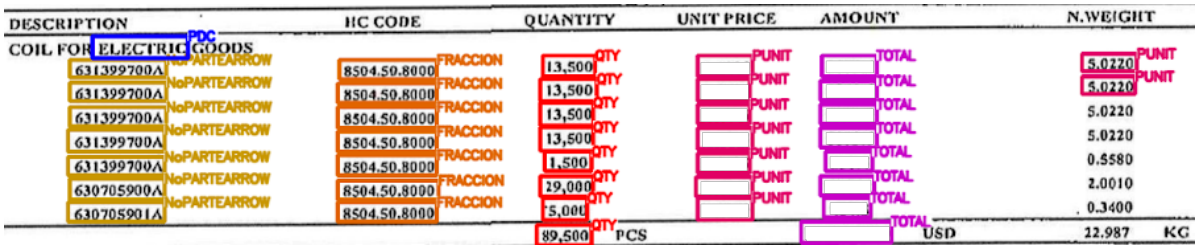


Figure 6-1: Example of key-value pairs following results from the Key-Value Matching Module.

6.2 Deterministic Algorithms Overview

We identified that the keys corresponding to waybill number, invoice date, and shipping method could be determined following a specific structure for every document. This deterministic quality made them less suitable to be extracted by a ML model, and hence we developed and used custom algorithms for them. Specifically, the waybill number was often parsed as multiple words, and so we used a rule-checking method to determine if the extracted value corresponded to a possible waybill number; the date was already preprocessed into the ISO format, and so a simple format comparison was needed; the shipping method was always from the company DHL (either "DHL EXPRESS" or "DHL 3RD PARTY INT'L BILL") and hence was also easy to extract.

6.3 Machine Learning Model Overview

The specific machine learning model that we chose to use within the module was LayoutLM, which is a state-of-the-art model used for interpreting complex document layouts. Specifically, the pipeline used the LayoutLMForTokenClassification version of the model, which classifies words and bounding boxes based upon the specified keys. For the model, we knew that we wanted to integrate a model whose foundation was built upon the problem of document processing - essentially identifying a model that could label the documents' texts accurately. For the training loop, we used Cross Entropy Loss with the AdamW optimizer, and built a custom data preparation method that allowed us to train and test on a variety of custom subsets of the entire dataset. A diagram of the ML model component's general structure is shown in Figure 6-2.

LayoutLM was inspired by the BERT model, which uses an attention-based bidirectional language modeling approach [56]. BERT accepts a sequence of tokens and stacks multiple layers to produce final representations. More specifically, using a set of tokens that have been processed, the input embeddings are computed by summing the corresponding word, position, and segment embeddings together. These input em-

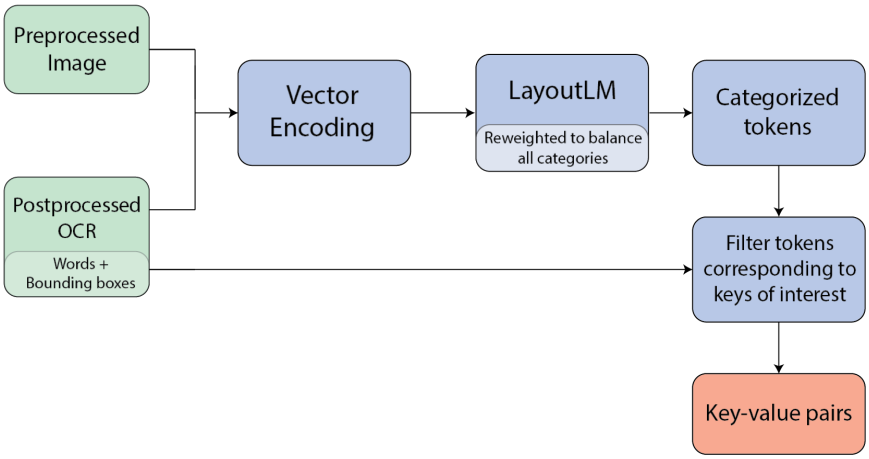


Figure 6-2: Diagram of the pipeline's ML Model component.

beddings are then passed through a multi-layer bidirectional transformer that is able to generate contextualized representations with an adaptive attention mechanism.

Document layouts contain visually rich information that can also be aligned with input texts, and this ideology serves as the foundation of the LayoutLM model. There is document layout information that contains the relative position of words within the invoice documents, which can be embedded as 2-D position representations. There is also visual information that primarily contains indications of which document segments are important and should accordingly be prioritized, which can be represented as image features. Thus, combining these two types of information allows for a more nuanced semantic representation of a document [57].

LayoutLM does exactly this by applying the BERT architecture and adding two additional input embeddings: a 2-D text position embedding and an image embedding. The 2-D position embedding is a way through which the relative spatial position in a document can be represented. The spatial position of elements (via bounding boxes) is represented by (x0, y0, x1, y1), where (x0, y0) corresponds to the position of the bounding box's upper left corner and (x1, y1) corresponds to the position of the bounding box's lower right corner. For the image embedding, with each of the word's bounding boxes from the OCR results, the image is split into several pieces, all of which have a one-to-one correspondence with the words. These image region features are then converted into token image embeddings. As shown in Figure 6-3, the

downstream task (in our case, key-value matching) is accomplished upon combining the image and LayoutLM embeddings after passing through the model.

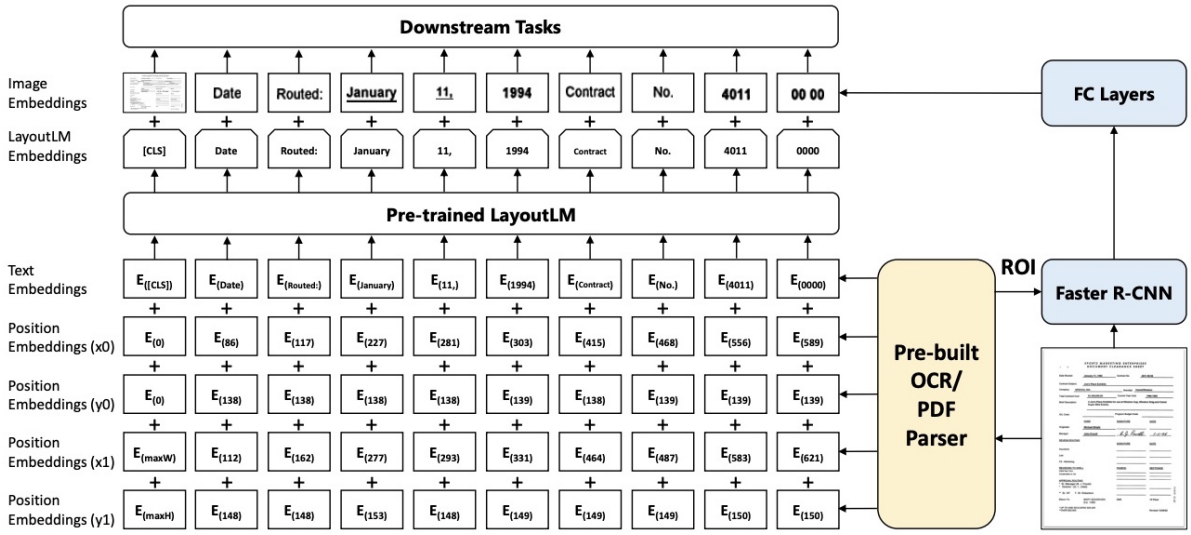


Figure 6-3: Architecture of the LayoutLM model.

A powerful component of the LayoutLM model was its independence from preprocessing, OCR, and postprocessing methods, meaning that its sole purpose was to identify the key-value pairs in the pipeline. Separating the first two modules from the third module containing the model in the pipeline, LayoutLM accordingly allowed us to more easily identify and remedy errors if they appeared earlier on - improving the pipeline's final results.

6.4 Machine Learning Model Alternatives

We also considered other models for the machine learning component, primarily the DONUT model, which stands for "Document Understanding Transformer" [36]. DONUT had been utilized in the development of other datasets and extraction methods similar to the key-value matching task we had been working on [58, 59]. This model's architecture uses a visual transformer encoder and textual decoder, and unlike most other models that use OCR first and then apply a machine learning approach such as LayoutLM, DONUT does not use any off-the-shelf OCR package. In other

words, it is an end-to-end model that handles the entire process of taking in processed input images and matching the key-value pairs.

The use of the DONUT model was appealing for several reasons. Unlike other models, we would not need to externally identify potentially effective OCR engines, errors from the OCR component would not propagate through the rest of the model, and no OCR postprocessing module would be needed. This in theory could allow the pipeline to be simpler and faster while also attaining a higher accuracy. As further shown in Figure 6-4, DONUT provides a full system with no outsourcing of processing approaches or OCR engine, allowing for focus on the objective of key-value extraction from a provided document [36].

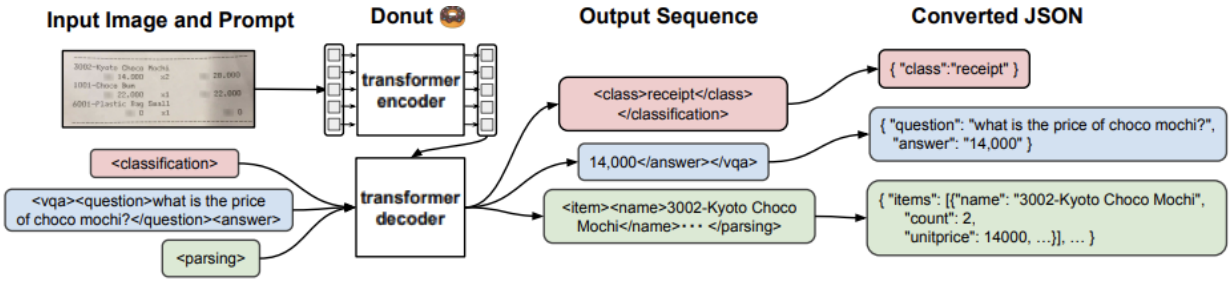


Figure 6-4: Architecture of the DONUT model.

To obtain a base understanding of the DONUT model, we trained and tested using the SROIE dataset. After 30 epochs of training, the results were slightly accurate with an accuracy score of 0.679 and an F1 score of 0.574. Next, to evaluate if integrating the DONUT model would be a better approach, we trained the model with the Arrow dataset, first for 10 epochs to verify proper training, and then further trained it for another 20 epochs, for a total of 30 epochs.

266 invoice documents from the Arrow dataset were used, on which DONUT did not perform well. Both the summarized and full accuracy results are shown in Tables 6.1 and 6.2. Deterministically matched keys such as "GUIA" and "SHIP_METHOD" had high accuracies as expected, but all other keys involved within the ML component greatly underperformed. We did note that the model learned the output representation extremely quickly, within 10 epochs. However, even after 30 epochs, the information output did ultimately not match with the ground-truth, and did not

show much improvement with additional epochs.

# of Correct Keys	# of Total Keys	Total Accuracy	Equally-weighted Accuracy
3,599	9,540	37.72%	50.50%

Table 6.1: Summarized results of the DONUT model on the Arrow dataset.

Keys	# of Correct Keys	# of Total Keys	Key Accuracy
GUIA	230	266	86.47%
FECHA	210	266	78.94%
SHIP_METHOD	242	266	90.07%
PDC	208	266	78.20%
NoPARTEARROW	338	1230	27.48%
NoPARTECLIENTE	13	27	48.15%
PUNIT	363	1190	30.50%
FRACCION	375	1152	32.55%
FACTURA	315	1225	25.71%
PORIGEN	498	1231	40.45%
QTY	447	1231	36.31%
TOTAL	360	1190	30.25%

Table 6.2: Individual results of the DONUT model on the Arrow dataset.

We soon realized that the results were poor not because the model was inferior, but because DONUT expected an input image of size 1280 x 720, while images of size such as (5 * 1280) x 720 were being passed in, meaning the image had to be compressed vertically or in another manner. In order to remedy this, we developed a script that modified the dataset such that the images were rotated correctly (as needed) and then were made as large as possible, instead of shrinking them down to 12 pages - making the 88% of documents with 6 or fewer pages more legible.

This, however, came with several caveats. The features in each document now varied considerably in size, and as we would need to apply the same process for future datasets, GCV would also be required as a preprocessing step, increasing cost and processing time. Converting the Arrow dataset into a compatible format and testing the DONUT model in the pipeline produced an output with accuracies that

were much lower than expected. As this model presented many inefficiencies and difficulties, we ultimately decided not to move forward with fully integrating the DONUT model into the pipeline.

The LayoutLMv2 and LayoutLMv3 models were also considered, which would provide further improved versions of the LayoutLM model [60, 61]. However, these new versions were not allowed to be used for commercial uses (a stipulation of the project), and hence LayoutLM was deemed as the final choice for the machine learning component.

Chapter 7

Document Tabulation

7.1 General Overview

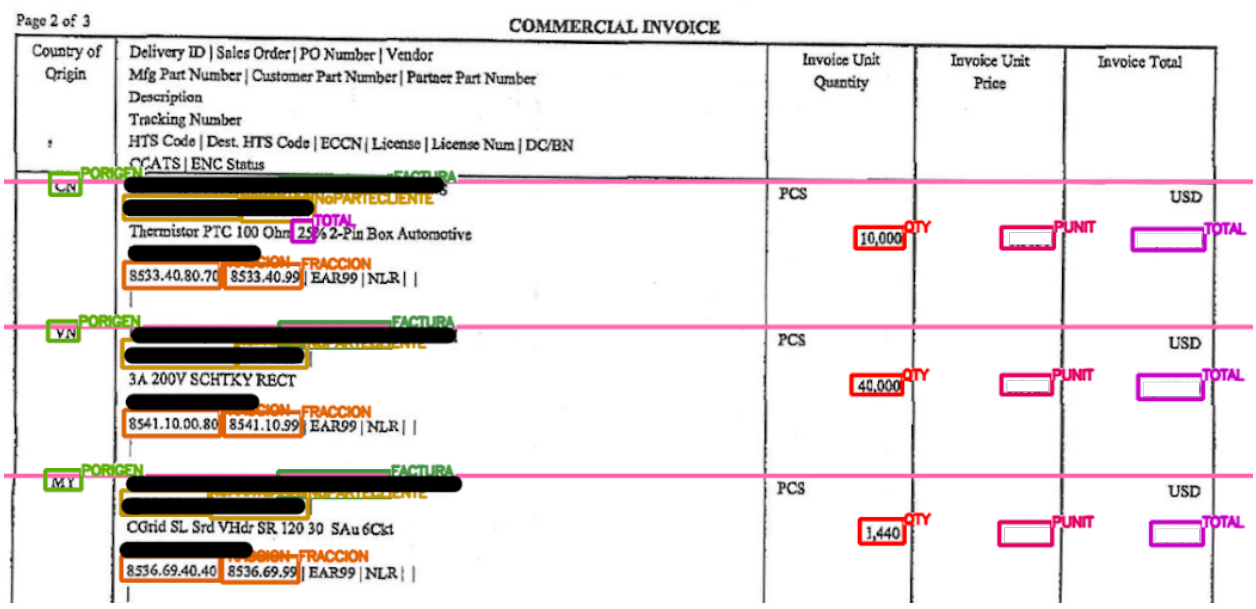


Figure 7-1: Sample result from the tabulation algorithm.

A custom, rule-based algorithm was developed to convert the output from the Key-Value Matching Module into a visual table form overlaid on top of the input document. This was critical as our dataset could have multiple rows of keys (such as multiple rows containing part numbers and quantities) within a single document, and hence the final result needed to be properly organized by row. The output was visual

key-value classification on the original invoice document processed by the pipeline, as shown in Figure 7-1 (some information is redacted for confidentiality reasons).

7.2 Algorithm Development

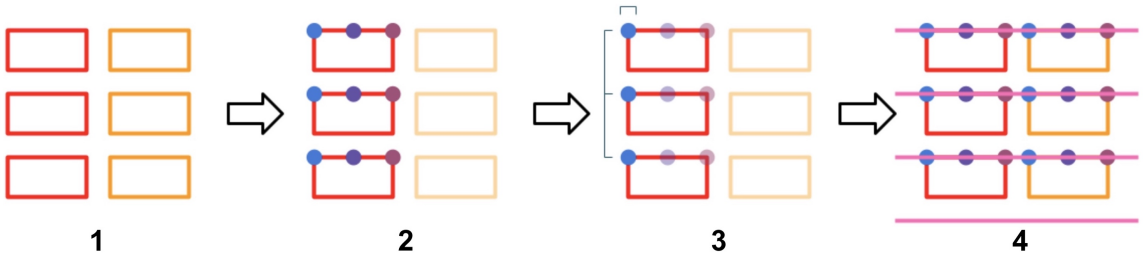


Figure 7-2: Diagram illustrating the tabulation algorithm's process.

To initially identify the rows, the algorithm took in the key classifications from the previous ML component, and divided the bounding boxes by classification. Then, to handle each key separately, it converted each box into a set of (x, y) coordinate pairs. Specifically, since words could be left-aligned, right-aligned, or center-aligned, the algorithm considered three separate sets of points for every bounding box: the top left corners, the top right corners, and the top middle point.

Next, the algorithm looked for a pattern of vertically-spaced, horizontally-aligned coordinate pairs, which became the guess for that key's row position. To obtain the position of the rows for the entire page, the algorithm combined the guess for each key, determined the most frequent row height, and selected the positions obtained by the highest key with that row height.

After determining the rows, the correct boxes within each row were selected, which was necessary as the model may have classified additional words per key - so the algorithm must filter for the correct word. This was accomplished via a combination of individual formatting checks and matching information across keys. For the formatting check, an example is checking if a word classified as a numeric-valued key (such as "QTY") was actually a numeric value. For matching information, the information should match up across keys; particularly, the quantity multiplied by the unit price

should equal the total price for each row.

Finally, after obtaining the preliminary values for each row, the algorithm post-processed the rows to improve certain cases. For example, an invoice may have the PO number written at the top of the page instead of a PO number in each row, so the algorithm would detect that PO number and append it to each row. After methods such as this, we obtained the final result of the pipeline: the extracted key-value information as a classified table.

7.3 Algorithm Improvement

An issue we identified in certain input cases was that the tabulation algorithm was not adept in processing rows of varying heights within a given document; an example of rows (pink lines) incorrectly superimposed onto the original document with different heights is shown in Figure 7-3. Rows of varying height were decently rare, and mainly occurred for documents from one particular company. Adjusting the algorithm to accommodate varying height rows just required the addition of a small adjustment factor to the tabulation algorithm, which was set to 20 pixels. With this improvement, our process for document tabulation - and development of the pipeline - was complete.

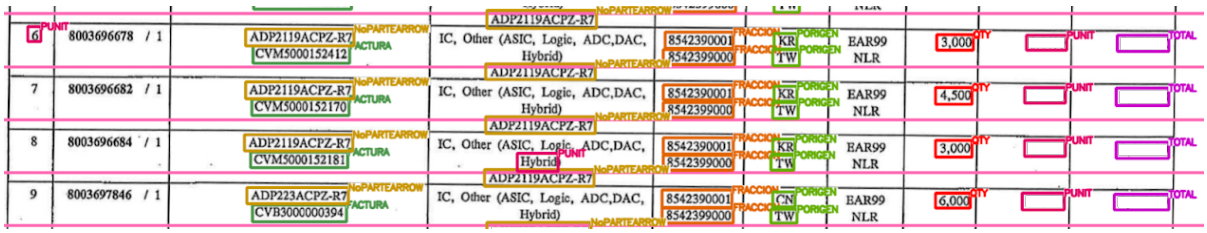


Figure 7-3: Example of a document with varying row heights.

Chapter 8

Final Evaluation

We developed three evaluation scripts separate from the three main modules, each of which were used to measure the performance of specific parts of the pipeline. The first evaluation script measured the performance of the first half of the pipeline (preprocessing, OCR, postprocessing). The model evaluation script measured the performance of the trained LayoutLM model, while the final evaluation script evaluated the accuracy of the comprehensive pipeline - beginning to end. A comparison of the pipeline to a leading commercially available solution is also presented.

8.1 Preprocessing, OCR, and Postprocessing Evaluation

The first evaluation method took in a JSON file containing the output of the pipeline's first half, which contained the first two modules. It compared the extracted outputs against the ground-truth CSV file, and measured the number of ground-truth words that were both present and correct; it is important to note that the words had not yet been matched with keys at this stage in the pipeline.

Looking at the results for the Arrow dataset, as shown in Tables 8.1 and 8.2, 92.22% of all keywords in the ground-truth CSV were both present and correct in the OCR output of their respective invoice documents. Each individual key's accu-

racy was also determined; for example, 91.67% of the "GUIA" (or waybill number) words from our ground-truth data was present within the information output from the pipeline's preprocessing, OCR, and postprocessing components. This communicated the fact that the processing half was able to correctly identify a significant majority of the information present in the invoice documents, which was a crucial first step in the objective of key-value extraction.

# of Correct Keys	# of Total Keys	Total Accuracy	Equally-weighted Accuracy
16,405	17,789	92.22%	91.68%

Table 8.1: Summarized results from the Preprocessing, OCR, and Postprocessing Evaluation.

Keys	# of Correct Keys	# of Total Keys	Key Accuracy
GUIA	396	432	91.67%
FECHA	374	435	85.98%
SHIP_METHOD	831	951	87.38%
PDC	624	716	87.15%
NoPARTEARROW	1,927	2,215	87.00%
NoPARTECLIENTE	81	83	97.59%
PUNIT	1,791	2,160	82.92%
FRACCION	1,883	2,037	92.44%
FACTURA	2,030	2,195	92.48%
PORIGEN	2,189	2,201	99.45%
QTY	2,196	2,204	99.64%
TOTAL	2,083	2,160	96.44%

Table 8.2: Individual results from the Preprocessing, OCR, and Postprocessing Evaluation.

8.2 Machine Learning Model Evaluation

The model evaluation script was used to independently evaluate the performance of the LayoutLM model on a test set of 266 invoice documents. The model had a **93.55% overall accuracy score** and a **74.20% macro F1 score**. A macro F1

score was used as opposed to a micro F1 score as micro F1 scores often don't return an objective measure of model performance when classes are imbalanced, while the macro F1 score does.

The model's accuracy was found to be higher than the F1 score. We deduced this to be due to the fact that the model sometimes classified extra words to each key. For example, looking at the "QTY" column in Figure 8-1, we observe there are 3,124 words which were classified as "QTY" and were indeed a "QTY," but there were also 631 words classified as "QTY" that were actually "_OTHER" - or not associated with any key. For our pipeline's identified objectives, classifying extra keys was preferred over missing keys, as the former could be handled within the tabulation algorithm.

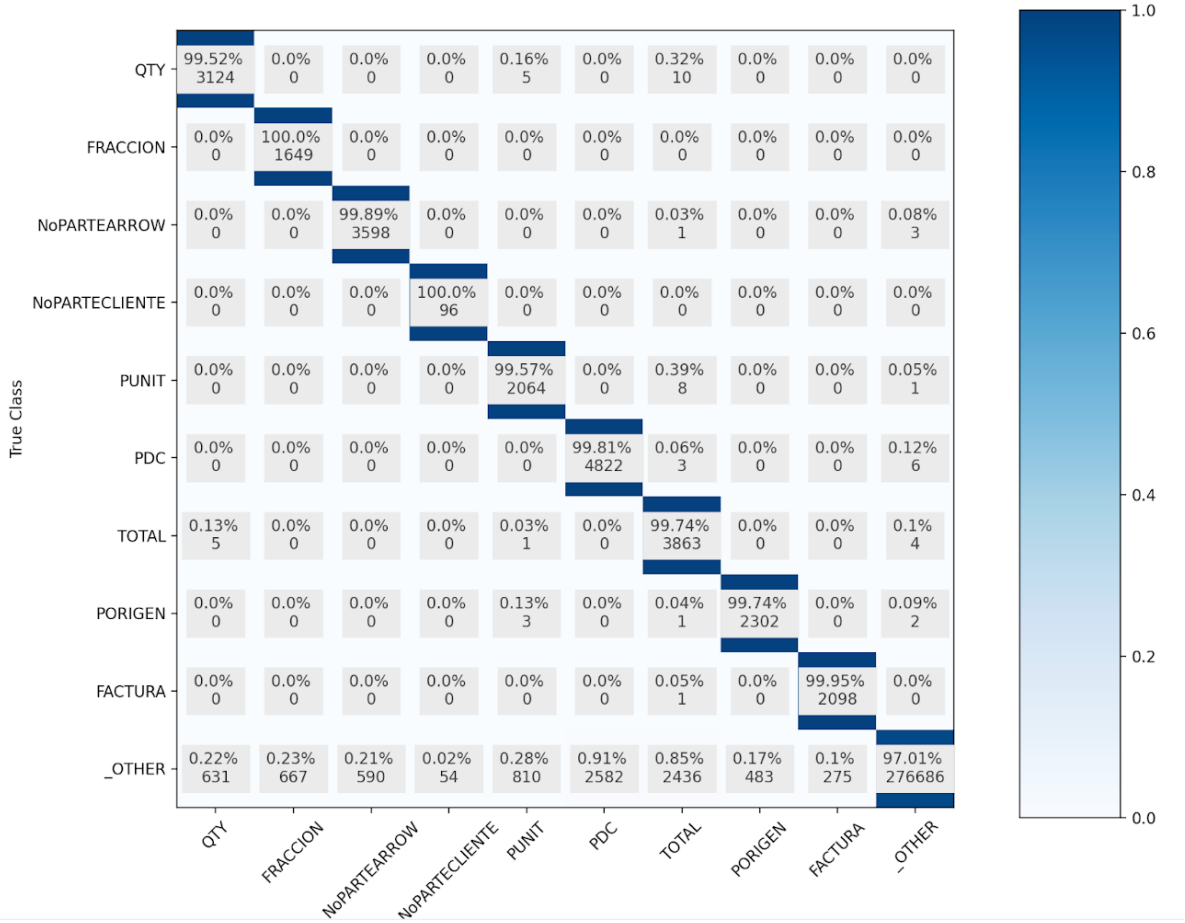


Figure 8-1: Confusion matrix for the trained LayoutLM model.

The model's output confusion matrix for the test set is presented in Figure 8-1, and is an evaluation technique used to summarize the performance of a classification

algorithm. In the matrix, the diagonal elements represent the number of datapoints for which the predicted label is equal to the true label, while the other elements are those that have been mislabeled by the classifier. The higher the diagonal values of the confusion matrix the better as it indicates more correct predictions. The bottom labels represent the classes given by the model, while the left labels represent the classes in the ground-truth. Visual assessment of our model's confusion matrix along the diagonal accordingly indicated effective classification.

It is also important to note that although the model was conclusively accurate, its input data was based on the output from the OCR and postprocessing modules, meaning that errors within those stages could have propagated into the model as well. For example, sometimes the training data within the Arrow dataset was missing some classifications, which could be attributed to the fact that the training data constructor was unable to match the tokens up to the ground truth in certain cases. This is something that can be improved upon as the pipeline continues to be developed.

8.3 Comprehensive Pipeline Evaluation

The final evaluation script measured the accuracy of the pipeline as a whole. Accuracy was measured as the number of correct key-value pairs divided by the number of total ground-truth key-value pairs on a test set of 266 invoice documents.

As shown in Tables 8.3 and 8.4, **the accuracy where each key was weighed equally was 87.85%, while the overall accuracy was 83.69%**. Additionally, the accuracy of each individual key extracted was determined. The most accurate keys ("GUIA," "PDC," "FECHA," "SHIP METHOD," and "NoPARTECLIENTE") were extracted with an accuracy greater than 90%, while all other keys except "NoPARTEAR-ROW" were extracted with an accuracy greater than 80%. It should be noted that the evaluation script only checked against rows that were present in the ground truth, so extra rows that the pipeline accidentally "processed" were ignored. Similarly, if a key was completely empty in the ground-truth, the evaluation script ignored any values given by the pipeline for that corresponding key.

# of Correct Keys	# of Total Keys	Total Accuracy	Equally-weighted Accuracy
7,984	9,540	83.69%	87.85%

Table 8.3: Summarized results from the Comprehensive Pipeline Evaluation.

Keys	# Correct of Keys	# of Total Keys	Key Accuracy
GUIA	259	266	97.37%
FECHA	250	266	93.98%
SHIP_METHOD	260	266	97.74%
PDC	249	266	93.61%
NoPARTEARROW	813	1230	66.10%
NoPARTECLIENTE	26	27	96.30%
PUNIT	1,008	1,190	84.71%
FRACCION	960	1,152	83.33%
FACTURA	1,016	1,225	82.94%
PORIGEN	1,053	1,231	85.54%
QTY	1,080	1,231	87.73%
TOTAL	1,010	1,190	84.87%

Table 8.4: Individual results from the Comprehensive Pipeline Evaluation.

8.4 Alternative Pipeline Comparison

In addition to evaluating the pipeline's performance, it was necessary to compare its final performance to other commercially available solutions. Previous studies have investigated the efficacy of such solutions for general purpose document processing tasks, but it was critical to understand those pipelines' performance on the Arrow dataset relative to our pipeline's own [62]. We identified Amazon Web Services (AWS) Textract as the most competitive, end-to-end commercial solution for comparison [10]. Textract's expense analysis can extract information such as contact information and vendor name without any particular template or explicit label from invoices and receipts, and hence was the best suited operation we could compare our pipeline with.

Textract explicitly extracts specified keys, of which we found the following that matched with our dataset: "VENDOR_NAME" ("PDC" in the Arrow dataset), "INVOICE_RECEIPT_DATE" ("FECHA" in the Arrow dataset), "QUANTITY" ("QTY" in the Arrow dataset), and "PRICE" ("TOTAL" in the Arrow dataset). The

other key-value pairs which Textract detected that didn't fall under one of these fields were classified as "OTHER," and Textract returned the keys it detected as a separate parameter.

To compare the final pipeline with Textract, we used six possible keys: the four keys shared with the standard fields, as well as "GUIA" and "PUNIT." These keys were chosen as they were the easiest to convert from the Textract output, after which we randomly selected a dataset of about 100 documents. We observed that our pipeline performed much more accurately than Textract on the Arrow dataset, as shown in Figure 8-2. Our pipeline more accurately extracts all of the specified key-value pairs in comparison to the Textract pipeline.

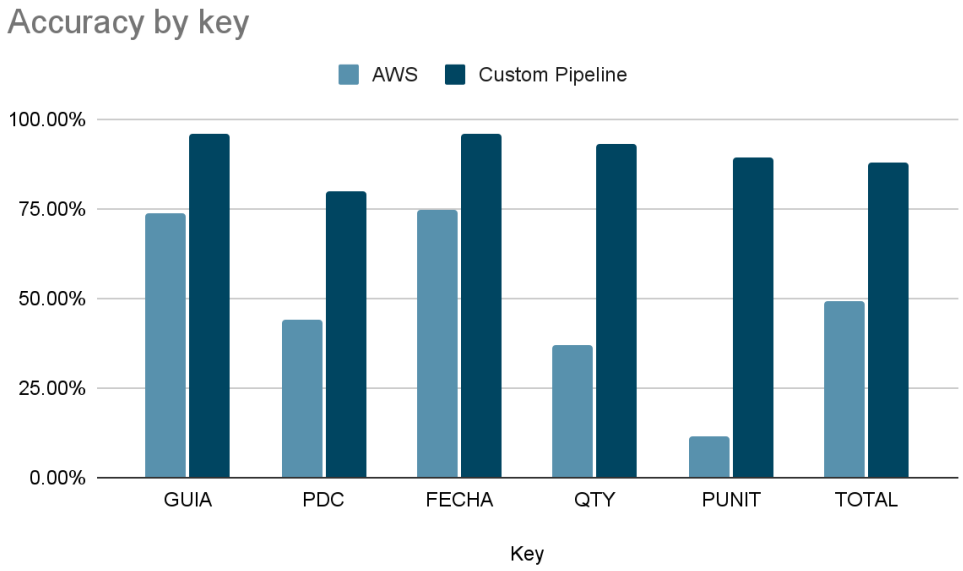


Figure 8-2: Accuracy comparison of AWS Textract and the developed pipeline.

We can partially attribute this to the fact that we have defined a specific methodology to extract the proper key-value pairs, while Textract is not fully positioned to handle the range of inputs from our dataset; for example, the waybill number ("GUIA") is not a standard field that Textract extracts, but by checking the obtained label for certain keywords it was able to obtain an accuracy score over 70%.

From this we concluded that the Textract pipeline could be improved, but only if a significant amount of time was spent in doing so. However with the objective of this comparison being to compare our complete pipeline with the commercial AWS

Textract pipeline, and the fact that adding custom rules to Textract would then signify that it isn't a standalone solution, we deduced our current pipeline as being more effective for key-value extraction from the Arrow dataset.

Chapter 9

Conclusion

9.1 Summary of Contributions

This thesis illustrates the development of a fully functional, end-to-end data pipeline that can extract key-value pairs from a complex dataset of commercial invoice documents provided by Arrow Electronics with an accuracy of ∼84%. Existing solutions would not provide the flexibility or ease of use needed for the identified objectives, and hence our custom process involved starting from the ground up. Previous studies from adjacent workspaces provided context about general issues associated with document processing and specific technical approaches that could be used to improve pipeline performance.

The deep-dive into the pipeline's architecture and its compartmentalized modules provide insight into its conceptual backgrounds, technical features, and design decisions. The Preprocessing and OCR Module used structured techniques to improve the dataset's OCR compatibility, after which the Google Cloud Vision service was selected and applied as the pipeline's OCR engine. The subsequent Postprocessing Module included the development of a custom Levenshtein distance-based method to correct identified errors in the extracted text, in addition to custom correction methods specific to the dataset. The Key-Value Matching Module finally matched key-value pairs via the use of deterministic algorithms or the LayoutLM machine learning model depending on the key. Custom methods were built and used to evalu-

ate performance, both on individual modules and on the comprehensive pipeline. To supplement the pipeline, we also developed a document tabulation algorithm to provide visual key-value classification on processed invoice documents, and an intuitive web interface to easily run the pipeline and examine results on a local system.

9.2 Future Work

Though the current state of the pipeline demonstrates that it is accurate in its extraction of key-value pairs, there is still room for improvement. New additions to the dataset could result in more kinds of unstructured documents and make accuracy maintenance difficult. To address this, we can begin by looking into ways of improving pipeline accuracy at its initial image processing stage. Incorporating neural networks or dimension reduction techniques into preprocessing or adding denoising to postprocessing are examples of methods that could further improve the inputs being fed into the Key-Value Matching Module [63, 64, 65].

As pipeline development continues, we hope to extend input types beyond just commercial invoices, meaning documents such as expense reporting and warehouse receiving should be able to be accepted, and their corresponding key-value pairs should be extracted. Previous studies involving such kinds of documents can provide inspiration on extending the current pipeline [66, 67]. For each new dataset of documents, we would need to restructure the data so that it is compatible with the pipeline, analyze/correct ground-truth data, retrain/evaluate the ML model within the Key-Value Matching Module, and ultimately test new key-value relationships. This would allow the pipeline to become more robust for widespread commercial extraction purposes.

Lastly, we hope to continue iterating on the web application interface to improve its functionality and overall user experience. Specifically, we hope to look into new ways of providing feedback on the accuracy of individual keys extracted from a particular document. Current development of a confidence metric within the interface is a good start, but we believe this can be improved and extended for a wider variety of key-value pairs in the future.

Bibliography

[1] Technavio, "Data-entry Outsourcing Services Market | Size, Share, Growth, Trends | Industry Analysis | Forecast 2025," January 2022. Available: https://www.technavio.com/report/data-entry-outsourcing-services-market-industry-analysis.

[2] Selwyn, Neil. “Data Entry: Towards the Critical Study of Digital Data and Education.” Learning, Media and Technology, vol. 40, no. 1, Informa UK Limited, 28 May 2014, pp. 64–82.

[3] Han, Jiang, et al. “Improving the Efficacy of the Data Entry Process for Clinical Research With a Natural Language Processing–Driven Medical Information Extraction System: Quantitative Field Research.” JMIR Medical Informatics, vol. 7, no. 3, JMIR Publications Inc., 16 July 2019, p. e13331.

[4] Hegghammer, Thomas. “OCR with Tesseract, Amazon Textract, and Google Document AI: A Benchmarking Experiment.” Journal of Computational Social Science, vol. 5, no. 1, Springer Science and Business Media LLC, 22 Nov. 2021, pp. 861–882.

[5] Singh, Himanshu. "Practical Machine Learning with AWS." Apress, 2021.

[6] Soysal, Ergin, et al. “CLAMP – a Toolkit for Efficiently Building Customized Clinical Natural Language Processing Pipelines.” Journal of the American Medical Informatics Association, vol. 25, no. 3, Oxford University Press (OUP), 24 Nov. 2017, pp. 331–336.

[7] Priya, K. “Customized Data Extraction and Effective Text Data Preprocessing Technique for Hydroxychloroquin Related Twitter Data.” Bioscience Biotechnology Research Communications, vol. 13, no. 13, Society for Science and Nature, 25 Dec. 2020, pp. 150–158.

[8] Cristani, Matteo, et al. “Future Paradigms of Automated Processing of Business Documents.” International Journal of Information Management, vol. 40, Elsevier BV, June 2018, pp. 67–75.

[9] Sage, Clement, et al. “Recurrent Neural Network Approach for Table Field Extraction in Business Documents.” 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, Sept. 2019.

[10] Rangarajan, P. "Analyzing Invoices and Receipts," Amazon, 2021. Available: https://docs.aws.amazon.com/textract/latest/dg/invoices-receipts.html

[11] Palacios, Rafael, and Gupta, Amar. “A System for Processing Handwritten Bank Checks Automatically.” Image and Vision Computing, vol. 26, no. 10, Elsevier BV, Oct. 2008, pp. 1297–1313.

[12] Gupta, Amar, et al. "Automatic Processing of Brazilian Bank Checks." 2016.

[13] Kim, Donghwa, et al. "Multi-co-training for document classification using various document representations: TF–IDF, LDA, and Doc2Vec." Information Sciences 477, 2019, pp. 15-29.

[14] Majumder, Bodhisattwa Prasad, et al. "Representation learning for information extraction from form-like documents." Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020.

[15] Mulfari, Davide, et al. "Using Google Cloud Vision in assistive technology scenarios." 2016 IEEE symposium on computers and communication (ISCC). IEEE, 2016.

[16] Q. Ye and D. Doermann. "Text Detection and Recognition in Imagery: A Survey." IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 7, July 2015, pp. 1480-1500.

[17] S. K. Badam, Z. Liu and N. Elmqvist. "Elastic Documents: Coupling Text and Tables through Contextual Visualizations for Enhanced Document Reading." IEEE Transactions on Visualization and Computer Graphics, vol. 25, no. 1, Jan. 2019, pp. 661-671.

[18] H. Zhou, L. Shao and H. Zhang. "SRRNet: A Transformer Structure with Adaptive 2D Spatial Attention Mechanism for Cell Phone-Captured Shopping Receipt Recognition." IEEE Transactions on Consumer Electronics, 2022.

[19] Park, Seunghyun, et al. "CORD: a consolidated receipt dataset for post-OCR parsing." Workshop on Document Intelligence at NeurIPS 2019, 2019.

[20] Huang, Zheng et al. "ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction." 2019 International Conference on Document Analysis and Recognition (ICDAR), 2019, pp. 1516-1520.

[21] Ghosh, Aindrila, et al. "A comprehensive review of tools for exploratory analysis of tabular industrial datasets." Visual Informatics 2.4, 2018, pp. 235-253.

[22] D. Baviskar, S. Ahirrao and K. Kotecha. "Multi-Layout Unstructured Invoice Documents Dataset: A Dataset for Template-Free Invoice Processing and Its Evaluation Using AI Approaches." IEEE Access, vol. 9, 2021, pp. 101494-101512.

[23] Rosid, Mochamad Alfan, et al. "Improving text preprocessing for student complaint document classification using sastrawi." IOP Conference Series: Materials Science and Engineering Vol. 874. No. 1. IOP Publishing, 2020.

[24] Shobha Rani, N., A. Sajan Jain, and H. R. Kiran. "A unified preprocessing technique for enhancement of degraded document images." International Conference on ISMAC in Computational Vision and Bio-Engineering, Springer, Cham, 2019.

[25] Binmakhashen, Galal M., and Sabri A. Mahmoud. "Document layout analysis: a comprehensive survey." ACM Computing Surveys (CSUR), 2019, pp. 1-36.

[26] Huang, Yilun, et al. "A YOLO-based table detection method." 2019 International Conference on Document Analysis and Recognition (ICDAR). IEEE, 2019.

[27] J. Memon, et al. "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)." IEEE Access, vol. 8, 2020, pp. 142642-142668.

[28] Patel, Chirag et al. "Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study." International Journal of Computer Applications, 2012, pp. 50-56.

[29] H. Hosseini, B. Xiao and R. Poovendran. "Google's Cloud Vision API is Not Robust to Noise." 2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA), 2017, pp. 101-105.

[30] Chakraborty, Sunandan, et al. "Extraction of (key, value) pairs from unstructured ads." 2014 AAAI Fall Symposium Series, 2014.

[31] Salloum, Said A., et al. "Using text mining techniques for extracting information from research articles." Intelligent Natural Language Processing: Trends and Applications, Springer, Cham, 2018, pp. 373-397.

[32] Karthikeyan, T., et al. "Personalized content extraction and text classification using effective web scraping techniques." International Journal of Web Portals (IJWP) 11.2, 2019, pp. 41-52.

[33] Xu, Y., et al. "LayoutLM: Pre-training of Text and Layout for Document Image Understanding." Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2020.

[34] Hong, Teakgyu, et al. "Bros: A pre-trained language model focusing on text and layout for better key information extraction from documents." Proceedings of the AAAI Conference on Artificial Intelligence Vol. 36, 2022.

[35] Appalaraju, Srikar, et al. "Docformer: End-to-end transformer for document understanding." Proceedings of the IEEE/CVF International Conference on Computer Vision, 2021.

[36] Kim, Geewook, et al. "OCR-Free Document Understanding Transformer." 5, arXiv, 2021.

[37] Morris, David, Peichen Tang, and Ralph Ewerth. "A neural approach for text extraction from scholarly figures." 2019 International Conference on Document Analysis and Recognition (ICDAR), IEEE, 2019.

[38] A. Lertpiya, T. Chalothorn and E. Chuangsuwanich. "Thai Spelling Correction and Word Normalization on Social Text Using a Two-Stage Pipeline With Neural Contextual Attention." In IEEE Access, vol. 8, 2020, pp. 133403-133419.

[39] Wang, Shirly, et al. "MIMIC-Extract: a data extraction, preprocessing, and representation pipeline for MIMIC-III." Proceedings of the ACM Conference on Health, Inference, and Learning (CHIL '20). Association for Computing Machinery (ACM), New York, NY, USA, 2020, pp. 222–235.

[40] Böschen, Falk, and Ansgar Scherp. "Formalization and preliminary evaluation of a pipeline for text extraction from infographics." CEUR Workshop Proceedings Vol. 1458, 2015.

[41] Munappy, A., Bosch, J., et al. "Modelling Data Pipelines." 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), 2020, pp. 13-20.

[42] Shen, Mande and Lei, Hansheng. "Improving OCR performance with background image elimination." 2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2015, pp. 1566-1570.

[43] Bradski, G. The OpenCV Library. Dr. Dobb's Journal of Software Tools. 2000.

[44] “Detect Text in Images | Cloud Vision API |.” Google Cloud, January 2022. Available: cloud.google.com/vision/docs/ocr.

[45] Paliwal, S., et al. "TableNet: Deep Learning Model for end-to-end table detection and tabular data extraction from scanned document images." 2019 International Conference on Document Analysis and Recognition (ICDAR), 2019.

[46] Chen, SH., Chen, YH. "A Content-Based Image Retrieval Method Based on the Google Cloud Vision API and WordNet." Intelligent Information and Database Systems. ACIIDS 2017. Lecture Notes in Computer Science(), vol 10191. Springer, Cham, 2017.

[47] Thammarak, Karanrat, et al. “Comparative Analysis of Tesseract and Google Cloud Vision for Thai Vehicle Registration Certificate.” International Journal of Electrical and Computer Engineering (IJECE), vol. 12, no. 2, Institute of Advanced Engineering and Science, 1 Apr. 2022, p. 1849.

[48] Malkadi, Abdulkarim, Mohammad, Alahmadi, and Haiduc, Sonia. "A study on the accuracy of ocr engines for source code transcription from programming screencasts." Proceedings of the 17th International Conference on Mining Software Repositories, 2020.

[49] Nguyen, T., et al. "Survey of Post-OCR Processing Approaches." ACM Comput. Surv., 54(6), 2021.

[50] X. Qiu, et al. "A Post-Processing Method for Text Detection Based on Geometric Features." IEEE Access, vol. 9, 2021, pp. 36620-36633.

[51] Berger, Bonnie, Michael S. Waterman, and Yun William Yu. "Levenshtein distance, sequence comparison and biological database search." IEEE Transactions on Information Theory 67.6, 2020, pp. 3287-3294.

[52] C. Yao, X. Bai and W. Liu, "A Unified Framework for Multioriented Text Detection and Recognition." IEEE Transactions on Image Processing, vol. 23, no. 11, Nov. 2014, pp. 4737-4749.

[53] Haldar, Rishin, and Debajyoti Mukhopadhyay. "Levenshtein Distance Technique in Dictionary Lookup Methods: An Improved Approach." 1, arXiv, 2011.

[54] Infoscout. Infoscout/weighted-levenshtein: Weighted Levenshtein Library. GitHub, January 2022. Available: https://github.com/infoscout/weighted-levenshtein

[55] Hicham, Gueddah. "Introduction of the Weight Edition Errors in the Levenshtein Distance." 1, arXiv, 2012.

[56] Devlin, Jacob, et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), Association for Computational Linguistics, Minneapolis, Minnesota, 2019, pp. 4171–4186.

[57] Nguyen, TA.D., Vu, H.M., Son, N.H., Nguyen, MT. "A Span Extraction Approach for Information Extraction on Visually-Rich Documents." Document Analysis and Recognition – ICDAR 2021 Workshops. ICDAR 2021. Lecture Notes in Computer Science(), vol 12917. Springer, 2021.

[58] H. Guo, et al. "EATEN: Entity-Aware Attention for Single Shot Visual Text Extraction." 2019 International Conference on Document Analysis and Recognition (ICDAR), 2019, pp. 254-259.

[59] Park, S., et al. "CORD: a consolidated receipt dataset for post-OCR parsing." Workshop on Document Intelligence at NeurIPS 2019, 2019.

[60] Xu, Yang, et al. "LayoutLMv2: Multi-Modal Pre-Training for Visually-Rich Document Understanding." 4, arXiv, 2020.

[61] Huang, Yupan, et al. "LayoutLMv3: Pre-Training for Document AI with Unified Text and Image Masking." 3, arXiv, 2022.

[62] Xu, Ting, et al. "Intelligent Document Processing: Automate Business with Fluid Workflow." Konica Minolta technology report 18, 2021, pp. 89-94.

[63] Rehman, Amjad, and Tanzila Saba. "Neural networks for document image preprocessing: state of the art." Artificial Intelligence Review 42.2, 2014, pp. 253-273.

[64] A. I. Kadhim, et al. "Text Document Preprocessing and Dimension Reduction Techniques for Text Document Clustering." 2014 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology, 2014, pp. 69-73.

[65] Nafchi, Hossein, et al. "Application of phase-based features and denoising in postprocessing and binarization of historical document images." 2013 12th International Conference on Document Analysis and Recognition, IEEE, 2013.

[66] Ding, Pan, et al. “Textual Information Extraction Model of Financial Reports.” Proceedings of the 2019 7th International Conference on Information Technology: IoT and Smart City, ACM, 20 Dec. 2019.

[67] Liu, Xiaojing, et al. "Graph Convolution for Multimodal Information Extraction from Visually Rich Documents." 1, arXiv, 2019.