Information Extraction From Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques

W.B. Darsha

2022

Information Extraction From Scanned Invoices using Machine Learning, OCR and Spatial Feature Mapping Techniques

A dissertation submitted for the Degree of Master of Computer Science

W.B. Darsha
University of Colombo School of Computing
2022

DECLARATION

The thesis is my original work and has not been submitted previously for a degree at this or any other university/institute.

To the best of my knowledge it does not contain any material published or written by another person, except as acknowledged in the text.

Student Name: W. B. Darsha

Registration Number: 2018/MCS/010

Index Number: 18440105

Signature: Date: 2023-02-28

This is to certify that this thesis is based on the work of

Mr./Ms.

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Certified by:

Supervisor Name:

Signature: Date:

ACKNOWLEDGEMENTS

First of all, I would like to express my sincere gratitude to UCSC for selecting me for the Master of Computer Science (MCS) degree programme. Specific thanks goes to my supervisor Mr. Gihan.P. Seneviratne for the support, guidance and patience given throughout this project.

Also I convey my heartiest thanks to my parents, wife, relations and friends who gave me strength and courage in many ways.

Furthermore, I recall all the resource persons spread all over the world who generated the vast knowledge I gained during this study.

ABSTRACT

Receiving invoices as scanned images is one of the biggest problems business organizations are still facing. Consuming human effort for converting scanned invoices to text documents is not sustainable because of their low performance even inherently capable of. With the recent escalations of Computer Vision technology with Machine Learning we were seeing new dimensions for addressing this bursting problem. Optical Character Reading (OCR) is the latest way of extracting text from images in general context, but the output was not much helpful for identifying key parameters from invoices. Hence we employed an object detection algorithm called You Only Looks Once (YOLO) first to capture text blobs in granular level, then streamlined them to OCR and finally processed spatial information with pattern matching techniques. Using this improved approach we could successfully extract not only key parameters like merchant information, invoice no, datetime, total but also the invoice items in the table body, and indeed with a high performance. Thus methodology we developed can be adapted to any scanned invoice dataset with proper adjustments, and also for any other document type.

Keywords: scanned invoices/receipts, machine learning, YOLO, OCR, Tesseract, image processing, pattern matching, spatial information

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Prolegomena

Businesses around the world operate in different modes such as Business-to-Business, Business-to-Consumer and Business-to-Government, and in different scales large, medium or small. In any business, two parties, buyer and seller maintain business documents to keep track of their transactions for financial processing, inventory control, auditing, exploring sales patterns etc. Nowadays there are plenty of document types exchanged between business entities, name a few Catalog, Order, Order Response, Invoice, Application Response, Goods Received Note, Delivery Note that emerge frequently. Among those document types, Invoice plays a vital role, since it is issued even by small sized retailers, maybe due to enforced by the government rules and Regulations.

With the rise of digital transformation, businesses started to generate, transmit and store documents electronically. But the majority of the business entities still have not fully adapted to this modern approach. For example some merchants still issue handwritten bills and some ones may generate invoices/receipts electronically but again deliver it to the customer as a printed copy. Then a smart customer has to re-digitize it if he/she has a requirement to process it in a digital medium. Such scenarios can be caused by both or either business parties lacking an adequate software system, and probably no easy integration between existing software systems. The Motivation section elaborates it with a real world case study.

1.2 Motivation

Pagero (“Pagero | Digitalise and streamline your business processes,” 2018) is one of the world's largest growing business networks which connects trading companies in all sizes including tech giants like Microsoft, SAP and HP. The key features of this middleware platform are electronic business document transformation and transferring, but not limited to. Pagero is a life saver for companies where they don’t have a direct integration between each other, and when making connections to hundreds of thousands of business partners is a nightmare. The majority of document traffic coming to this organization is electronic

documents that are processable text files, but meanwhile a significant portion of traffic is still received as scanned invoices (images). However ninety nine percent of the times the recipient of the invoice needs it in a textual format i.e. XML document instead of an image to further process it in their ERP system. In addition to that, Pagero needs those textual data for transformation, validation, enrichment, business analytics, sending to tax authorities of certain governments prior to the recipient, and perhaps for customer billing purposes.

In image to text conversion, the system should process the textual and visual data in scanned images to extract information semantically (understand and map the related text, i.e {"Total": "$100"}). Since Pagero does not have an in-house tool to fulfill this goal, images are sent to a third party service and retrieves data back in XML, but no transparency on whether it is a manual, automated or hybrid system. And also there is a risk of exposing customer data to a third party company that may lead to a possible violation of GDPR ("General Data Protection Regulation (GDPR) Compliance Guidelines," n.d.) in European Union and any other country specific data protection acts. Therefore the company is willing to have a fully automated in-house developed tool for extracting information from scanned invoices. Author is currently working for this fortune company and commenced a research project aiming to find a solution for this lacking area.

1.3 Statement of the problem

Problem: Extracting information from scanned invoices (images) is a challenging task.

Human beings are inherently capable of reading and understanding text and other types of objects in an image easily. Required intelligence consists of vision and linguistics aspects has been developed unconsciously from childhood through practice. But if a human operator is assigned to enter data read from scanned documents to a software system, it will be a tedious job. When the content is too long, messy, in low quality and a high volume of documents arrives continuously the job will become worse. Therefore organizations need fully automated systems to get this job done without human operators, or at least with very less intervention for making minor corrections.

Computers are robust in processing direct text since it can be done using typical rule-based algorithms (Symbolic AI). But going beyond it appears that computers should require a

human like Non-Symbolic AI to extract information from scanned documents in heterogeneous formats and styles. Even location details of a particular part in the same document format varies from one image to another during the scanning process. And the intelligence systems developed to perform such a reading task should be greater in speed and accuracy than humans for it to be really usable.

Data Science theories seem to be promising for solving this kind of problem which needs an expert artificial intelligence. And special kinds of tools and techniques are rapidly popping up to cater those theories. But the problem is general software engineering knowledge and skills are not sufficient to perform in this muddy playground. There is a huge learning curve for getting familiarized with these broad theories and becoming skilled in relevant tools and techniques. Also an engineer needs a high patience to keep experimenting until finds a proper solution for the problem through many trial and error rounds, since many AI techniques are not based on clear reasoning approaches. As a result Data Science Engineers are highly paid than typical Software Engineers in the industry.

Building an expert artificial intelligence system using techniques that lay under Data Science is very costly in resource consumption and time. For example creating an Artificial Neural Network is a long running task which executes the same process through many rounds continuously until it comes to a saturation level. In that case a personal computer with an average CPU and memory may not be helpful, even though it can do the job it may take a lot of hours or days to complete the task. A personal computer with a GPU can increase the performance to some extent. Another option to address this problem is to go for a cloud computing platform but it is a bit expensive and probably university projects are non-funding ones.

1.4 Research Aims and Objectives

1.4.1 Aims

- The main aim of this research work is to create a software tool that can semantically extract information, i.e key-value pairs like {“Total”: “$100”} from scanned invoice/receipt images with a higher speed and accuracy than humans.

- Also hope to identify how to customize the above approach for any other scanned document type.
- As a side benefit, derive a common way to detect target object types from any image (theoretically supported for any vision problem).

1.4.2 Objectives

- Extract textual information from scanned invoice/receipt images.
- More specifically, extract key parameter values such as merchant, customer, date and time, invoice no, invoice/purchase items, total amount etc.
- Create a company (i.e. Pagero) specific XML document composed with extracted information which then can be transformed into any other recipient specific format.
- Thus the company can reduce the cost by eliminating the use of third party services of this kind.
- The company can eliminate the risk of exposing customer data to third party services, thus refraining from violating regional or country specific data protection rules and regulations like GDPR.
- Take the marketing advantage by spreading the word that the organization is now using AI techniques.

1.5 Scope

During the past few years information extraction from scanned invoices emerged as a most interesting research area with a significant business value. Community is encouraged by the competition ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction (Huang et al., 2019) organized by the International Conference on Document Analysis and Recognition (ICDAR) (“IEEE Xplore - Conference Table of Contents,” n.d.). According to the ICDAR2019, research is carried out through three key tasks (“Tasks - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition,” n.d.) as listed below.

Task 1 - Scanned Receipt Text Localization
Task 2 - Scanned Receipt OCR (Optical Character Reading)
Task 3 - Key Information Extraction from Scanned Receipts

Thus, the scope of the study lies among these three areas as more described in Chapter 2 - Literature Review targeting to implement an end-to-end scanned invoice/receipt reading system.

1.6 Structure of the Thesis

Rest of the document is organized through chapters: Literature Review, Methodology, Evaluation and Results, and Conclusion and Future Work. The Literature Review chapter discusses early development, latest development and challenges in scanned document reading. Latest development is broken down into three areas as listed in the scope section. The Methodology chapter presents how the experiments went, what are the alternative solutions, why and how those were used or left aside, process flows, system architecture and components, used technologies etc. Evaluation and Results chapter thoroughly discusses the evaluation criteria used for each stage, definitions and meanings of their terms, and the results drawn in textual and visual representations. Finally, the Conclusion and Future Work chapter converges the work into a conclusion stating whether the research assumption is proven or not, and the future tasks that should be carried out to direct the proof of concept work to a production grade system.

CHAPTER 2
LITERATURE REVIEW

2.1 Introduction

Under e-invoice presentation formats, PDF (Portable Document Format) is a widely used one, therefore a lot of research has been conducted to read texts in PDF docs and consequently many successful implementations were invented. But those techniques cannot be used directly in this research since this is about reading bare scanned invoice images that only contain RGB or gray-scale color values of pixels. Furthermore, invoice image reading is far more difficult than PDF invoice reading. In these both streams, correlating text parts semantically is met definitely, but some previously invented algorithms seem to be unnecessarily complicated, i.e. space algorithm by (Pettagam Tharindu Rukshan Ubewikkrama, 2020). Hence simple and robust approaches will be explored in this research.

According to the ICDAR2019 Tasks and other sources of literature, information extraction from scanned invoice/receipt images can be broken down into three major tasks as mentioned in 1.5 Scope section also.

Task 1 - Scanned Receipt Text Detection and Localization
Task 2 - Scanned Receipt OCR (Optical Character Reading)
Task 3 - Key Information Extraction from Scanned Receipts

As per the domain experts, the first two tasks are not too difficult to achieve, but for a novel researcher they are still challenging because a vast knowledge and a skill set should be acquired, and a lot of literature review and experiments should be carried out.

2.2 Task 1 - Scanned Receipt Text Detection and Localization

For the Task 1 - Scanned Receipt Text Detection and Localization community has widely used machine learning techniques such as deep neural network algorithms. When considering Optical Character Reading (OCR) algorithms which will be more detailed under Task 2, that can do text detection and localization also to some extent in addition to text extraction. They have used Long Short-Term Memory (LSTM) which falls under Recurrent Neural Network

(RNN) family that consists of a memory module (Staudemeyer and Morris, 2019). But according to the previous research RNNs are more suitable for Natural Language Processing (NLP) tasks (Yin et al., 2017), while the Convolutional Neural Network (CNN) family is used for computer vision related chores (Du, 2018). Thus apparently CNNs are the ideal algorithms for object detection and localization. The popular CNN based algorithms considered are Convolutional Neural Network (CNN), Recurrent-Convolutional Neural Network (R-CNN), Fast R-CNN, Faster R-CNN (Du, 2018), RetinaNet (Lin et al., 2018), Single-Shot MultiBox Detector (SSD) (Liu et al., 2016) and You Only Look Once (YOLO) (Redmon et al., 2016). Usually these algorithms are capable of doing classification too in addition to object detection and localization, both in still images and videos. The series of CNN, R-CNN, Fast R-CNN and Faster R-CNN are two-stage detectors that have two separate models for object detection and classification, and use sliding window technique which degrades the performance of the work. But algorithms like RetinaNet, SSD and YOLO have mechanisms to divide the image to a grid and apply object detection and classification concurrently for all the cells in one-stage, thus they have been given their names with that meaning.

According to Figure 1 (Redmon and Farhadi, n.d., p. 3) YOLO is the leading object detection and classification algorithm in the computer vision world with regards to the performance matrices. YOLO evolved through several versions and YOLOv5 (“ultralytics/yolov5,” 2022) is the widely used version in the industry and YOLOv7 (Wang et al., 2022, p. 7) is the latest version at the time of writing. Furthermore each version has several variations targeting different problems, platforms and devices.

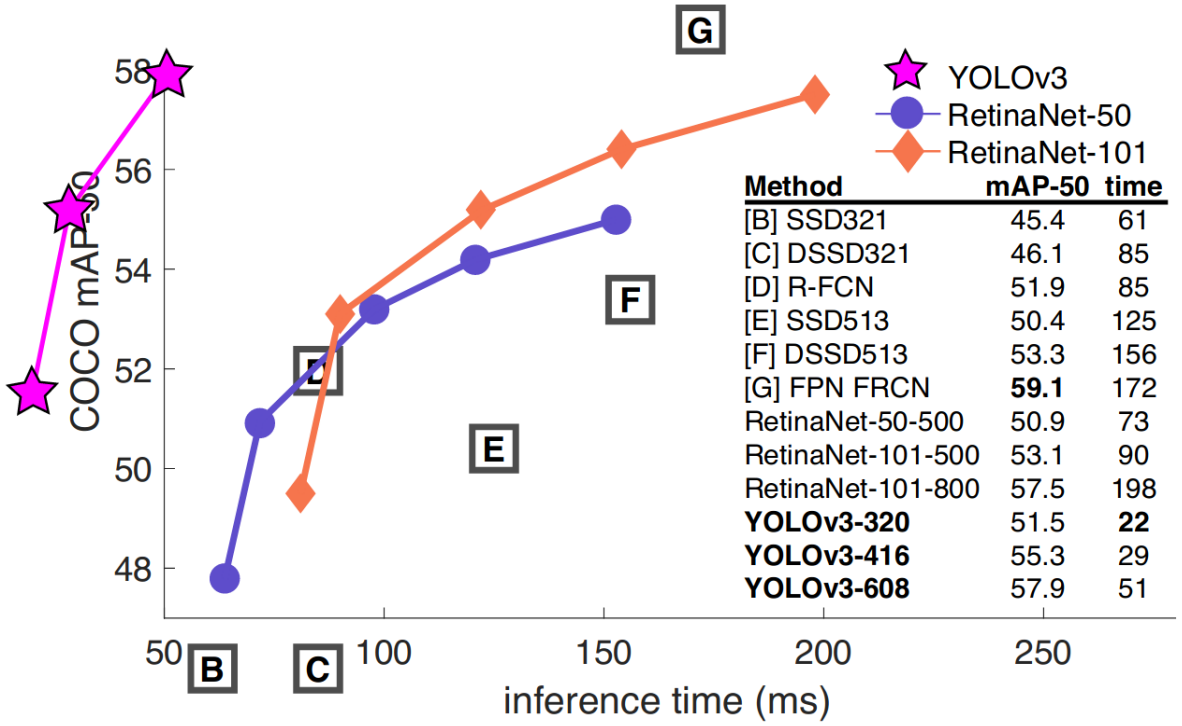


Figure 1: Speed/accuracy tradeoff between YOLOv3 and few other object detection and classification algorithms. Less inference time (more to the left) and high mAP-50 accuracy (more to the top) is better.

2.2 Task 2 - Scanned Receipt OCR (Optical Character Reading)

Task 2 - Scanned Receipt OCR (Optical Character Reading) is about extracting text from detected text blobs in an image. But some algorithms are capable of text detection and localization too in addition to extracting texts as mentioned in Task 1 literature above. Usually they expect large and clear image blobs, otherwise the operation will fail or ended up with a low accuracy. As mentioned previously OCR tools mainly use the Long Short-Term Memory Recurrent Neural Network (LSTM-RNN) algorithms. Tesseract OCR engine (Smith, 2007) is the most popular tool that can be found under this category which was originally developed by Hewlett Packard (HP) and later sponsored by Google. Tesseract OCR engine uses LSTM which is focused on line recognition from version 4 while older versions work by recognizing character patterns ("Tesseract OCR," 2022). It supports over 100 spoken languages and can be further trained for new languages. Now it is a free open-source and standalone tool written in C/C++ and has wrappers for several other programming languages including Java and Python.

Nowadays cloud based OCR tools are also available, that they can be called via APIs from our systems, ie. Google Cloud Vision API (“Detect text in images | Cloud Vision API | Google Cloud,” n.d.). But the exact technology behind those proprietary billable tools are not revealed. Since accuracy issues still persist, researchers are further seeking better algorithms for OCR.

2.3 Task 3 - Key Information Extraction from Scanned Receipts

Task 3 - Key Information Extraction from Scanned Receipts/Invoices is still at an unmatured level as declared by ICDAR2019. Gaining a high accuracy and speed is still a challenge in this area. Regular expressions can be seen as a basic tool for the work and limited to identifying of a few domain specific key parameters, i.e. Company name, Sender, Recipient, Total, Tax etc. Corresponding values are located around those key parameters, probably on the right or bottom side. Then extracting correct key-value pairs through some correlating technique is a challenging task, i.e. {“Total”: “$100”}.

Furthermore the lacking and hardest part is extracting the invoice items, in other words table items, i.e. {“Item no”: “123”, “name”: “Product_1”, “Unit price”: “20”, “Quantity”: “50”}. Specifically common techniques above may give a low accuracy when invoice items go farther from the table header. Extracting invoice items becomes more crucial when they are in handwritten format (not typewritten) which overflows to other rules and text overlapping occurs. More difficulty might be added when the language of the text is not English. Another general problem is some images are skewed or faded or some parts are erased. Hence researchers are trying to process textual, visual and spatial features in the scanned invoices with advanced techniques like biLSTM and word embedding to achieve this goal (Patel and Bhatt, 2020). Exploiting the classification feature from Task 1 - Scanned Receipt Text Detection and Localization algorithms in this step may be the life saver that should be experimented. Thus Task 3 is the dominating part at the moment that needs more improvements.

2.4 Summary

Generally as per the results of ICDAR 2019 Challenge on "Scanned receipts OCR and key information extraction" (SROIE) competition (Huang et al., 2019) there should be further

improvements in both accuracy and speed in all three tasks discussed above. Consequently enthusiast researchers from different information technology companies and universities are still publishing their valuable work in the ICDAR 2019 Results section ("Results - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition," n.d.), and also in other reputable journals and mediums throughout the three tasks separately and sometimes as a whole.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This is the most important chapter of this thesis document. It describes in detail about the methodology used to design and implement the experiment based on the research hypothesis. Experiment was carried out based on three tasks mentioned in the section 1.5 Scope. In some tasks alternative approaches were tested and selected reasonable ones considering resource/time tradeoffs and limitations. This chapter is organized through Hypothesis, Dataset, Input, Output, Process, Design and Implementation of the proposed system.

3.2 Hypothesis

Information extraction from scanned invoice/receipt images should be able to be achieved using Machine Learning, OCR and spatial feature mapping techniques.

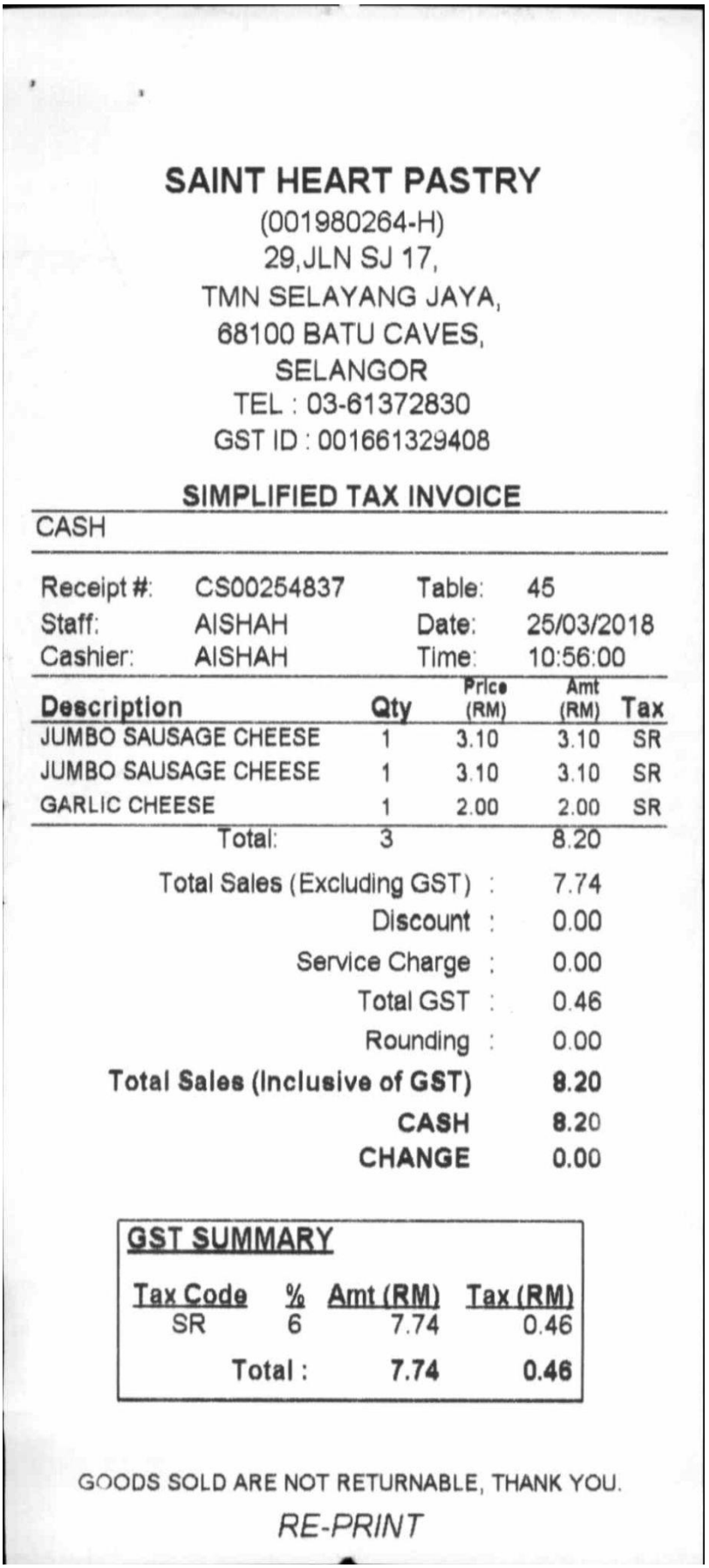
3.3 Dataset

The very first step is to prepare the dataset as the project is mainly going to be driven through Machine Learning techniques thus indeed dataset is the key part. This research was inspired by the case study mentioned in 1.2 Motivation section, despite it is not possible to receive Pagero (“Pagero | Digitalise and streamline your business processes,” 2018) data since the majority of them are from european customers and author is a non-european even though works for Research and Development department of the same organization, therefore issuing data to the author clearly leads to a violation of GDPR regulation (“General Data Protection Regulation (GDPR) Compliance Guidelines,” n.d.) in European Union. Also when proceeding through this section the reader will understand that only invoice/receipt images are not enough to run relevant Machine Learning algorithms. They require corresponding annotations which are text files containing metadata that explains what is inside the image. Annotating hundreds-thousands of invoice/receipt images consisting of many parts is a very time consuming manual work even though there are graphical tools out there, i.e Roboflow, an online image annotation tool (“Roboflow,” n.d.). Therefore considering these impediments the

author decided to use a well annotated free dataset named SROIE dataset that will help to be productive immediately.

SROEI dataset is a benchmarking dataset issued to researchers of ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction. Anyone can retrieve the dataset from the conference website's Downloads page after free registration and login, it will be shared as a Google Drive folder ("Downloads - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition," n.d.). It includes images of real receipts issued by Point-of-Sales (POS) machines of small sized merchants captured through a scanning process, and no generated or augmented images involved. Images are in .jpg format and their corresponding annotation/label files are in .txt format. For training purposes there are 700+ records and for testing there are 350+ records, but are not very much clean and should be filtered and pre-processed appropriately prior to use. Dataset consists of images in different sizes and in RGB color format. Furthermore, the dataset includes a few images with zoomed out mode, skewed, faded and with missing parts for considering such scanning defects. And also a very few other images contain barcodes and handwritten parts as shown in Figure 3 below, but they include redundant information like Invoice No and Total without a direct label nearby which makes it harder for computers to determine the class of the value. Due to that reason creators of the dataset have not annotated those barcodes and handwritten values, in other words they are not considered for training and testing tasks.

A sample scanned receipt image from the SROIE dataset is shown in Figure 2 below.



SAINT HEART PASTRY
(001980264-H)
29,JLN SJ 17,
TMN SELAYANG JAYA,
68100 BATU CAVES,
SELANGOR
TEL : 03-61372830
GST ID : 001661329408

SIMPLIFIED TAX INVOICE

CASH

Receipt #:	CS00254837	Table:	45
Staff:	AISHAH	Date:	25/03/2018
Cashier:	AISHAH	Time:	10:56:00

Description	Qty	Price (RM)	Amt (RM)	Tax
JUMBO SAUSAGE CHEESE	1	3.10	3.10	SR
JUMBO SAUSAGE CHEESE	1	3.10	3.10	SR
GARLIC CHEESE	1	2.00	2.00	SR
Total:	3		8.20	

Total Sales (Excluding GST)	:	7.74
Discount	:	0.00
Service Charge	:	0.00
Total GST	:	0.46
Rounding	:	0.00
Total Sales (Inclusive of GST)		**8.20**
CASH		**8.20**
CHANGE		**0.00**

GST SUMMARY

Tax Code	%	Amt (RM)	Tax (RM)
SR	6	7.74	0.46
Total :		**7.74**	**0.46**

GOODS SOLD ARE NOT RETURNABLE, THANK YOU.

RE-PRINT

Figure 2: Scanned receipt image

A part of the corresponding annotation text file content for the above image is shown below. Full content can be found in the Appendix I.

215,215,720,215,720,255,215,255,SAINT HEART PASTRY
342,269,590,269,590,314,342,314,(001980264-H)
347,319,581,319,581,358,347,358,29,JLN SJ 17 ,
263,372,666,372,666,410,263,410,TMN SELAYANG JAYA,
285,421,643,421,643,461,285,461,68100 BATU CAVES,
363,469,568,469,568,504,363,504,SELANGOR
305,515,625,515,625,549,305,549,TEL : 03-61372830
279,562,649,562,649,595,279,595,GST ID : 001661329408
238,638,694,638,694,671,238,671,SIMPLIFIED TAX INVOICE

When comparing these values with the image, the reader can identify that each row in the annotation file corresponds to a text blob in the image called a Bounding Box, often ordered from top to bottom. First eight integers in a row are x, y coordinates of four vertices of the bounding box calculated relative to the top left corner of the image, and the rest are text inside it.

tan woon yann

BOOK TA .K (TAMAN DAYA) SDN BHD
789417-W
NO.5: 55,57 & 59, JALAN SAGU 18,
TAMAN DAYA,
81100 JOHOR BAHRU,
JOHOR.

Document No : TD01167104
Date : 25/12/2018 8:13:39 PM
Cashier : MANIS
Member :

CASH BILL

CODE/DESC		PRICE	Disc	AMOUNT
QTY		RM		RM
9556939040118	KF MODELLING CLAY KIDDY FISH			
1 PC	*	9.000	0.00	9.00

Total : 9.00
Rour ding Adjustment : 0.00
Round d Total (RM): 9.00

Cash 10.00
CHANGE 1.00

9.00

GOODS SOLD ARE NOT RETURNABLE OR EXCHANGEABLE
[illegible]
[illegible]

THANK YOU
PLEASE COME AGAIN !

Figure 3: Scanned receipt image with a barcode and a handwritten value

3.4 Input

- For system development/training, a bunch of pre-processed scanned invoice/receipt images and definitely their corresponding annotations (labels) are input to the system.
- For testing/inference, one item from the pre-processed scanned invoice/receipt images is given to the system at a time. Note that the corresponding annotation file of the image is not needed in this step.

3.5 Output

- From system development/training, mainly an inference tool, weights file (model) with training/validation results from Machine Learning modules, and image labeling tool under pre-processing are generated.
- From testing/inference, an annotation (labels) corresponding to input image is emitted in a middle stage and finally the extracted information as key-value pairs are given.

3.6 Technologies Used

3.6.1 Artificial Intelligence

Last few decades Artificial Intelligence (AI) has spread around the technology world like a superhero opening gates for many blocking problems. Intention of the AI is to obtain a human-like intelligence for delegating the advanced tedious tasks to machines. AI can be categorized into two main classes, Symbolic AI and Non-Symbolic AI. Typical rule-based techniques can be seen as Symbolic AI which have been used to develop many expert systems over many years. Symbolic AI can present what conditions were considered when drawing a conclusion. On the other hand, Non-Symbolic AI cannot provide any facts on how it came into a particular conclusion. But it can do the intended task successfully, hence it can be seen as a muscle memory. This research project mainly relies on following AI technologies.

3.6.2 Machine Learning

Machine Learning (ML) is the most hot topic in today's AI world, which is a subcategory of AI. It is mainly used to implement Non-Symbolic AI systems that a machine can be trained

using examples, such as a human is trained to ride a bicycle through trial and error. There are many ML algorithms namely Linear regression, Logistic regression, Artificial Neural Network (ANN), Decision tree, SVM algorithm, Naive Bayes algorithm, KNN algorithm, K-means, Random forest algorithm, Dimensionality reduction algorithms etc. An algorithm is not suitable for all the problems, it should be selected wisely based on the context.

3.6.3 Artificial Neural Network (ANN)

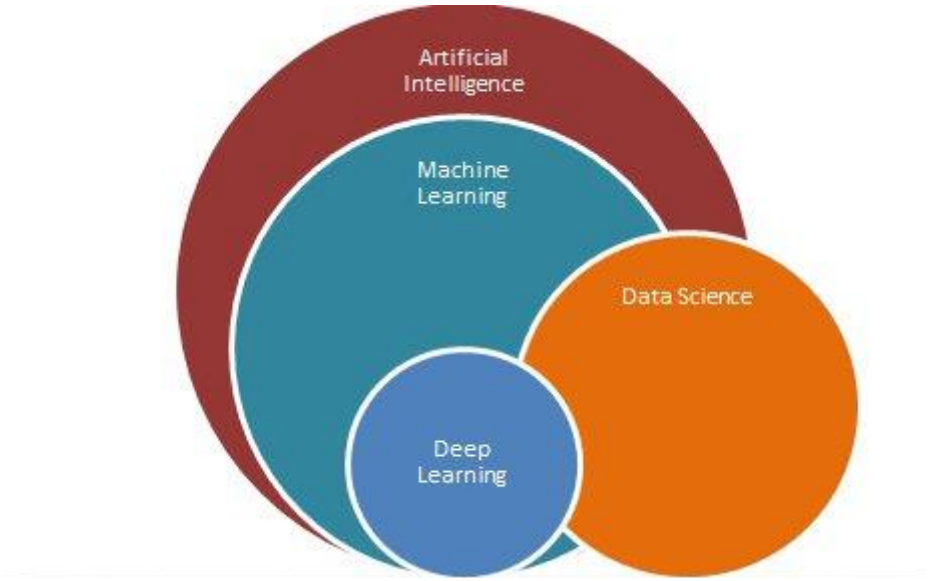


Figure 4: ANN/DNN is a subset of ML | Image from https://www.quora.com/

Artificial Neural Network (ANN) is a subset of Machine Learning (ML) and also called Deep Neural Network (DNN) also as shown in Figure 4. It is influenced by the biological brain. In the neurological aspect it is simply a network of neurons. Neuron is the smallest unit of a neural network. Usually when a neuron fires subsequent neurons connected to it also fire, then another set because of them. That way it forms a network of layers.

3.6.4 Perceptron

Perceptron is the smallest unit in an ANN which is analogous to a biological neuron (Staudemeyer and Morris, 2019) as shown in Figure 5. Perceptron usually has several inputs including a biased input that are weighted sum (linear regression) and may be further thresholded to get a binary output (logistic regression) as the formula presented in Figure 5. That last function is actually called the activation function and there are few such as Sigmoid

and Relu for binary classification, and Softmax for multi-class classification problems. For training a perceptron for classification, for each record in the provided dataset, the mentioned formula is executed, then output is compared to the ground truth label value and the difference between them is calculated as the error. This process executes many rounds (epochs) until the error reaches near zero by updating the weight values as appropriate. Error calculation logic is also called loss function and there are several variants such as Mean Absolute Error, Mean Squared Error, Cross-Entropy etc. When whole dataset is used in each epoch, it is called Batch Gradient Descent (GD), if one random record is used for an epoch it is called Stochastic Gradient Descent (SGD), and if a subset of dataset is used for an epoch it is called Mini Batch Gradient Descent.

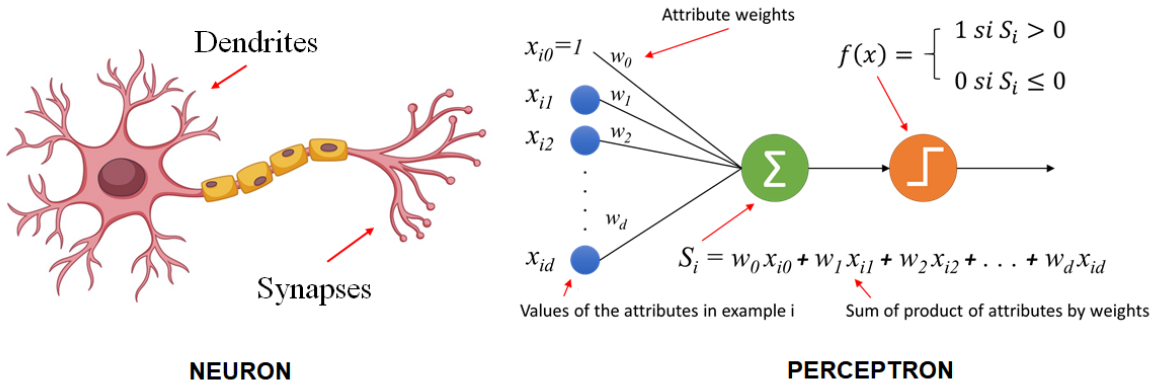


Figure 5: Biological neuron vs perceptron | Image from https://inteligenciafutura.mx

3.6.5 Backpropagation

ANN or DNN are also called multi-layer perceptrons. It contains three types of layers namely input, hidden and output. There can be more than one hidden layer as shown in Figure 6. Moreover they are called feed forward networks also since neurons are fired from input to output side only. But when error on output layers is calculated it is reflected to the backward layers one by one by updating the weight values. It is not simple as in a single perceptron. It uses chain rule differentiation in mathematics to achieve this goal.

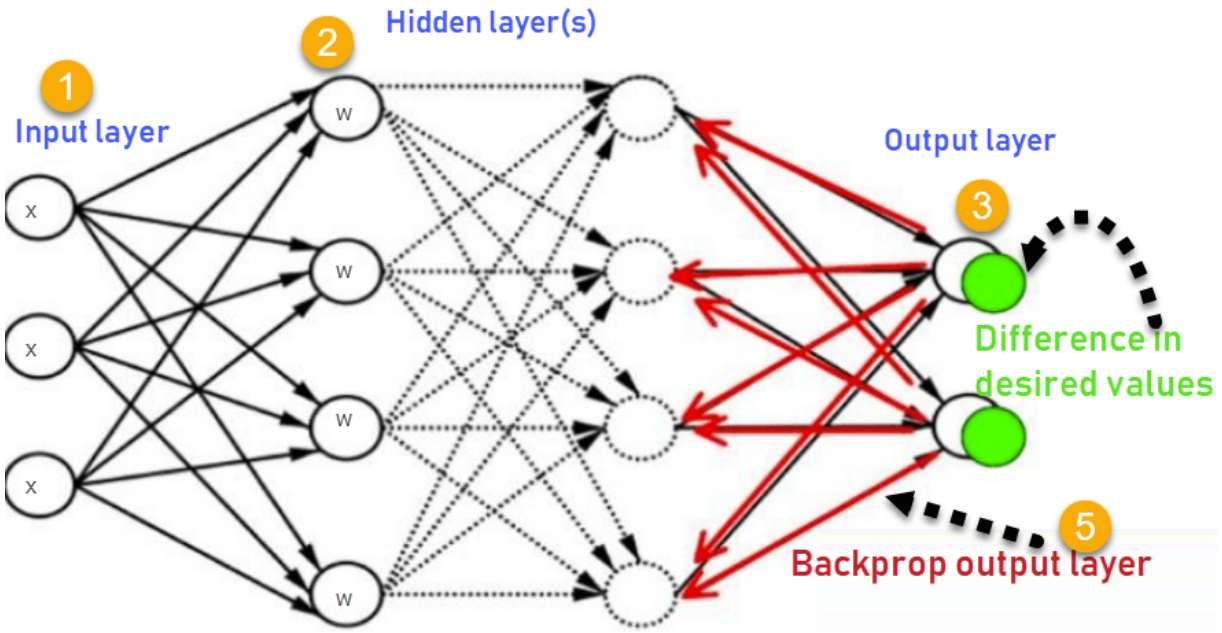


Figure 6: Multi-layer perceptrons | Image from https://www.niser.ac.in/

3.6.6 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a special type of ANN mainly used in computer vision algorithms to extract features from images (Du, 2018). It consists of several convolution layers with pooling layers and finally followed by a fully connected one or few ANN layers. Initial convolution layers capture small features then consecutively converge into large features, finally they are classified by the fully connected layers at the end. Pooling layers are used to reduce the output space from one convolution layer prior to the next convolution layer. Figure 7 below shows an example CNN.

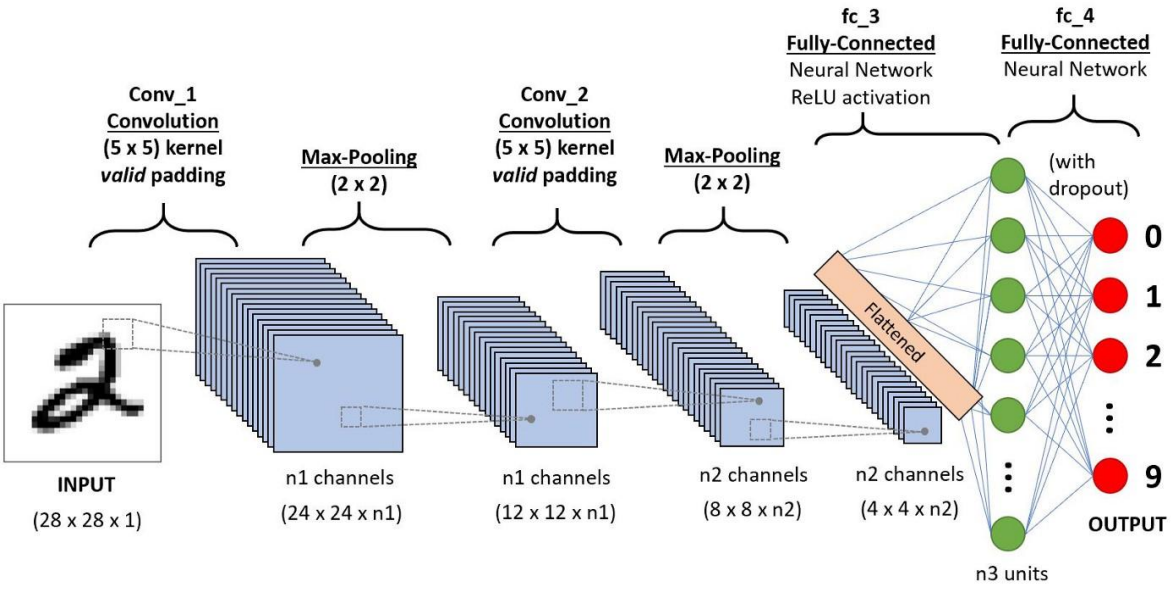


Figure 7: Example Convolutional Neural Network (CNN) | Image from https://towardsdatascience.com/

3.6.7 You Only Look Once (YOLO)

You Only Look Once (YOLO) is an object detection algorithm that falls under the CNN family (Du, 2018). It is capable of detecting multiple objects in an image, collecting their location details and classifications. Rather than using the sliding window technique it breaks the image into a grid and applies the detection/classification function to all the cells at once. Furthermore it uses a technique called Intersection over Union (IoU) to eliminate multiple detections for an image and chooses the one with highest confidence value. Some more details will be in the implementation section. Figure 8 shows the network architecture of YOLOv5.

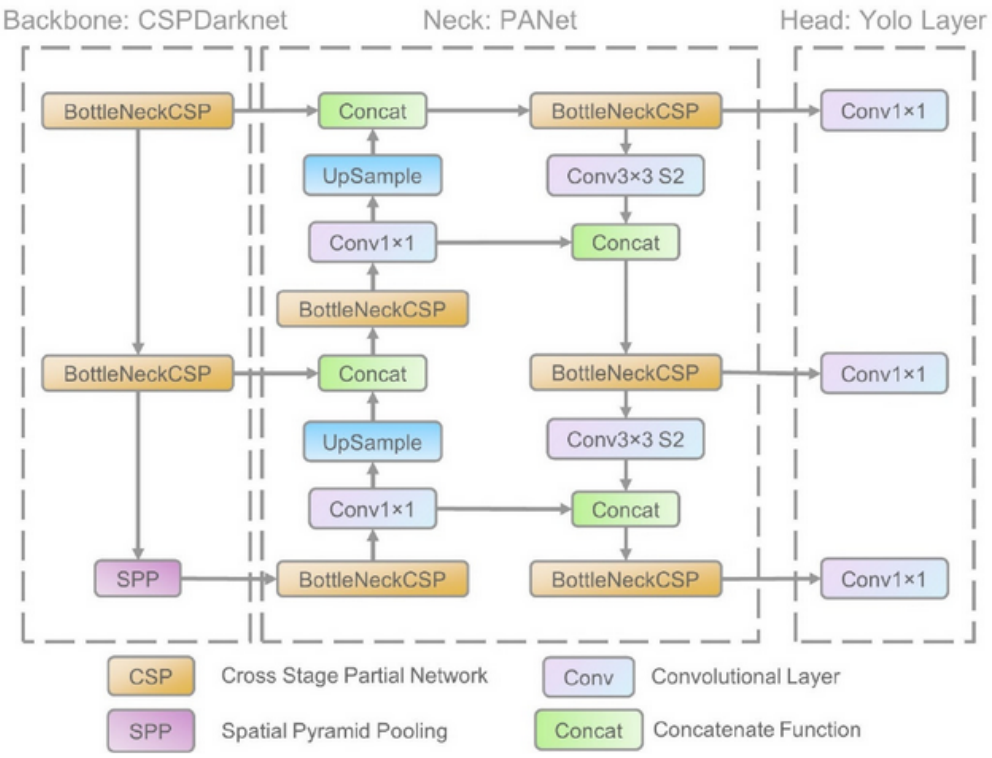


Figure 8: Network architecture for YOLO v5 | Image from https://iq.opengenus.org/yolov5/

3.6.8 Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) is a type of an ANN that a neuron has a cyclic input from its own output. That way it maintains an internal state (memory) and can affect subsequent inputs in a temporal manner. They are mostly used for tasks such as unsegmented, connected handwriting recognition or speech recognition (Staudemeyer and Morris, 2019). Figure 9 below depicts an example RNN.

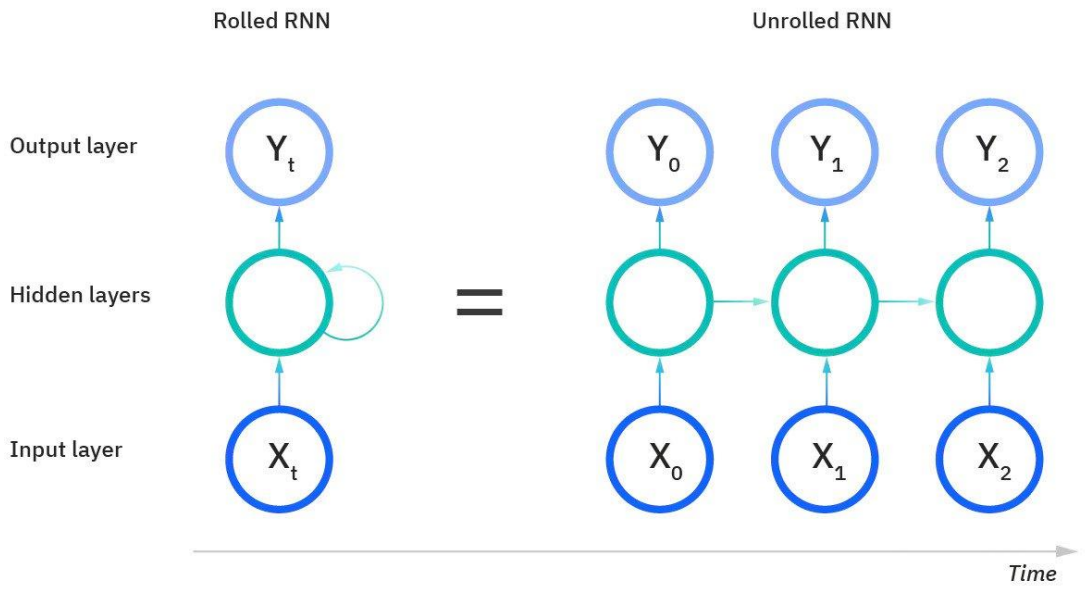


Figure 9: Example Recurrent Neural Network (RNN)

3.6.9 Long short-term memory

Long short-term memory (LSTM) is an extended version of RNN to occupy both long and short time memories (Staudemeyer and Morris, 2019). Unlike a typical RNN which processes a single data point, LSTM can process a sequence of data points like character stream, speech or video. Possible use cases it is utilized for are unsegmented, connected handwriting recognition, speech recognition, machine translation, robot control, video games, and healthcare. Figure 10 shows the building block of LSTM. In this research project LSTM comes into play where it is the backbone of the OCR algorithms.

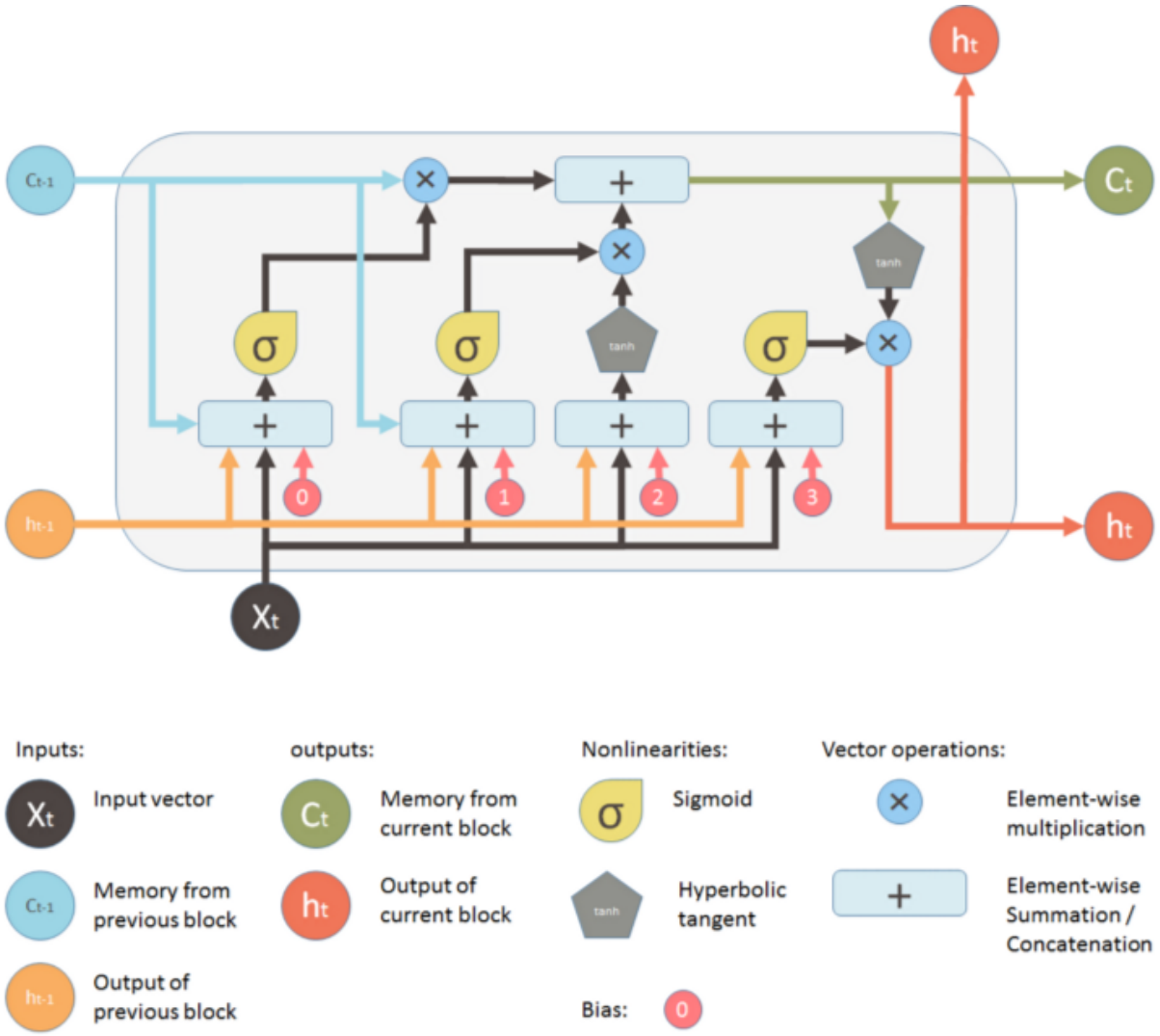


Figure 10: LSTM building block

3.7 Process

Following are steps of the proposed process for building a scanned invoice/receipt reader. Figure 11 below visualizes the process in detail while Figure 12 shows a simplified version of it. Inferring a test image using the developed model is depicted in Figure 13. And Figure 14 shows the data pre-processing flow.

1. Pre-process original dataset until it matches for training/validation/testing algorithms.
2. Train a deep neural network model using pre-processed data to detect/localize and classify texts in the image.
3. OCR detected text blobs to extract data out of them.
4. Process classification, textual and spatial information collected from above steps to correlate text items into related key-value pairs

5. Do validations to measure the speed and accuracy of the steps 2 to 4 separately.
6. Repeat steps from 2 to 5 until the system reaches a higher performance level.
7. Create an XML document with information extracted from previous steps.

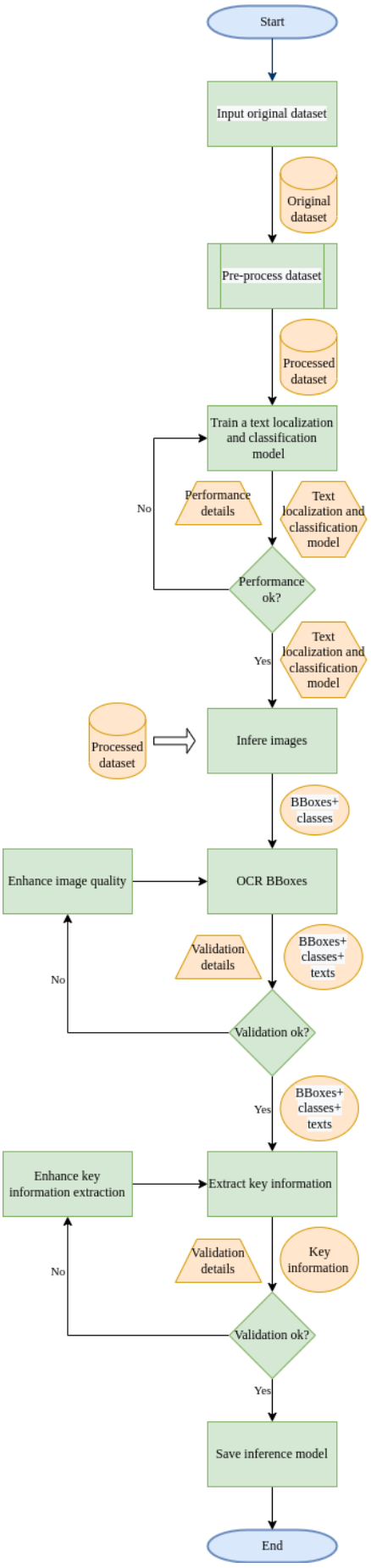


Figure 11: Process of developing the inference model - simplified version

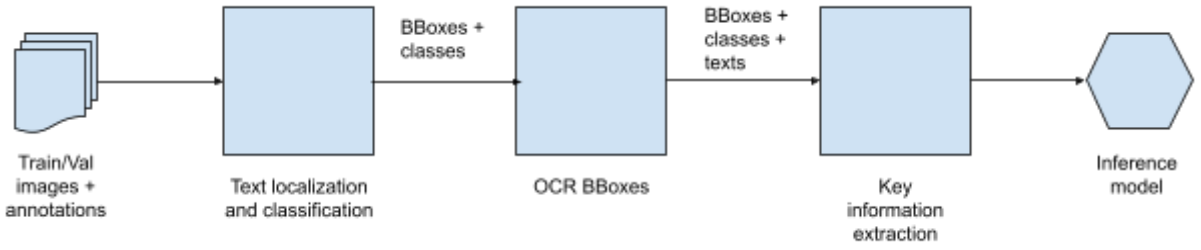


Figure 12: Process of developing the inference model - simplified version

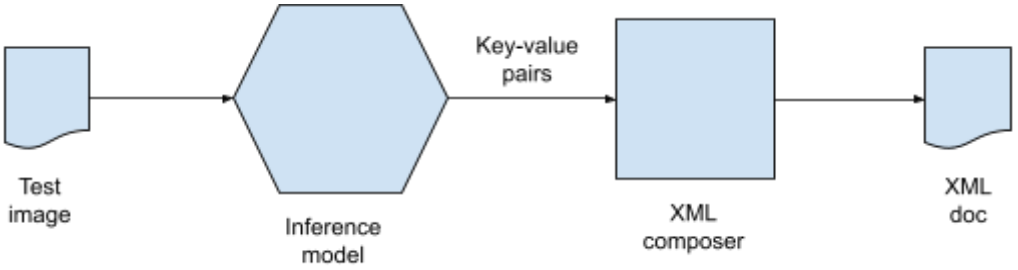


Figure 13: Inferencing an image

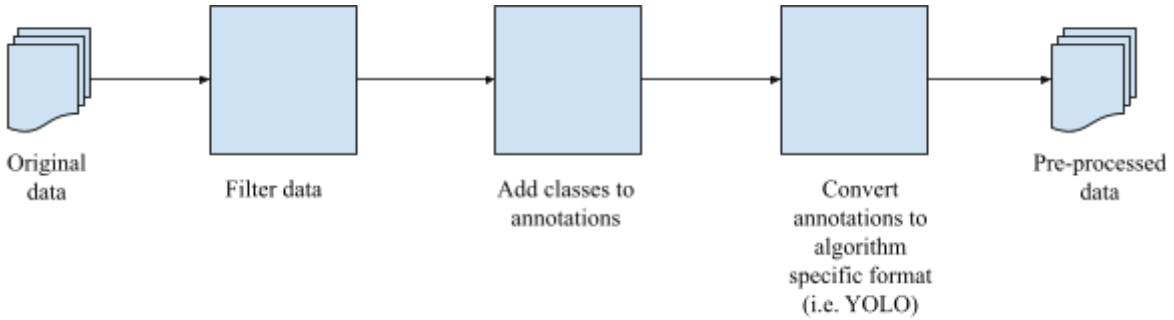


Figure 14: Data pre-processing

3.8 Design

3.7.1 Design of Image Labeling Tool

Design diagram in Figure 15 shows the architecture of the image labeling GUI tool described in section 3.8.1.2 Labeling Bounding Boxes.

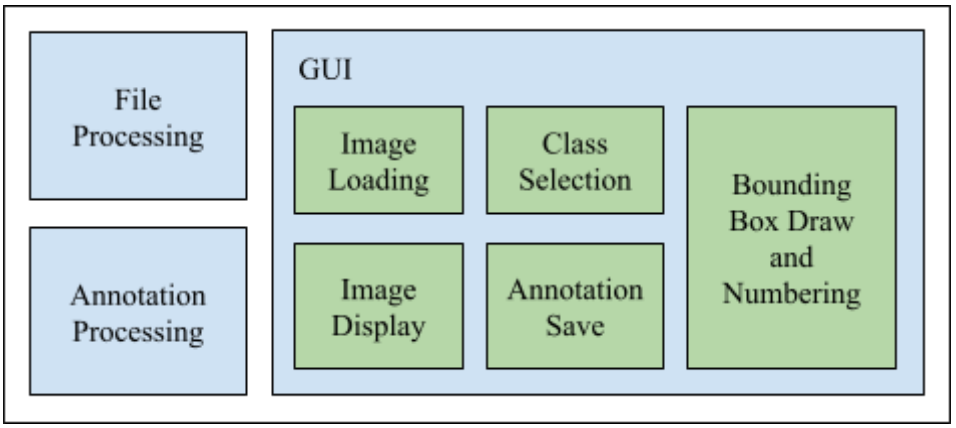


Figure 15: Architecture of the image labeling GUI tool

3.7.2 Design of Scanned Receipt Reading System

Following Figure 16 is the high level design architecture of the proposed artificial intelligence system for scanned receipt reading which presents the software components used inside. The main components come into play are Text Localization and Classification, Image Enhancement, OCR, Key Information Extraction, File Processing and XML Doc Generation.

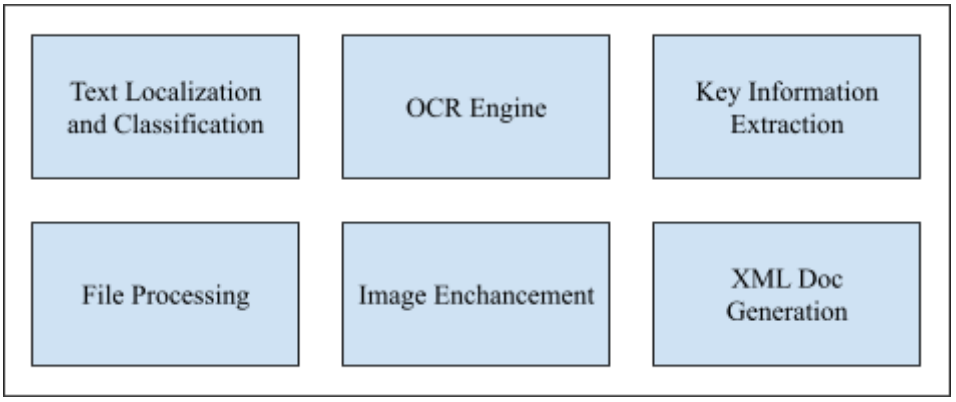


Figure 16: Architecture of the proposed scanned invoice/receipt reader

3.9 Implementation

Implementation lies on several steps namely pre-process data, training a deep neural network for text localization and classification, image enhancement for OCR, extracting text using OCR, processing extracted text to formulate semantic key-value pairs, and finally creating an XML document with drawn information. The baseline technology stack used for implementation works is Anaconda (“Anaconda | Anaconda Distribution,” n.d.), the Python

Data Science environment with Jupyter Notebook IDE ("Project Jupyter," n.d.) runs on a local machine. Implementation source codes and the results can be found in the Github repository https://github.com/binaradarsha/InvoiceReader. It is still a private repository and can provide permissions upon request.

3.8.1 Pre-processing Data

Data pre-processing is the first and most important task in any Data Science project since the dataset can be imbalanced and consists of incorrect, incomplete and low quality items, or in a format that dataset creators designed. In this case the SROIE dataset is also not very clean and not in the exact format that we expect. Therefore the author had to apply a few pre-processing steps namely filtering data, labeling bounding boxes and converting annotations to format required by the machine learning algorithm used for text localization and classification.

3.8.1.1 Filtering Data

Original SROIE dataset includes more annotation files than the amount of image files. To remove those redundant items images were taken first and corresponding labels were listed accordingly. Related image file and the annotation file were mapped by both using the same name except the file extension. Before filtering there were 772 training images and after filtering it was reduced to 704 records. Corresponding image and annotation text files are shown in Figure 17 below, and can map two sides with the file name. A Python script was written for this step and it can be found in Appendix A.

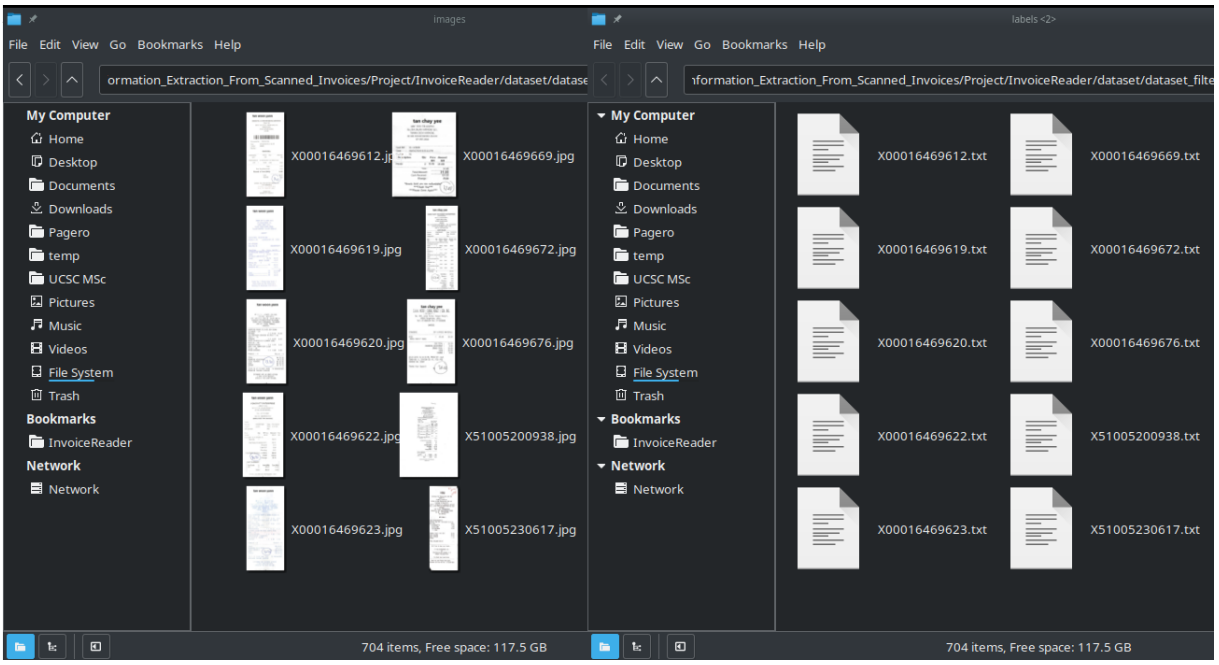


Figure 17: Dataset after filtering, images are in left side and annotation files are in right side

3.8.1.2 Labeling Bounding Boxes

In SROIE dataset originally bounding boxes are not labeled as seen in Figure 2: Scanned receipt image followed by its annotation file content. Hence had to label bounding boxes using a new GUI tool shown in Figure 18 below and was developed by the author using Appendix B Python script. When an image is opened and loaded to the display panel in the left side its corresponding annotation file also is loaded underneath and using records inside, it draws red coloured bound boxes over text blobs, also assigns an zero based index for better referencing. All those bounding boxes drawn are clickable. When clicking on a bounding box it shows the corresponding index and containing text on top of the right hand side panel for confirming which box was really clicked now. List view below that panel contains all the possible classes or labels that a bounding box can be assigned to. Currently there are 227 identified classes that can be found in Appendix E considering a granular level of classification biased to this SROIE dataset. After selecting a label for a bounding box by clicking an item on list, “Save Labels” button can be pushed to save the selected labels to the same annotation file. The annotation file content appears after Figure 18 shows the assigned labels of all the bounding boxes of the image displayed in Figure 18. Labels are placed between the coordinate values and containing text, encapsulated by <<< >>> angles brackets. However labeling bounding boxes is a tedious manual task, which makes it more harder when

there are several formats of invoices included in the dataset. Thus the author was able to label only 40 records within the short time period and it was used in Approach 3 below. For Approach 4 only a single class is used for all the bounding boxes.

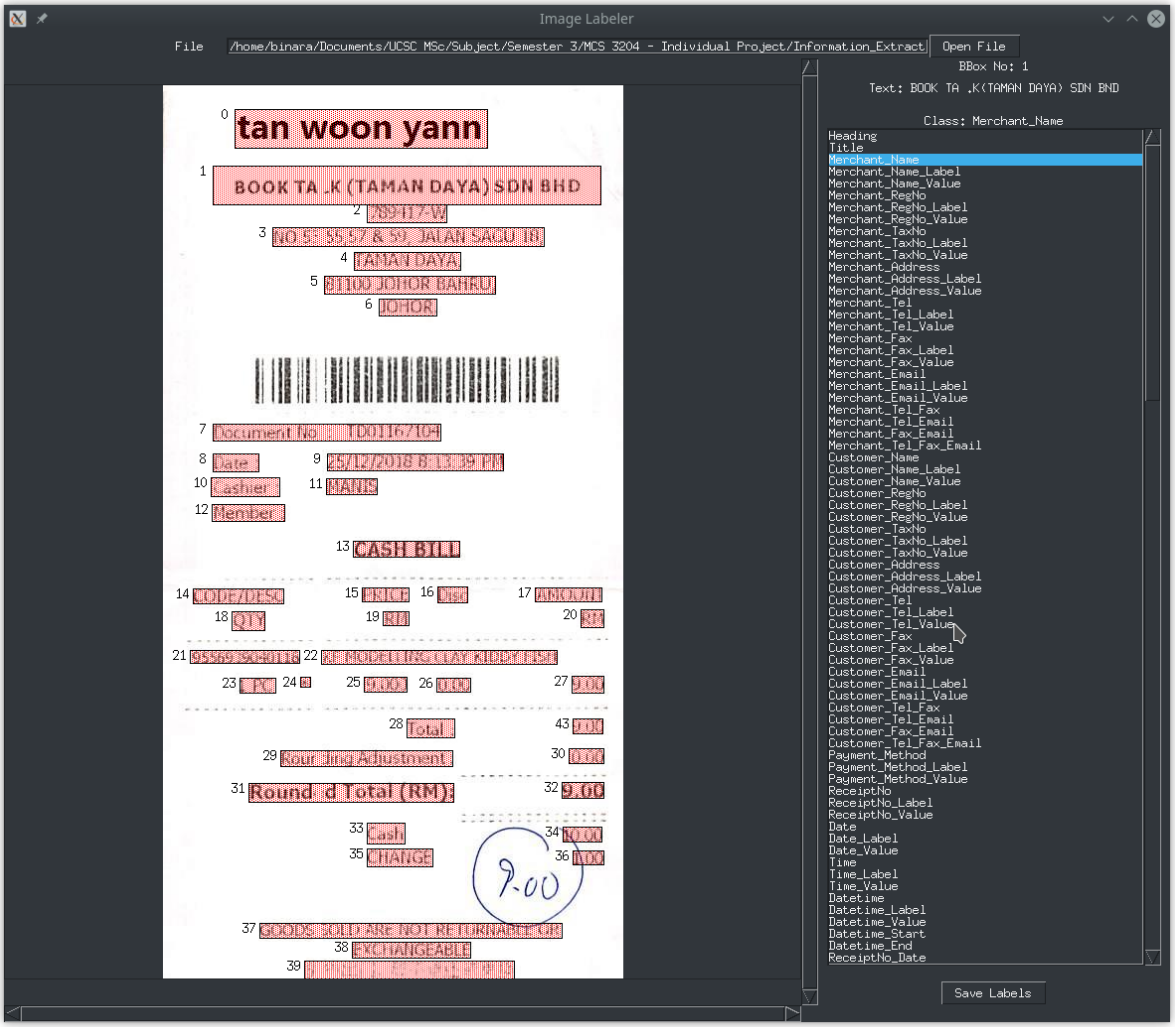


Figure 18: Bounding box labeling GUI tool

72,25,326,25,326,64,72,64,<<<Heading>>>,TAN WOON YANN

50,82,440,82,440,121,50,121,<<<Merchant_Name_Value>>>,BOOK TA .K(TAMAN DAYA) SDN BND

205,121,285,121,285,139,205,139,<<<Merchant_RegNo_Value>>>,789417-W

110,144,383,144,383,163,110,163,<<<Merchant_Address_Value>>>,NO.53 55,57 & 59, JALAN SAGU 18,

192,169,299,169,299,187,192,187,<<<Merchant_Address_Value>>>,TAMAN DAYA,

162,193,334,193,334,211,162,211,<<<Merchant_Address_Value>>>,81100 JOHOR BAHRU,

217,216,275,216,275,233,217,233,<<<Merchant_Address_Value>>>,JOHOR.

50,342,279,342,279,359,50,359,<<<ReceiptNo>>>,DOCUMENT NO : TD01167104

50,372,96,372,96,390,50,390,<<<Datetime_Label>>>,DATE:

165,372,342,372,342,389,165,389,<<<Datetime_Value>>>,25/12/2018 8:13:39 PM

48,396,117,396,117,415,48,415,<<<Cashier_Label>>>,CASHIER:

164,397,215,397,215,413,164,413,<<<Cashier_Value>>>,MANIS

49,423,122,423,122,440,49,440,<<<Member>>>,MEMBER:

191,460,298,460,298,476,191,476,<<<Title>>>,CASH BILL

30,508,121,508,121,523,30,523,<<<Table_Head_Code_Description>>>,CODE/DESC

200,507,247,507,247,521,200,521,<<<Table_Head_Price>>>,PRICE

276,506,306,506,306,522,276,522,<<<Table_Head_Discount>>>,DISC

374,507,441,507,441,521,374,521,<<<Table_Head_Amount>>>,AMOUNT

69,531,102,531,102,550,69,550,<<<Table_Head_Quantity>>>,QTY

221,531,247,531,247,545,221,545,<<<Currency>>>,RM

420,529,443,529,443,547,420,547,<<<Currency>>>,RM

27,570,137,570,137,583,27,583,<<<Table_Value_Code>>>,9556939040116

159,570,396,570,396,584,159,584,<<<Table_Value_Description>>>,KF MODELLING CLAY KIDDY FISH

77,598,113,598,113,613,77,613,<<<Table_Value_Quantity>>>,1 PC

138,597,148,597,148,607,138,607,<<<Multiplication>>>,*

202,597,245,597,245,612,202,612,<<<Table_Value_Price>>>,9.000

275,598,309,598,309,612,275,612,<<<Table_Value_Discount>>>,0.00

411,596,443,596,443,613,411,613,<<<Table_Value_Amount>>>,9.00

245,639,293,639,293,658,245,658,<<<Total_Amount_Label>>>,TOTAL:

118,671,291,671,291,687,118,687,<<<Rounding_Adjustment_Label>>>,ROUR DING ADJUSTMENT:

408,669,443,669,443,684,408,684,<<<Rounding_Adjustment_Value>>>,0.00

86,704,292,704,292,723,86,723,<<<Total_Amount_Rounded_Label_Currency>>>,ROUND D TOTAL (RM):

401,703,443,703,443,719,401,719,<<<Total_Amount_Rounded_Value>>>,9.00

205,744,243,744,243,765,205,765,<<<Cash_Label>>>,CASH

402,748,441,748,441,763,402,763,<<<Cash_Value>>>,10.00

205,770,271,770,271,788,205,788,<<<Change_Label>>>,CHANGE

412,772,443,772,443,786,412,786,<<<Change_Value>>>,1.00

97,845,401,845,401,860,97,860,<<<Extra>>>,GOODS SOLD ARE NOT RETURNABLE OR

190,864,309,864,309,880,190,880,<<<Extra>>>,EXCHANGEABLE

142,883,353,883,353,901,142,901,<<<Extra>>>,***

137,903,351,903,351,920,137,920,<<<Extra>>>,***

202,942,292,942,292,959,202,959,<<<Extra>>>,THANK YOU

163,962,330,962,330,977,163,977,<<<Extra>>>,PLEASE COME AGAIN !

412,639,442,639,442,654,412,654,<<<Total_Amount_Value>>>,9.00

3.8.1.3 Converting Annotations to Format Required by the Text Detection/Classification Algorithm

As grasped from literature review YOLO is the best objection detection and classification algorithm used in the industry today. Considering text also is some form of object YOLO should be able to be used for text detection and classification too. Thus annotations were converted to YOLO specific format shown below and done using Python scripts Appendix C for multi-class annotations and Appendix D for single-class annotations. Zero based indexed class/label list can be found in Appendix E as mentioned above. Following is the YOLO format for a bounding box and it is illustrated visually in Figure 19.

Class_index BBoxCenterX BBoxCenterY BBoxWidth BBoxHeight

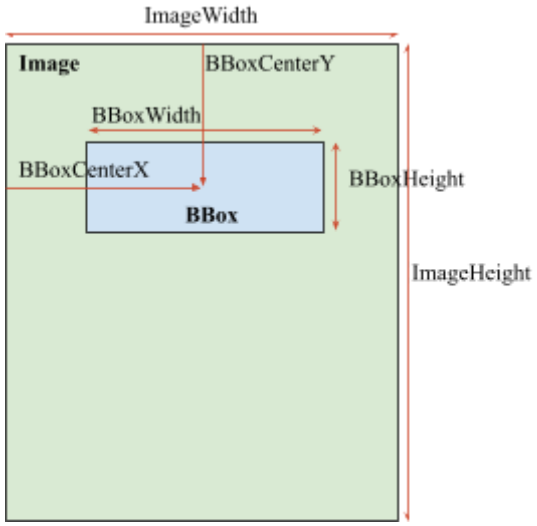


Figure 19: Visual representation of YOLO Annotation values

Very important point to note here is that all the values in the above format are numeric. Class_index is a zero based integer while the remaining four values are floating points. BBoxCenterX and BBoxWidth are divided by ImageWidth, and BBoxCenterY and BBoxHeight are divided by ImageHeight to take those values between zero and one. It is called normalization and done due to training time can be reduced enormously by using small numbers.

Following are a few multi-class annotation lines retrieved through this conversion process.

114 0.4298056155507559 0.043928923988154 0.5485961123110151 0.0384995064165844
91 0.5291576673866091 0.10019743336623889 0.8423326133909287 0.0384995064165844
17 0.5291576673866091 0.12833168805528133 0.17278617710583152 0.017769002961500493

And following are a few lines of a single-class annotation.

0 0.4298056155507559 0.043928923988154 0.5485961123110151 0.0384995064165844
0 0.5291576673866091 0.10019743336623889 0.8423326133909287 0.0384995064165844
0 0.5291576673866091 0.12833168805528133 0.17278617710583152 0.017769002961500493

3.8.2 Text Detection and Classification

For this task four approaches were examined as described below.

3.8.2.1 Approach 1: Creating a CNN on Scratch

Since the base algorithm family ideal for object detection is Convolutional Neural Networks (CNNs), the author was initially trying to build a custom made CNN from scratch with fewer convolutional layers and neurons. When not getting any positive results, examined well established object detection algorithms thus it was seeming a huge deep neural network should be used and creating our own one requires years of tough research work and more resources should be acquired. Hence had to go for a well established object detection algorithm considering the limited scope and timeframe.

3.8.2.2 Approach 2: Applying OCR on Entire Image

Before going to a well established object detection algorithm, needed to examine how OCR algorithms behave here, doing text detection and extraction altogether for an entire image. Tesseract OCR engine version 4 was used in this preliminary investigation. When the image in Figure 2 is passed to Tesseract it will output below content. It is clear that it extracts texts

line by line which may include several fields together, hence bit hard to break apart, specially the invoice item columns. Thus using only an OCR engine is not robust because of its low processing capability.

SAINT HEART PASTRY
(001980264-H)
29,JLN SJ 17,

TMN SELAYANG JAYA,
68100 BATU CAVES,
SELANGOR
TEL : 03-61372830
GST ID : 001661329408

SIMPLIFIED TAX INVOICE

CASH
Receipt#: ©S00254837 Table: 46
Staff: AISHAH Date: 26/03/2018
Cashier: AISHAH Time: 10:56:00
Description Q RM) RM Tax
JUMBO SAUSAGE CHEESE 1 ue 3.10 SR
| JUMBO SAUSAGE CHEESE 1 3.10 3.10 SR
| GARLIC CHEESE 1 2.00 2.00 SR
otal: :
Total Sales (Excluding GST) : 7.74
| Discount : 0.00
Service Charge : 0.00
TotalGST : 0.46
Rounding : 0.00
Total Sales (Inclusive of GST) 8.20
CASH 8.20

CHANGE 0.00

GST SUMMARY
TaxCode % Amt(RM)

Tax (RM)
SR 6 7.74 0.46
Total: 7.74 0.46

GOODS SOLD ARE NOT RETURNABLE, THANK YOU.
RE-PRINT

la bende

3.8.2.3 Approach 3: Training a Text Detection Algorithm with Multi-Class Annotations

In this approach a deep neural network model was trained using pre-processed data with multi-classes to detect bounding boxes and their classes using YOLOv5 algorithm with the below commands. It was executed with 40 records as mentioned in section 3.8.1.2 Labeling Bounding Boxes and 151 classes to get a more granular level of detail. Then it was halted because of receiving a low accuracy because of the less number of records thus requiring many days to train to achieve a high accuracy. More about that will be discussed in the Results section later. However theoretically if this approach could be succeeded with a high volume of dataset and robust computing resources, both Task 1: Text Localization and Classification and Task 3: Key Information Extraction are accomplished in single-shot to a greater extent. Therefore more weight is added to the Task 1 and a very tiny portion towards Task 3 as shown in Figure 20.

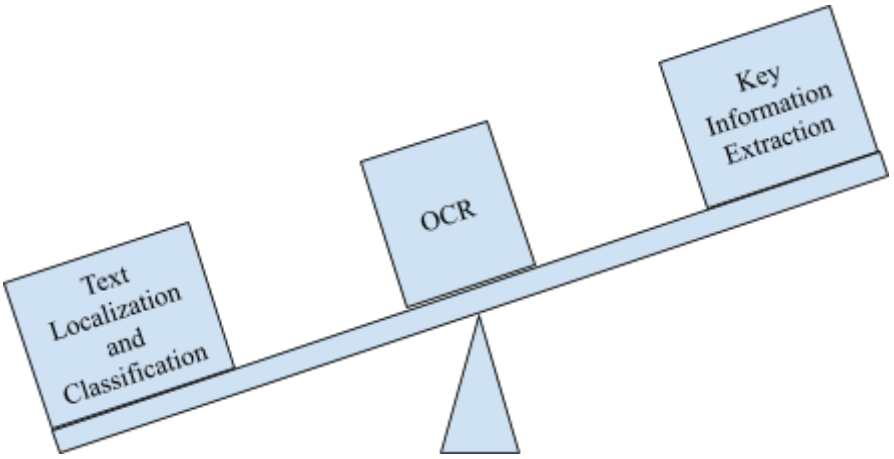


Figure 20: More weight towards Text Localization and Classification when doing multi-class training

Retrieve YOLOv5 algorithm source code from GitHub repository and install required dependencies.

```
git clone https://github.com/ultralytics/yolov5  # clone
pip install -r yolov5/requirements.txt  # install
```

Train the neural network for text detection/localization and classification.

```
python yolov5/train.py --img 640 --batch 4 --epochs 20 --data dataset-151.yaml --weights yolov5/yolov5s.pt
```

Parameters used in the command:
yolov5/train.py - YOLO training script written in Python
--img - Image size used inside the training, no matter what is the original image size is
--epochs - No of rounds
--batch - Actual batch size for one epoch is retrieved by dividing the total records count by this given number
--data - Configuration file path that includes file paths to training/validation/testing data, used classes list and classes count
--weights - YOLO weights file that should be based on (since transfer learning)

3.8.2.4 Approach 4: Training a Text Detection Algorithm with Single-Class Annotations

This is the continued approach in this research work that was fit into the having short time period and resources. In this way a deep neural network model was trained using pre-processed data with single-class to detect bounding boxes only, in other words classification was not involved. Therefore a more weight was added to Task 3: Key Information Extraction as shown in Figure 14. Model was trained with 704 records and gained accuray over 95 percent, more details will be in the Results section.

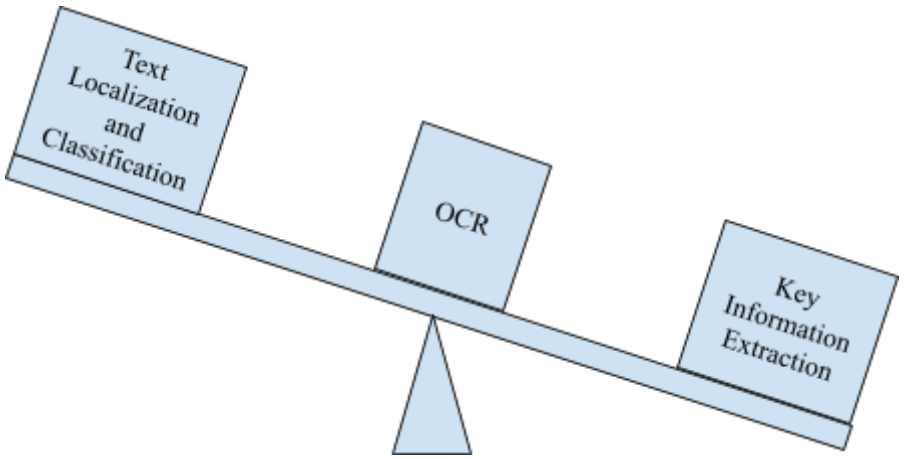


Figure 21: More weight towards Key Information Extraction when doing single-class training

Training command is very similar to the one in 3.8.2.3 Approach 3 above, except the number of epochs is 65 and the data configuration file contains corresponding single-class dataset paths and one class.

python yolov5/train.py --img 640 --batch 4 --epochs 65 --data dataset.yaml --weights yolov5/yolov5s.pt

Following is part of the training output. It shows that running epochs from 0 to 64, starting from a low accuracy and reaches to a higher accuracy. It has taken 3 hours to complete the training and has saved the trained model, the weight files to the disk.

Image sizes 640 train, 640 val
Using 4 dataloader workers
Logging results to **yolov5/runs/train/exp8**

Starting training for 65 epochs...

```
    Epoch   gpu_mem       box       obj       cls    labels  img_size
     0/64    0.883G     0.135    0.1911         0       291       640: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all        704      37554     0.0739      0.157     0.0406    0.00868

    Epoch   gpu_mem       box       obj       cls    labels  img_size
     1/64    0.996G     0.115    0.2437         0       344       640: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all        704      37554      0.322      0.368      0.253     0.0731
……..
Epoch   gpu_mem       box       obj       cls    labels  img_size
    63/64    0.996G   0.07035    0.2013         0       452       640: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all        704      37554      0.975      0.919      0.967      0.658

    Epoch   gpu_mem       box       obj       cls    labels  img_size
    64/64    0.996G   0.06866    0.2025         0       340       640: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all        704      37554      0.974      0.922      0.967      0.658

65 epochs completed in 3.104 hours.
Optimizer stripped from yolov5/runs/train/exp8/weights/last.pt, 14.3MB
Optimizer stripped from yolov5/runs/train/exp8/weights/best.pt, 14.3MB
```

Then using the trained model an scanned receipt image can be inferred using the following YOLO's ready made detect.py script or the custom Python code followed by.

```
python yolov5/detect.py --weights yolov5/runs/train/exp8/weights/best.pt --source ../../dataset/dataset_test/images/X51005719905.jpg --img 640

# *** PROGRAMATIC INFERENCE ***
# Loading model
model = torch.hub.load('ultralytics/yolov5', 'custom', 'yolov5/runs/train/exp8/weights/best.pt')
```

```
# Image path
img_path = f'../../dataset/dataset_test/images/X51005719905.jpg'
# Inference
results = model(img_path)
# Printing results
results.print()
df = results.pandas().xyxy[0]
print(df)
```

Following is the output of this essential Python code snippet. Table at the end of it shows the extracted bounding box details in format (index, xmin, ymin, xmax, ymax, confidence, class_index, class_text). Values index is the position in the original output matrix, xmin and ymin are coordinates of the top left corner, xmax and ymax are coordinates of the bottom right corner, confidence is how the model is sure about detecting the text. The above code snippet actually cannot be run standalone due to required library imports are not mentioned at this point. By executing the full code presented under Appendix F, the following textual output and its visual illustration on the inferred image that is shown by Figure 22 can be obtained. You can see that the following inference has completed in a total of 45.8 milliseconds.

```
image 1/1: 2079x936 87 Ts
Speed: 11.8ms pre-process, 31.3ms inference, 2.7ms NMS per image at shape (1, 3, 640, 320)

        xmin        ymin        xmax        ymax  confidence  class name  \
44  210.264252   211.671616  722.274475   257.040192    0.843771      0    T
79  332.542908   238.154755  612.257812   304.402985    0.407757      0    T
4   337.624969   269.689636  589.738831   309.250122    0.878084      0    T
53  338.616058   315.009491  582.568542   355.635803    0.832079      0    T
54  268.219757   366.780884  659.022644   409.151642    0.831085      0    T
..         ...         ...         ...         ...         ...    ...  ...
31  300.369354  1798.351074  413.538452  1837.271362    0.854160      0    T
57  513.473267  1800.498169  585.944458  1836.296875    0.827876      0    T
67  690.112183  1801.687256  760.652161  1838.426392    0.800087      0    T
71  103.586212  1954.370850  803.918091  1985.302002    0.785074      0    T
35  358.396851  2006.130737  562.030701  2048.257812    0.852010      0    T
```

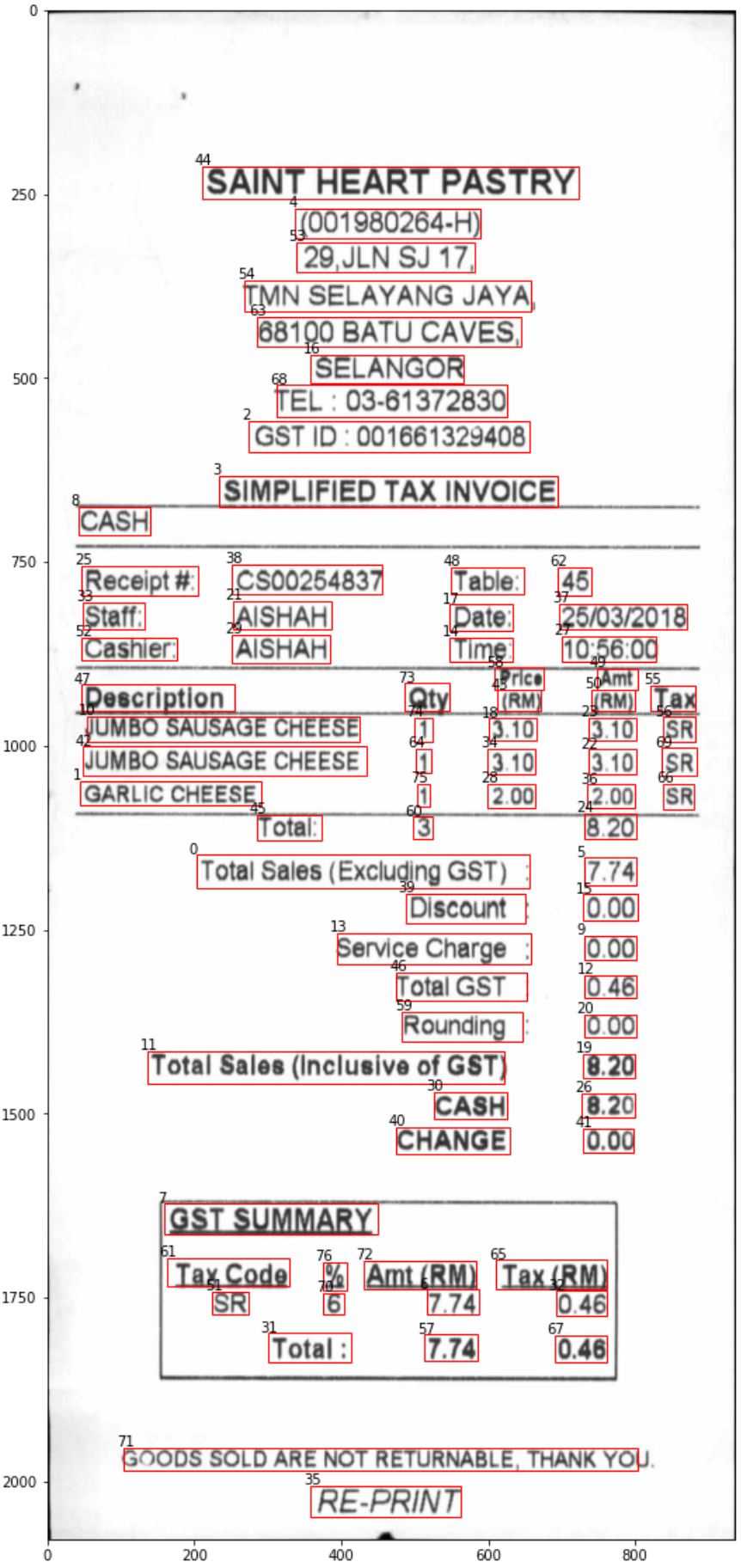


Figure 22: Drawing bounding boxes on the image extracted from inference

3.8.3 Extracting Text Using OCR

Either used Approach 3 or 4, OCR should be used with the same weight for extracting processable text from detected text blobs. To get this job done two tools were examined, Tesseract OCR engine (version 4) and Google Cloud Vision API. Actually Tesseract OCR engine is also a Google maintained software but free, open-source and standalone. Google Cloud Vision API is supposed to be more robust, but however requests from third party software (our script here) is throttled due to rate limits imposed as mentioned in its Quotas and Limits page https://cloud.google.com/vision/quotas. Therefore Tesseract was the most affordable and immediately productive tool that had to be used throughout the work.

To use Tesseract it should be first installed in the environment, follow the Tesseract installation guide on https://tesseract-ocr.github.io/tessdoc/Installation.html. The Python code below can be used to convert a text blob into a processable text. Here it uses a Python wrapper to access the Tesseract executable installed in the machine.

```
from pytesseract import pytesseract
path_to_tesseract = r'/usr/bin/tesseract'
pytesseract.tesseract_cmd = path_to_tesseract
text = pytesseract.image_to_string(image)
```

When using these OCR engines, initially extracted pure text blobs were directly passed and there were no positive results. Then conducted further research on this matter and found that accuracy of OCR results is highly dependent on quality of the image. Hence had to enlarge the image and apply a few image processing filters to enhance the quality of the image. Sharpen filter was used to reduce blur effect and sharpening the letter edges. Median filter was used to remove the salt and pepper noise. That way OCR accuracy increased, but should do further enhancements.

Full OCR source code including image enhancements can be found in Appendix G. By default it is enabled to use the Tesseract OCR engine, but if someone needs to try Google Cloud Vision API he/she can switch to the corresponding commented lines. However beforehand users have to enable relevant API in Google Cloud Console, obtain an application credential and set it locally.

Following is the OCR output for text blobs in Figure 22 image, in format (index ---> text). Corresponding text blob and OCR output can be mapped using the index.

*** OCR and store texts ***
44 ---> SAINT HEART PASTRY
4 ---> (001980264-H)
53 ---> 29.JLN SJ 17.
54 ---> TMN SELAYANG JAYA
63 ---> 68100 BATU CAVES.
16 ---> SELANGOR
68 ---> TEL : 03-61372830
2 ---> GST ID : 001661329408
3 ---> SIMPLIFIED TAX INVOICE
8 ---> CASH
38 ---> CS00254837
25 ---> Receipt #:
62 ---> 45
48 ---> Table:
21 ---> AISHAH
33 ---> Staff:
37 ---> 25/03/2018
17 ---> Date:
29 ---> AISHAH
27 ---> 10'56:00
52 ---> Cashier
14 ---> Time:
58 ---> Price
49 ---> Amt
73 ---> Qty
47 ---> Description
55 ---> Tax
50 ---> (RM)
43 ---> (RM)
10 ---> JUMBO SAUSAGE CHEESE
56 ---> SR

23 ---> 4.10

74 ---> 1

18 ---> 4.10

42 ---> JUMBO SAUSAGE CHEESE

69 ---> SR

64 ---> 1

34 ---> 4.10

22 ---> 3.10

1 ---> GARLIC CHEESE

75 ---> 1

28 ---> 2.00

66 ---> SR

36 ---> 2.00

24 ---> 8 20

45 ---> T otal:

60 ---> 3

0 ---> Total Sales (Excluding GST) |;

5 ---> 7.74

39 ---> Discount

15 ---> 0.00

13 ---> Service Charge .

9 ---> 0.00

46 ---> Total GST

12 ---> 0.46

59 ---> Rounding

20 ---> O00

11 ---> Total Sales (Inclusive of GST)

19 ---> 8.20

30 ---> CASH

26 ---> 8.20

40 ---> CHANGE

41 ---> 0.00

7 ---> GST SUMMARY

61 ---> Tax Code

65 ---> Tax (RM)

72 ---> Amt (RM)

76 ---> %

6 ---> 7.74

32 ---> 0.46

51 ---> SR

70 ---> 6

31 ---> Total:

57 ---> 7.74

67 ---> 0.46

71 ---> SO0DS SOLD ARE NOT RETURNABLE, THANK YO!

35 ---> RE-PRINT

3.8.4 Key Information Extraction

The last major step was to map the extracted texts from previous stages to generate related key-value pairs and table rows (receipt items) with header information. To achieve that captured textual, visual and spatial information were processed altogether. In the approach, first of all the image was logically broken into four sections. In other words, text blobs were grouped into four categories, namely Merchant Information, Receipt Information, Receipt Items and Totals as shown in Figure 23. It was done comparing the ymin coordinate values. Majority of the receipts have a title like “tax invoice”, “simplified tax invoice”, “cash bill” etc. Therefore any text blob having ymin less than the ymin of that title is categorized as a merchant information. Then most receipts have a table header starting with the column name called “description” or “item”. Thus any text blob having ymin between title’s ymin and description column header’s ymin is categorized as receipt information. Finally we get the ymin of the first occurring “total” field from the bottom section. Then text blobs having ymin between description header column’s ymin and total’s ymin are categorized as receipt items, and ones having ymin greater than or equal to total’s ymin are categorized into totals.

For extracting the merchant information, mainly pattern matching was used. Merchant reg no was captured through the regex pattern "\d+[-]{1}[A-Z]{1}". Tax no was extracted by checking whether it starts with a label similar to "GST ID", "GST NO" etc., same way telephone no starts with “TEL: ” and email starts with “EMAIL: ”. Merchant name is usually with bold effect, hence to identify it image blobs were binarized and checked whether the

calculated black and white pixel ratio is greater than the specified threshold. Texts in remaining blobs were combined together to form the merchant address.

From receipt information, receipt no and datetime are the values in interest. To find the receipt no, first have to track the receipt no label blob by checking whether the text starts with a term like "Receipt #", "Receipt No", "Invoice No" etc. When found it then needs to get the reference to the blob just next to it on the right hand side. It's done by searching blobs with ymin fall within the receipt no label blob's ymin±5 range, and xmin is greater than the receipt no label blob's xmax, and finally getting the first candidate's text. Then date and time may be presented as a single field or as separate fields, and sometimes both label and value may reside in the same blob or separate blobs. First detects such blobs by checking whether the text starts with "Datetime", "Date" or "Time". When found one, if both label and value are present in the same blob its is matched against the regex "\d{2}\/\d{2}\/\d{4} | \d{2}\:\d{2}\:\d{2}(\s*([AaPp][Mm]))? | (\d{2}\/\d{2}\/\d{4}\s*\d{2}\:\d{2}\:\d{2}(\s*([AaPp][Mm])))" which works for both combined and separate date and time values. If the label and value are in separate blobs, first need to find the label blob by checking whether the text starts with "Datetime", "Date" or "Time". Then get the blob presented in front of it in the same way that receipt no value was taken out. Finally have to merge date and time values if they were presented in different blobs.

Extracting values from receipt items table rows is a bit tricky since have to process information in both horizontal and vertical axes. In this case, first need to find the description table header blob by checking whether the text starts with a name like "description", "item" etc. Then should find description values from all the rows whose xmin lies between the description header blob's xmin±10 range and ymin is greater than the description header blob's ymax, and ymin is less than total blob's ymin too. Thereafter retrieved description values list is iterated and for each description value its row siblings are searched compared to its ymin, and candidate's xmin lies between xmin ranges of previously taken quantity, unit price and amount header blobs.

Finally for totals, first their label blobs are tracked, then text from blobs in front of them are retrieved correspondingly, following the same approaches as used above. Source code written to implement this logic can be found in Appendix H and the output shown below is for the

receipt in Figure 23. Nevertheless the logic is written so far using the Figure 23 receipt format as the baseline, for other formats it may need to be slightly adjusted.

*** Final result ***

--- Merchant Information ---

Name = SAINT HEART PASTRY
Address = 29 JLN SJ 17., TMN SELAYANG JAYA, 68100 BATU CAVES., SELANGOR
Reg No = 001980264-H
Tax No = 001661329408
Email =
Tel = 0361372830

--- Receipt Information ---

Receipt No = 1500254837
Datetime = 25/03/2018 10:56:00

--- Receipt Items ---

	description	qty	price	amount
0	JUMBO SAUSAGE CHEESE	1	3,10	4.10
1	JUMBO SAUSAGE CHEESE	{	4.10	3.10
2	GARLIC CHEESE	{	2.00	2.00

--- Totals ---

Total Sales (Excluding GST) = 7.74
Total GST = 0.46
Total Sales (Inclusive of GST) = 8.20

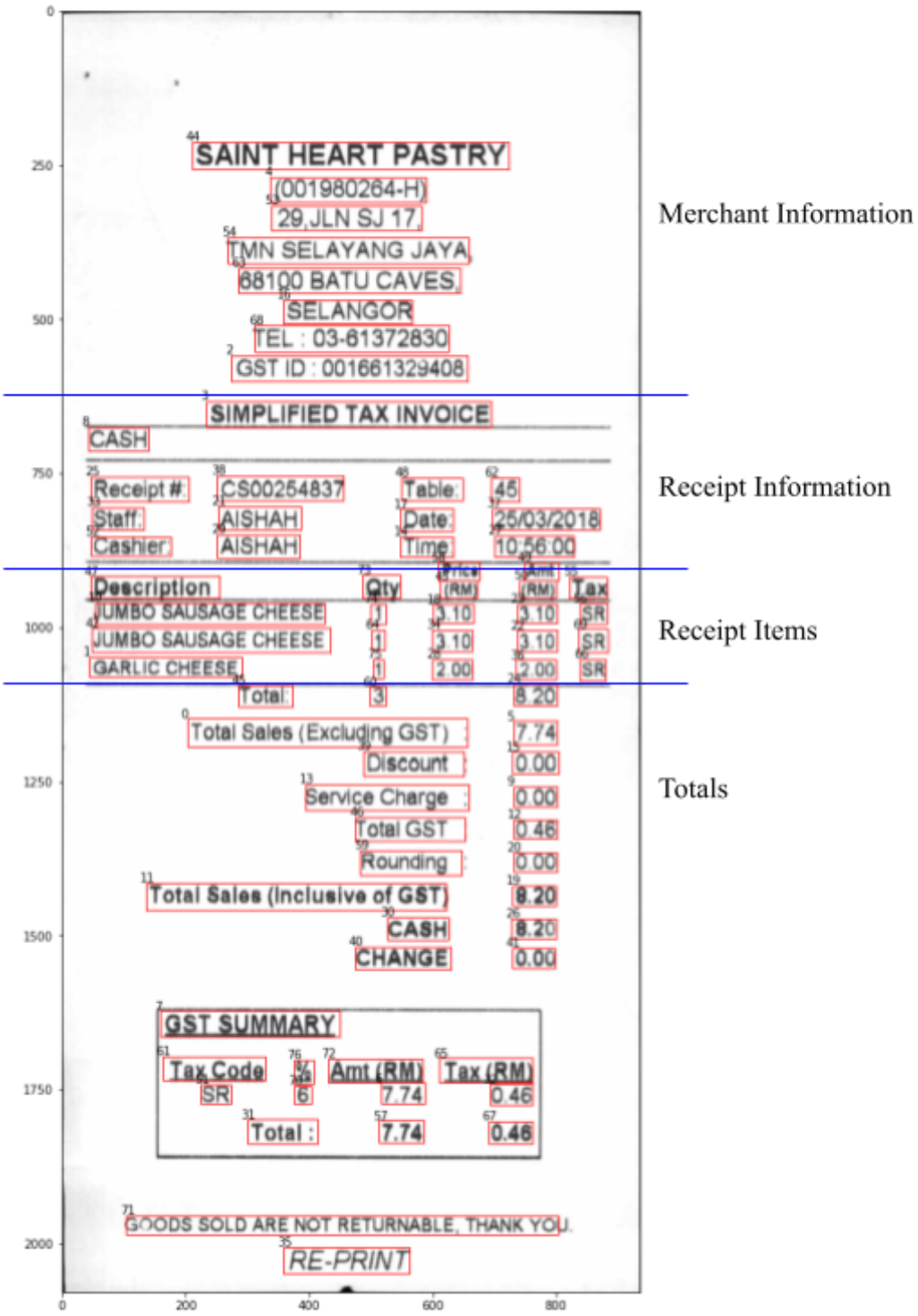


Figure 23: Receipt broken into four categories

CHAPTER 4
EVALUATION AND RESULTS

4.1 Introduction

This experiment was carried out through three tasks described in section 1.5 Scope and SROIE 2019 receipts dataset was used for both training/validation and testing in all stages. Evaluation was first done in those three tasks/stages separately, but then combined into a final metric at the end. Following sections discuss in detail about the evaluation methods.

4.2 Evaluation on Task 1: Scanned Receipt Text Localization and Classification

Text detection/localization and classification was done using a Machine Learning algorithm called You Only Looks Once (YOLO) version 5 ("ultralytics/yolov5," 2022, p. 5). While this algorithm is training/validating the model, it collects the performance metrics such as **Precision, Recall, mAP_0.5, mAP_0.5:0.95, box_loss, obj_loss and cls_loss** with saving into a CSV file in each epoch and finally draws a graph as in Figure 25 and Figure 26. In addition to that, few other graphs like PR curve, F1 curve and confusion matrix etc. are also generated. Those metrics can be used to evaluate the task 1 performance. Exact comparison between ground truth and prediction bounding box coordinates are not accepted in this case. Above mentioned major performance metrics and their sub components are explained below in the context of this task.

True Positive (TP): Correct detection made by the model.
False Positive (FP): Incorrect detection made by the model.
False Negative (FN): A Ground-truth missed (not detected) by the model.
True Negative (TN): This is the background region correctly not detected by the model. Usually not used.

Intersection over Union (IoU): $\frac{\text{Area of intersection between ground truth and detection}}{\text{Area of union between ground truth and detection}}$

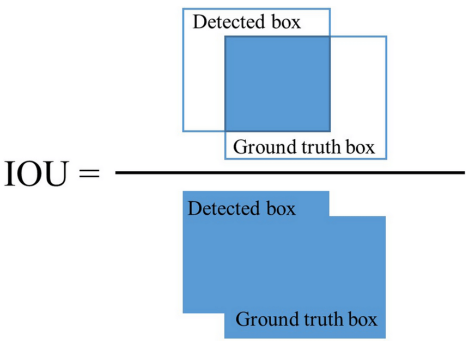


Figure 24: Intersection over Union (IoU) | Image from (Mahdi et al., n.d.)

Precision: TP / TP + FP = Correct detections / All detections
Recall: TP / TP + FN = Correct detections / All ground-truths

Average Precision (AP): Area under Precision-Recall (PR) Curve. Calculated based on IoU threshold.

Mean Average Precision (mAP): AP is calculated for each class and mAP is to get the average out of them.

mAP_0.5: mAP calculated at IoU threshold 0.5
mAP_0.5:0.95: mAP calculated at IoU threshold 0.5 to 0.95 with 0.05 steps with averaging

box_loss: bounding box regression loss (Mean Squared Error)
obj_loss: the confidence of object presence is the objectness loss (Binary Cross Entropy)
cls_loss: the classification loss (Cross Entropy)

F1 Score = 2 * ((Precision * Recall) / (Precision + Recall))

4.2.1 Results of Training a Multi-Class Text Detection Model

Following is the output of the last one out of 20 epochs for 40 images with 151 classes in training. Execution environment spec was DELL-Latitude-5591, Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz 12-core CPU, 2 GB NVIDIA GeForce MX130 GPU, 32 GM RAM (developer laptop). Mainly GPU was utilized. As it shows, it has run with difficulty through several timeouts and retries. And ended up with significantly low accuracy metrics: precision = 0.000155, recall = 0.00152, mAP_0.5 = 9.32e-05, mAP_0.5:0.95 = 1.65e-05. Hence theoretically to gain a mAP_0.5 ~ 0.9 (90 percent) accuracy it requires approximately 200,000 epochs and 700 hours (29 days) in this environment. Main culprit for this very low accuracy is high no of classes used.

```
Epoch   gpu_mem       box       obj       cls    labels  img_size
    19/19     1.25G    0.1292    0.1761    0.1152       484       640: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@WARNING: NMS
time limit 0.540s exceeded
               Class     Images     Labels          P          R     mAP@.5 mAP@WARNING: NMS
time limit 0.540s exceeded
               Class     Images     Labels          P          R     mAP@.5 mAP@WARNING: NMS
time limit 0.540s exceeded
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all         40       2295   0.000155    0.00152   9.32e-05   1.65e-05

20 epochs completed in 0.070 hours.
```

4.2.2 Results of Training a Single-Class Text Detection Model

Following is the output of the last one out of 65 epochs for 704 images with a single class in training. Execution environment spec was DELL-Latitude-5591, Intel(R) Core(TM) i7-8850H CPU @ 2.60GHz 12-core CPU, 2 GB NVIDIA GeForce MX130 GPU, 32 GM RAM (developer laptop). Mainly GPU was utilized. As it shows, within nearly 3 hours without much hassle the model has trained to a very high accuracy level: precision = 0.974, recall = 0.922, mAP_0.5 = 0.967, mAP_0.5:0.95 = 0.658. This was a very successful training and selected as the object detector in the system. Figures 18, 19, 20 and 21 visually represent the performance metrics of the model.

```
Epoch   gpu_mem       box       obj       cls    labels  img_size
  64/64    0.996G   0.06866    0.2025         0       340       640: 100%|
               Class     Images     Labels          P          R     mAP@.5 mAP@
                 all        704      37554      0.974      0.922      0.967      0.658
```

65 epochs completed in 3.104 hours.

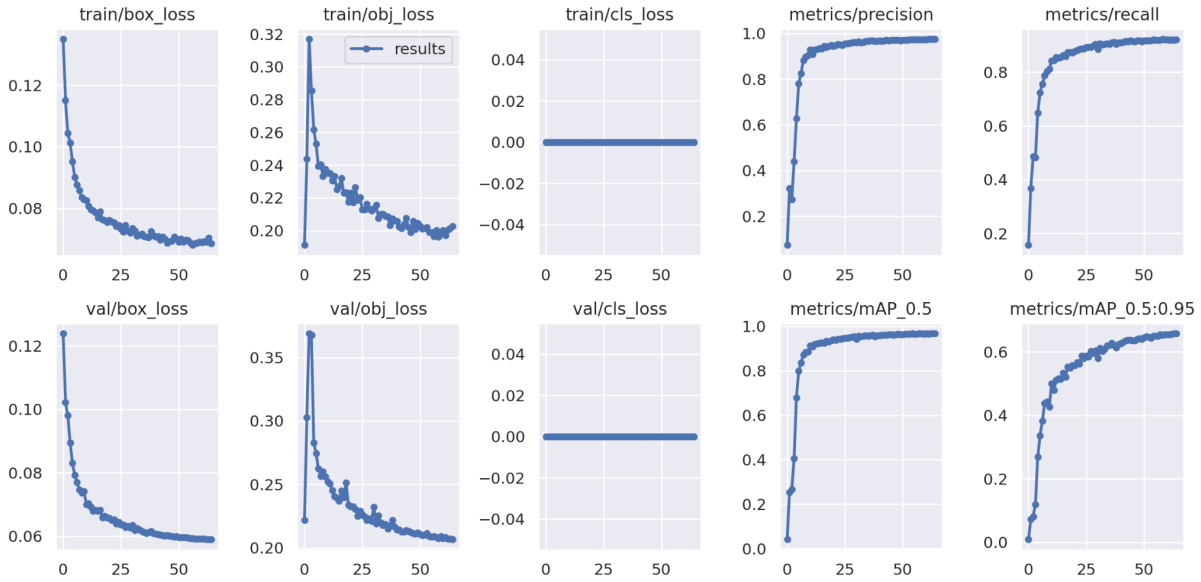


Figure 25: Main performance metrics results

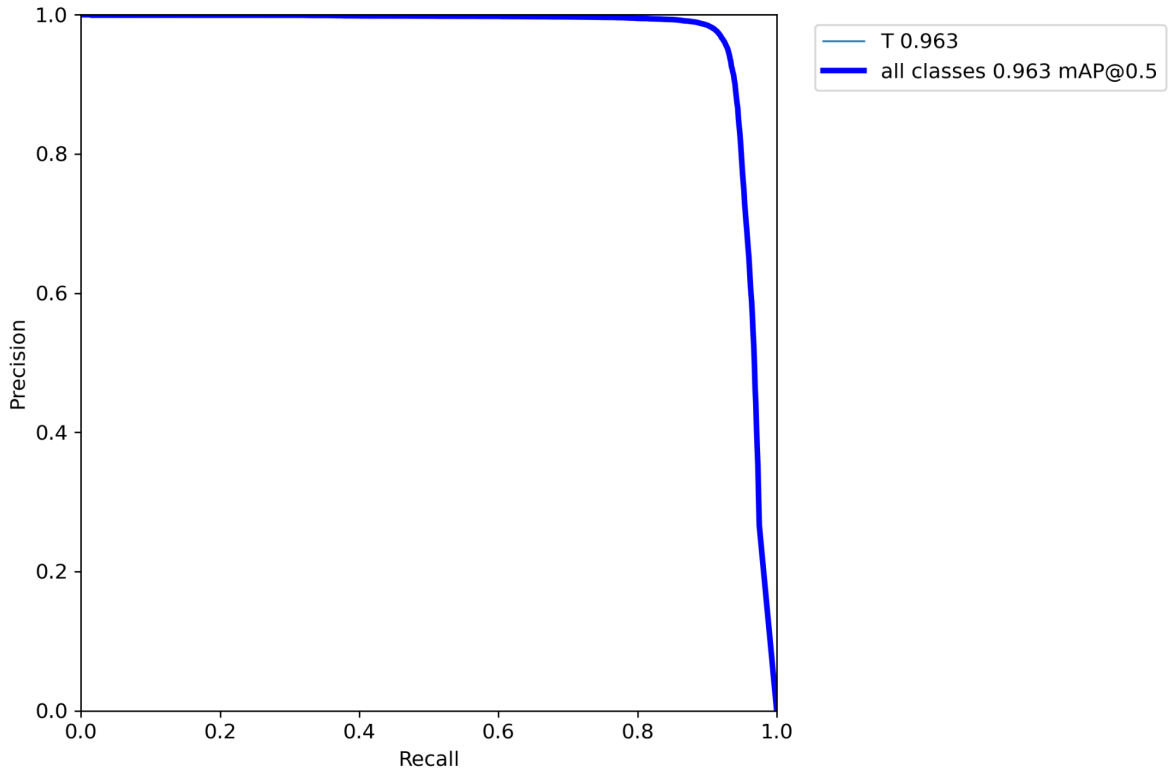


Figure 26: Precision-Recall curve

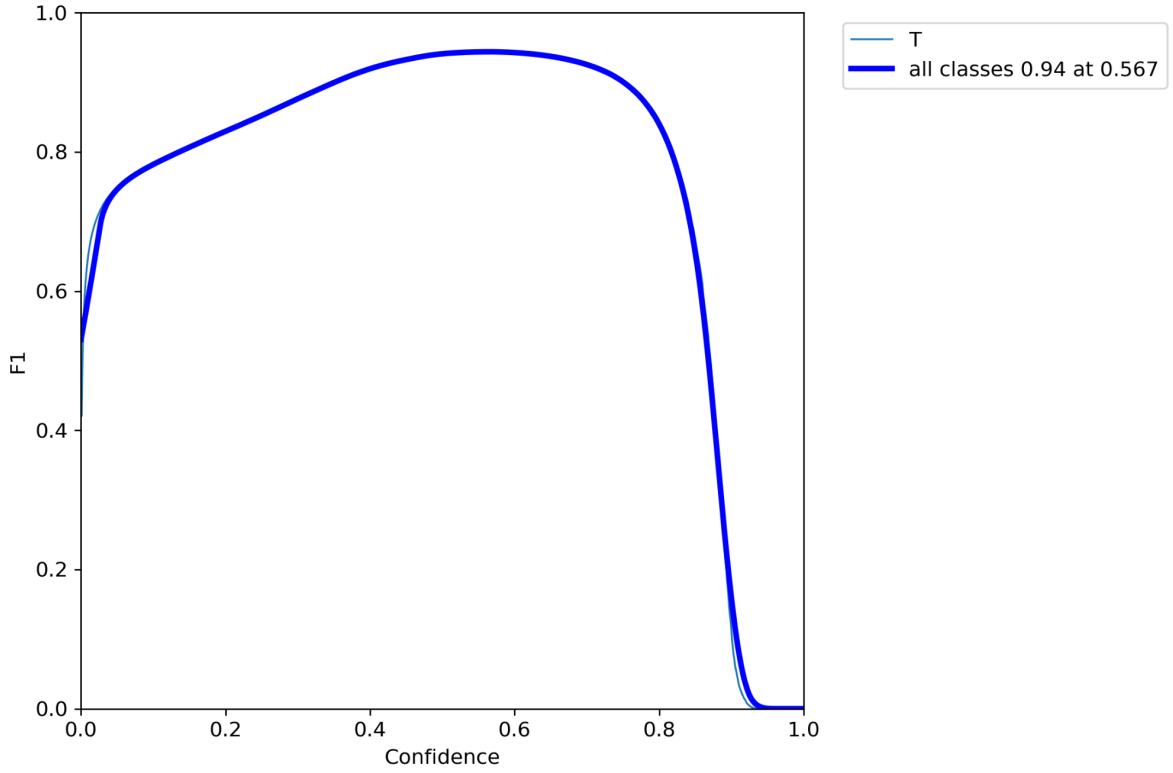


Figure 27: F1 curve

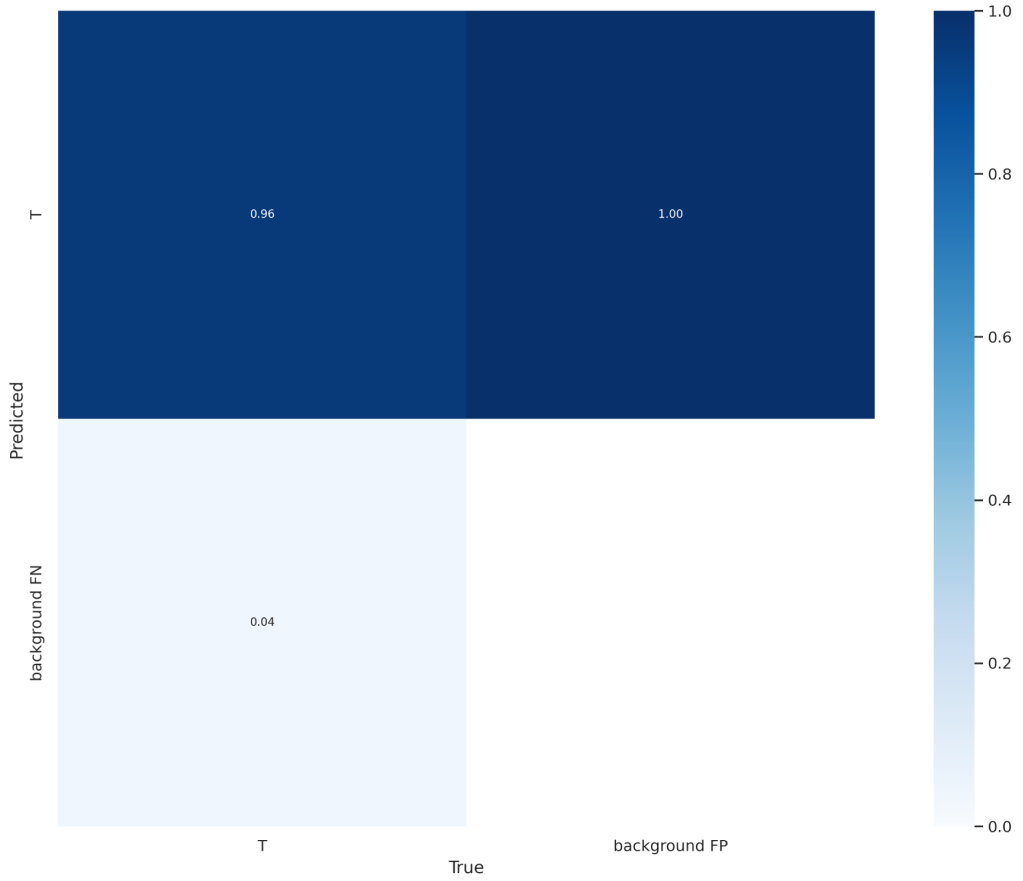


Figure 28: Confusion matrix

4.3 Evaluation on Task 2: Scanned Receipt OCR

In the dataset's original format, each image annotation file includes what is the text included in each bounding box (ground truth). Text extracted from the OCR engine can be compared with that ground truth for each bounding box. Thus can measure correctly inferred bounding boxes count out of the total no of bounding boxes in the image as a percentage. It can be repeated to all the images in the training dataset and obtain the final OCR accuracy. And aligned with that an average time for OCR can be calculated too.

4.4 Evaluation on Task 3: Key Information Extraction

In the original dataset, each image has a separate annotation file to hold mapped key-value pairs that how it should be when the information is finally extracted from the image (ground truth). Extracted information mappings from the inference rules can be compared with that ground truth. Following is an example ground truth content from the original dataset with basic key-value information defined by the ICDAR 2019 conference (Huang et al., 2019). It will be extended to include invoice item rows and other important information as required.

{

"company": "BOOK TA .K (TAMAN DAYA) SDN BHD",
"date": "25/12/2018",
"address": "NO.53 55,57 & 59, JALAN SAGU 18, TAMAN DAYA, 81100 JOHOR BAHRU, JOHOR.",
"total": "9.00"
}

CHAPTER 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

This research work was based on the hypothesis “Information extraction from scanned invoice/receipt images should be able to be achieved using Machine Learning, OCR and spatial feature mapping techniques”. Those three techniques were identified through a thorough literature review. After adapting those to the problem context in the way described in the Methodology chapter, hypothesis could be successfully achieved. As per the research work has proved, the methodology adopted can be applied to any scanned invoice/receipt dataset with proper adjustments. Moreover not limited to invoices/receipts, it can be utilized for extracting information from any document type.

5.2 Future Work

As this was a proof of concept work carried out through a certain large scope and a limited time frame, it is not fully production ready yet. Therefore future works listed below should be conducted to make the project progressive.

- Extend the key information mapping rules to support all the formats included in the dataset. More better if format-agnostic techniques can be adopted.
- Improve the accuracy of the OCR output by enhancing the quality of input images using more image processing techniques.
- Train multi-class text detection model in a robust cloud server, i.e. Google Compute Engine. That way simplifies the key information mapping rules.
- Write evaluation scripts for OCR and key information mapping steps and prepare the dataset for that.

APPENDICES

Appendix A

```
import glob
from PIL import Image
import time

start_time = time.time()
count = 0
print("Started Filtering")

x_train_paths = glob.glob('../../../../Datasets/SROIE2019/0325updated.task1train(626p)/*.jpg')
print(f"Processing {len(x_train_paths)} items")
print("----------------------------------------")

for imgPath in x_train_paths:
    imgFilename = imgPath[(imgPath.rfind('/') + 1):len(imgPath)]
    img = Image.open(imgPath)
    imgWidth = img.width
    imgHeight = img.height
    try:
        txtPath = imgPath.replace(".jpg", ".txt")
        txtFile = open(txtPath, "r")
        txtFilename = imgFilename.replace(".jpg", ".txt")
        labelFile = open(f"../../dataset/dataset_unprocessed/labels/{txtFilename}", "w")
        labelFile.write(txtFile.read())
        labelFile.flush()
        img.save(f"../../dataset/dataset_unprocessed/images/{imgFilename}")
        count += 1

    except FileNotFoundError as err:
        print(err)
```

```
print("---------------------------------------")
print(f"Selected {count} items")
print(f"Finished in {(time.time() - start_time)} seconds")
```

Appendix B

```
from os import path
from tkinter import *
from tkinter import filedialog as fd
from tkinter import ttk
from PIL import Image, ImageTk

classes = []
classesFile = open("../../Classes.txt")
for item in classesFile.readlines():
    classes.append(item.replace("\n", ""))
classesFile.close()

root = Tk()
root.title("Image Labeler")
row1 = Frame(root)
row1.pack()
row2 = Frame(root)
row2.pack(side = BOTTOM)

fileLable = Label(row1, text="File", width=10)
fileLable.pack(side = LEFT)

filePathText = StringVar()
filePath = Entry(row1, width=100, bd=1, textvariable=filePathText)
filePath.pack(side = LEFT)

labelFilePath = ""
bboxes = []
```

```
bbox = []
bboxIndex = 0
labeledBBoxIndexes = set()


def hasFileLabeled(filename):
    labeledListFile = open("../../dataset/dataset_labeled/labeled_list.txt", "r")
    labeledList = labeledListFile.read()
    labeledListFile.close()
    return filename in labeledList

def openFile():
    filetypes = (
        ('Image files', '*.jpg'),
        ('Text files', '*.txt'),
        ('All files', '*.*')
    )

    imgFilePath = fd.askopenfilename(
        title="Open files",
        initialdir="../../dataset/dataset_labeled/images",
        filetypes=filetypes)

    if(len(imgFilePath) != 0):
        canvas.delete("all")
        bboxes.clear()
        labeledBBoxIndexes.clear()
        saveMsgText.set("")
        classDropdown.selection_clear(0, END)
        bboxText.set(f"BBox No: \n\nText: \n\nClass: ")

        filePathText.set(imgFilePath)
        imgFile = Image.open(imgFilePath)
        global img
```

```
        img = ImageTk.PhotoImage(imgFile)
        canvas.create_image(0, 0, anchor=NW, image=img)

        global labelFilePath
        labelFilePath = imgFilePath.replace("images", "labels").replace(".jpg", ".txt")
        labelFile = open(labelFilePath, "r")

        index = 0
        for row in labelFile.readlines():
            bbox = row.split(",")
            bboxes.append(bbox)
            tag = f"bbox{index}"
            canvas.create_rectangle(bbox[0], bbox[1], bbox[4], bbox[5], fill="red",
stipple="gray25", tags=tag)
            canvas.create_text(int(bbox[0])-10, int(bbox[1])+5, text=index, fill="black",
font=('Helvetica 10 bold'))
            canvas.tag_bind(tag,"<Button-1>", bboxClicked)
            index += 1
        labelFile.close()
        imgFile.close()

        filename = path.basename(labelFilePath).replace(".txt", "")
        if hasFileLabeled(filename):
            for x in range(len(bboxes)):
                labeledBBoxIndexes.add(x)

        saveBtn.configure(state=NORMAL)
    else:
        labelFilePath = ""



fileBtn = Button(row1, text="Open File", command=openFile)
fileBtn.pack(side = LEFT)
```

```
def bboxClicked(*args):
    classDropdown.selection_clear(0, END)
    global bboxIndex
    bboxIndex = int(canvas.gettags("current")[0].replace("bbox", ""))
    global bbox
    bbox = bboxes[bboxIndex]
    text = ""
    label = bbox[8]
    if(label.startswith("<<<") & label.endswith(">>>")):
        text = " ".join(bbox[9:len(bbox)])
        label = label.replace("<<<", "").replace(">>>", "")
        classDropdown.selection_set(classes.index(label))
    else:
        label = ""
        text = " ".join(bbox[8:len(bbox)])
    bboxText.set(f"BBox No: {bboxIndex}\n\nText: {text}\n\nClass: {label}")



def onMouseWheel(event):
    if(event.state == 16):
        if(event.num == 4):
            canvas.yview_scroll(-1, "units")
        elif(event.num == 5):
            canvas.yview_scroll(1, "units")
    elif(event.state == 17):
        if(event.num == 4):
            canvas.xview_scroll(-1, "units")
        elif(event.num == 5):
            canvas.xview_scroll(1, "units")



canvas = Canvas(row2, width=800, height=900)
canvas.grid(row=0, column=0)
```

```
canvas_scroll_x = Scrollbar(row2, orient="horizontal", command=canvas.xview)
canvas_scroll_x.grid(row=1, column=0, sticky="ew")
canvas_scroll_y = Scrollbar(row2, orient="vertical", command=canvas.yview)
canvas_scroll_y.grid(row=0, column=1, sticky="ns")
canvas.configure(yscrollcommand=canvas_scroll_y.set,
xscrollcommand=canvas_scroll_x.set)
canvas.bind("<MouseWheel>", onMouseWheel)  # Windows
canvas.bind("<Button-4>", onMouseWheel)    # Linux
canvas.bind("<Button-5>", onMouseWheel)    # Linux

col2 = Frame(row2)
col2.grid(row=0, column=2, sticky="ns")

bboxText = StringVar()
bboxText.set("BBox no: \n\nText: \n\nClass: ")
bboxLable = Label(col2, textvariable=bboxText, width=50)
bboxLable.grid(row=0, column=0, sticky="ns")

def onClassSelect(evt):
    if(len(bbox) > 8):
        item = bbox[8]
        if(item.startswith("<<<") & item.endswith(">>>")):
            bbox.remove(item)
        w = evt.widget
        index = int(w.curselection()[0])
        value = w.get(index)
        bbox.insert(8, f"<<<{value}>>>")
        labeledBBoxIndexes.add(bboxIndex)



classDropdownFrame = Frame(col2)
classDropdownFrame.grid(row=1, column=0, sticky="ns")
classDropdown_scroll_y = Scrollbar(classDropdownFrame)
```

```
classDropdown_scroll_y.grid(row=0, column=1, sticky="ns")
classText = StringVar()
classDropdown = Listbox(classDropdownFrame, yscrollcommand =
classDropdown_scroll_y.set, listvariable = classText, width=45, height=70)
classDropdown.bind('<<ListboxSelect>>', onClassSelect)
for item in classes:
  classDropdown.insert(END, item)
classDropdown.grid(row=0, column=0, sticky="ns")


def saveImage():
   label = classDropdown.get(ANCHOR)

   if(len(labeledBBoxIndexes) == len(bboxes)):
      labelFile = open(labelFilePath, "w")
      content = ""
      for bbox in bboxes:
         content += ",".join(bbox)
      labelFile.write(content)
      labelFile.close()

      labeledListFile = open("../../dataset/dataset_labeled/labeled_list.txt", "a")
      filename = path.basename(labelFilePath).replace(".txt", "")
      if not hasFileLabeled(filename):
         labeledListFile.write(f"{filename}\n")
      labeledListFile.close()
      saveMsgText.set(f"Successfully saved {filename}")
   else:
      notLabeledBBoxCount = len(bboxes) - len(labeledBBoxIndexes)
      saveMsgText.set(f"{notLabeledBBoxCount}/{len(bboxes)} bboxes are not labeled.")


saveMsgText = StringVar()
saveMsgLable = Label(col2, textvariable=saveMsgText, width=50)
```

```
saveMsgLable.grid(row=2, column=0, sticky="ns")

saveBtn = Button(col2, text="Save Labels", command=saveImage, state=DISABLED)
saveBtn.grid(row=3, column=0, sticky="ns")

mainloop()
```

Appendix C

```
import glob
from PIL import Image
import tensorflow as tf
import time

classes = []
classesFile = open("../../Used_Classes.txt")
for item in classesFile.readlines():
    classes.append(item.replace("\n", ""))
classesFile.close()

labeledList = []
labeledListFile = open("../../dataset/dataset_labeled/labeled_list.txt")
for item in labeledListFile.readlines():
    labeledList.append(item.replace("\n", ""))
labeledListFile.close()

start_time = time.time()
count = 0
print("Started processing")

print(f"Processing {len(labeledList)} items")

for item in labeledList:
    imgFilename = item + ".jpg"
    print(imgFilename)
```

```
    imgFile = Image.open(f"../../dataset/dataset_labeled/images/{imgFilename}")
    imgWidth = imgFile.width
    imgHeight = imgFile.height
    try:
        txtFilename = item + ".txt"
        txtPath = f"../../dataset/dataset_labeled/labels/{txtFilename}"
        txtFile = open(txtPath, "r")
        labelFile = open(f"../../dataset/dataset_multi_class-40/labels/{txtFilename}", "w")

        for row in txtFile.readlines():
            tokens = row.split(",")

            labelIndex = classes.index(tokens[8].replace("<<<", "").replace(">>>", ""))
            bboxWidth = (int(tokens[2]) - int(tokens[0]))
            bboxHeight = (int(tokens[7]) - int(tokens[1]))
            bboxCenterX = (int(tokens[0]) + bboxWidth/2) / imgWidth
            bboxCenterY = (int(tokens[1]) + bboxHeight/2) / imgHeight
            bboxWidth = bboxWidth / imgWidth
            bboxHeight = bboxHeight / imgHeight

            newRow = f"{labelIndex} {bboxCenterX} {bboxCenterY} {bboxWidth}
{bboxHeight}\n"
            labelFile.write(newRow)

        txtFile.close()
        labelFile.close()
        imgFile.save(f"../../dataset/dataset_multi_class-40/images/{imgFilename}")
        imgFile.close()
        count += 1

    except FileNotFoundError as err:
        print(err)

print(f"Created {count} items")
```

```
print(f"Finished in {(time.time() - start_time)} seconds")
```

Appendix D

```
import glob
from PIL import Image
import time

start_time = time.time()
x_train_paths = glob.glob('../../dataset/dataset_filtered/images/*.jpg')
print(f"Processing {len(x_train_paths)} records")
count = 0

for imgPath in x_train_paths:
#     print(imgPath)
    imgFilename = imgPath[(imgPath.rfind('/') + 1):len(imgPath)]
    img = Image.open(imgPath)
    imgWidth = img.width
    imgHeight = img.height
    img.save(f"../../dataset/dataset_single_class/images/{imgFilename}")
    img.close()
    try:
        txtPath = imgPath.replace("images", "labels").replace(".jpg", ".txt")
        txtFile = open(txtPath, "r")
        txtFilename = imgFilename.replace(".jpg", ".txt")
        labelFile = open(f"../../dataset/dataset_single_class/labels/{txtFilename}", "w")

        for row in txtFile.readlines():
            tokens = row.split(",")

            labelIndex = 0
            bboxWidth = (int(tokens[2]) - int(tokens[0]))
            bboxHeight = (int(tokens[7]) - int(tokens[1]))
            bboxCenterX = (int(tokens[0]) + bboxWidth/2) / imgWidth
            bboxCenterY = (int(tokens[1]) + bboxHeight/2) / imgHeight
```

```
            bboxWidth = bboxWidth / imgWidth
            bboxHeight = bboxHeight / imgHeight

            newRow = f"{labelIndex} {bboxCenterX} {bboxCenterY} {bboxWidth}
{bboxHeight}\n"
            labelFile.write(newRow)

        labelFile.close()

    except FileNotFoundError as err:
        print(err)

    count += 1

print(f"Finished in {(time.time() - start_time)} seconds")
```

Appendix E

Heading, Title, Merchant_Name, Merchant_Name_Label, Merchant_Name_Value, Merchant_RegNo, Merchant_RegNo_Label, Merchant_RegNo_Value, Merchant_TaxNo, Merchant_TaxNo_Label, Merchant_TaxNo_Value, Merchant_Address, Merchant_Address_Label, Merchant_Address_Value, Merchant_Tel, Merchant_Tel_Label, Merchant_Tel_Value, Merchant_Fax, Merchant_Fax_Label, Merchant_Fax_Value, Merchant_Email, Merchant_Email_Label, Merchant_Email_Value, Merchant_Tel_Fax, Merchant_Tel_Email, Merchant_Fax_Email, Merchant_Tel_Fax_Email, Customer_Name, Customer_Name_Label, Customer_Name_Value, Customer_RegNo, Customer_RegNo_Label, Customer_RegNo_Value, Customer_TaxNo, Customer_TaxNo_Label, Customer_TaxNo_Value, Customer_Address, Customer_Address_Label, Customer_Address_Value, Customer_Tel, Customer_Tel_Label, Customer_Tel_Value, Customer_Fax, Customer_Fax_Label, Customer_Fax_Value, Customer_Email, Customer_Email_Label, Customer_Email_Value, Customer_Tel_Fax, Customer_Tel_Email, Customer_Fax_Email, Customer_Tel_Fax_Email, Payment_Method, Payment_Method_Label, Payment_Method_Value, ReceiptNo, ReceiptNo_Label, ReceiptNo_Value, Date, Date_Label, Date_Value, Time, Time_Label, Time_Value, Datetime,

Datetime_Label, Datetime_Value, Datetime_Start, Datetime_End, ReceiptNo_Date, Cashier, Cashier_Label, Cashier_Value, Salesperson, Salesperson_Label, Salesperson_Value, Reference, Reference_Label, Reference_Value, Member, Member_Label, Member_Value, Location, Location_Label, Location_Value, Room, Room_Label, Room_Value, Table_Head, Table_Head_Code, Table_Head_Description, Table_Head_Code_Description, Table_Head_Price, Table_Head_Price_Currency, Table_Head_Taxed_Price, Table_Head_Taxed_Price_Currency, Table_Head_Discount_Rate, Table_Head_Discount, Table_Head_Discount_Currency, Table_Head_Amount, Table_Head_Amount_Currency, Table_Head_Taxed_Amount, Table_Head_Taxed_Amount_Currency, Table_Head_Quantity, Table_Head_Unit, Table_Head_Tax_Code, Table_Head_Tax_Amount, Table_Head_Tax_Amount_Currency, Table_Head_Quantity_Price_Amount, Table_Head_Quantity_Price_Amount_Currency, Table_Head_Quantity_Description, Table_Value_Code, Table_Value_Description, Table_Value_Code_Description, Table_Value_Description_Amount, Table_Value_Price, Table_Value_Price_Currency, Table_Value_Taxed_Price, Table_Value_Taxed_Price_Currency, Table_Value_Discount_Rate, Table_Value_Discount, Table_Value_Discount_Currency, Table_Value_Amount, Table_Value_Amount_Currency, Table_Value_Taxed_Amount, Table_Value_Taxed_Amount_Currency, Table_Value_Quantity, Table_Value_Unit, Table_Value_Tax_Code, Table_Value_Tax_Amount, Table_Value_Tax_Amount_Currency, Table_Value_Quantity_Price, Table_Value_Quantity_Price_Amount, Table_Value_Quantity_Price_Amount_Currency, Table_Value_Quantity_Description, Table_Value_Price_Tax_Amount, Total_Items, Total_Items_Label, Total_Items_Value, Total_Quantity, Total_Quantity_Label, Total_Quantity_Value, Total_Amount, Total_Amount_Label, Total_Amount_Label_Currency, Total_Amount_Value, Total_Amount_Value_Currency, Total_Amount_Excluding_Tax, Total_Amount_Excluding_Tax_Label, Total_Amount_Excluding_Tax_Label_Currency, Total_Amount_Excluding_Tax_Value, Total_Amount_Excluding_Tax_Value_Currency, Total_Tax, Total_Tax_Label, Total_Tax_Label_Currency, Total_Tax_Value, Total_Tax_Value_Currency, Total_Amount_Including_Tax, Total_Amount_Including_Tax_Label, Total_Amount_Including_Tax_Label_Currency, Total_Amount_Including_Tax_Value, Total_Amount_Including_Tax_Value_Currency, Total_Discount, Total_Discount_Label, Total_Discount_Label_Currency, Total_Discount_Rate, Total_Discount_Value, Total_Discount_Value_Currency, Total_Discount_Rate_Value, Total_Discount_Rate_Value_Currency, Total_Amount_Discounted, Total_Amount_Discounted_Label,

Total_Amount_Discounted_Label_Currency, Total_Amount_Discounted_Value, Total_Amount_Discounted_Value_Currency, Rounding_Adjustment, Rounding_Adjustment_Label, Rounding_Adjustment_Label_Currency, Rounding_Adjustment_Value, Rounding_Adjustment_Value_Currency, Total_Amount_Rounded, Total_Amount_Rounded_Label, Total_Amount_Rounded_Label_Currency, Total_Amount_Rounded_Value, Total_Amount_Rounded_Value_Currency, Service_Charges, Service_Charges_Label, Service_Charges_Value, Total_Amount_Final, Total_Amount_Final_Label, Total_Amount_Final_Label_Currency, Total_Amount_Final_Value, Total_Amount_Final_Value_Currency, Cash, Cash_Label, Cash_Label_Currency, Cash_Value, Cash_Value_Currency, Change, Change_Label, Change_Label_Currency, Change_Value, Change_Value_Currency , Tax_Summary_Title, Tax_Code_Label, Tax_Code_Value, Tax_Rate_Label, Tax_Rate_Value, Tax_Code_Rate_Value, Total_Label, Amount_Label, Amount_Label_Currency, Amount_Value, Amount_Value_Currency, Tax_Label, Tax_Label_Currency, Tax_Value, Tax_Value_Currency, Items_Label, Items_Value, Currency, Barcode, Multiplication, Multiplication_Currency, Colon, Equal, Extra

Appendix F

```
# *** PROGRAMATIC INFERENCE ***

import torch
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image, ImageFilter
import os
import numpy as np
import pandas as pd
%matplotlib inline
from pytesseract import pytesseract

from xml.dom import minidom
```

```
# Model
model = torch.hub.load('ultralytics/yolov5', 'custom', 'yolov5/runs/train/exp8/weights/best.pt')
# or yolov5n - yolov5x6, custom

# Images
# X00016469670, X00016469671, X51005676534, X51005715006, *X51005719905,
X51006008081, X51006008083
img_filename = "X51005719905"
img_path = f'../../dataset/dataset_test/images/{img_filename}.jpg'  # or file, Path, PIL,
OpenCV, numpy, list

# Inference
results = model(img_path)

# Results
results.print()  # or .show(), .save(), .crop(), .pandas(), etc.
# results.show()
df = results.pandas().xyxy[0]
# Sort by bbox first point y value
df = df.sort_values(by=['ymin'])
# Add extra columns to existing dataframe
df["text"] = ""
df["baw_ratio"] = np.nan
df["label"] = ""
# print(df)

img = Image.open(img_path)
# img = img.convert('L')
# print(img.dpi)

CONFIDENCE_THRESHOLD = 0.5

# Display bounding boxes
fig, ax = plt.subplots(figsize=(40,20))
```

```
for x, row in df.iterrows():
    confidence = row[4]
    if confidence > CONFIDENCE_THRESHOLD:
        rect = patches.Rectangle((row[0], row[1]), (row[2] - row[0]), (row[3] - row[1]),
linewidth=1, edgecolor='r', facecolor='none')
        ax.add_patch(rect)
        ax.text(row[0]-10, row[1]-2, x)

ax.imshow(img)
```

Appendix G

```
# --- OCR and store texts ---

print("\n*** OCR and store texts ***\n")

# Point tessaract_cmd to tessaract executable
path_to_tesseract = r'/usr/bin/tesseract'
pytesseract.tesseract_cmd = path_to_tesseract

# # Setup Google Vision API
# from google.cloud import vision
# import base64
# client = vision.ImageAnnotatorClient()

# Create ImageParts directories
if not os.path.exists(f"ImageParts/ImageParts-{img_filename}"):
    os.makedirs(f"ImageParts/ImageParts-{img_filename}")

# OCR full image
# fulltext = pytesseract.image_to_string(img)
# print("\n" + fulltext)
```

```
# --- OCR and store texts in dataframe ---
for x, row in df.iterrows():
    confidence = row[4]
    if confidence > CONFIDENCE_THRESHOLD:
        # Extract image part
        cropped_img = img.crop((row[0], row[1], row[2], row[3]))
        cropped_img.save(f"ImageParts/ImageParts-{img_filename}/ImagePart{x}.jpg",
dpi=(300.0, 300.0))
        # Convert image part to black and white, and get ratio between pixel counts
        bawh = cropped_img.convert("1").histogram()
        df.at[x, "baw_ratio"] = bawh[0] / bawh[255]

        # Median cropped image to remove salt and pepper noise
        medianed_img = cropped_img.filter(ImageFilter.MedianFilter);

medianed_img.save(f"ImageParts/ImageParts-{img_filename}/ImagePart-medianed{x}.jpg",
dpi=(300.0, 300.0))
        sharpened_img = medianed_img.filter(ImageFilter.SHARPEN);
        # Sharpen original (cropped) image - PIL
#         sharpened_img = cropped_img.filter(ImageFilter.SHARPEN);
#         sharpened_img = sharpened_img.filter(ImageFilter.SHARPEN);

sharpened_img.save(f"ImageParts/ImageParts-{img_filename}/ImagePart-sharpened{x}.jpg",
dpi=(300.0, 300.0))

        # Enlarge sharpened image
        scale = 17
        enlarged_img = sharpened_img.resize((sharpened_img.width * scale,
sharpened_img.height * scale))

enlarged_img.save(f"ImageParts/ImageParts-{img_filename}/ImagePart-enlarged{x}.jpg",
dpi=(300.0, 300.0))
#         print(len(enlarged_img.histogram()))
```

```
        # Extract text from image (OCR)
        text = pytesseract.image_to_string(enlarged_img)

        if text.strip() == "":
            # OCR with option '--psm 10' when (assuming) there is one character in image
            text = pytesseract.image_to_string(sharpened_img, config='--psm 10') # Used
sharpened_img here, not enlarged_img

            if text.strip() == "":
                # Median sharpened image to remove salt and pepper noise
                medianed_img = sharpened_img.filter(ImageFilter.MedianFilter);
                medianed_img = medianed_img.filter(ImageFilter.MedianFilter);

medianed_img.save(f"ImageParts/ImageParts-{img_filename}/ImagePart-medianed{x}.jpg",
dpi=(300.0, 300.0))
                # Enlarge (sharpened + medianed) image
                enlarged_img = medianed_img.resize((medianed_img.width * scale,
medianed_img.height * scale))

enlarged_img.save(f"ImageParts/ImageParts-{img_filename}/ImagePart-enlarged{x}.jpg",
dpi=(300.0, 300.0))
                # OCR
                text = pytesseract.image_to_string(enlarged_img)

#         # Image enhancement for Google Vision API
#         medianed_img = sharpened_img.filter(ImageFilter.MedianFilter);
#         medianed_img = medianed_img.filter(ImageFilter.MedianFilter);
#         enlarged_img = medianed_img.resize((medianed_img.width * scale,
medianed_img.height * scale))

#         # Using Google Vision API
#         content = base64.b64encode(cropped_img.tobytes())
#         image = vision.Image(content=content)
#         response = client.text_detection(image=image)
```

```
#         print(x, " ---> ", response)

        df.at[x, "text"] = text.strip()
        print(x, " ---> ", text.strip())
#         print(x, " ---> ", text.strip(), " / ", len(text.strip()), " / ", (bawh[0]/bawh[255]))
#     if x == 1:
#         break

# print(df)
```

Appendix H

```
# --- Key-Value identification ---

import re

print("\n*** Key-Value identification ***\n")

headings = ["tan chay yee", "tan woon yann", "190"]
titles = ["tax invoice", "taa invoice", "t&x invoice", "simplified tax invoice", "simplified taa
invoice", "cash bill", "invoice no"]
descriptions = ["description", "descriptian", "item", "jtam"]
totals = ["total exclude gst", "total include gst"]
quantities = ["qty", "oty", "aty"]
unit_prices = ["price", "s/price"]
amounts = ["amount", "amt"]

title_ymin = df[df["text"].str.lower().isin(titles)]["ymin"].values[0]
total_ymin = df[df["text"].str.lower().str.contains("total|fotal", na=False)]["ymin"].values[0]
description = df[df["text"].str.lower().isin(descriptions)]
description_ymin = description["ymin"].values[0]
description_ymax = description["ymax"].values[0]
table_headers = df[(df.ymin >= description_ymin-30) & (df.ymin <= description_ymin+10)]
qty_xmin = table_headers[table_headers.text.str.lower().isin(quantities)]["xmin"].values[0]
```

```
price_xmin = table_headers[table_headers.text.str.lower() == "price"]["xmin"].values[0]
amount_xmin = table_headers[table_headers["text"].str.lower().str.contains("amount|amt",
na=False)]["xmin"].values[0]

merchant_name = ""
merchant_address = ""
merchant_reg_no = ""
merchant_tax_no = ""
merchant_email = ""
merchant_tel = ""
invoice_no = ""
datetime = ""
date = ""
time = ""

item_rows = pd.DataFrame(columns=("description", "qty", "price", "amount"))

total_excluding_gst = ""
total_gst = ""
total_inclusive_gst = ""

invoice_no_value_index = -1
datetime_value_index = -1

for x, row in df.iterrows():
    confidence = row[4]
    if (confidence > CONFIDENCE_THRESHOLD):
        xmin = row[0]
        ymin = row[1]
        xmax = row[2]
        text = row[7]
        baw_ratio = row[8]
        label = row[9]
```

```
        if text.lower() in titles:
            label = "Title"
        else:
            if ymin < title_ymin:
                if "\n" in text:
                    label = "Malformed"
                elif text in headings:
                    label = "Heading"
                elif " COPY " in text:
                    label = "Extra"
                else:
                    if baw_ratio >= 0.3:
                        if text.startswith("ROC NO:") | text.startswith("Co No:"):
                            label = "Merchant_Reg_No"
                            merchant_reg_no = text.replace("ROC NO:", "").replace("Co No:",
"").strip()
                        else:
                            label = "Merchant_Name"
                            merchant_name = text
                    else:
                        reg_nos = re.search("\d+[-]{1}[A-Z]{1}", text)
                        if reg_nos:
                            label = "Merchant_Reg_No"
                            merchant_reg_no = reg_nos[0].strip()
                        elif text.startswith("GST ID") | text.startswith("GST NO") |
text.startswith("GST ID No") | text.startswith("GST Reg No"):
                            label = "Merchant_Tax_No"
                            merchant_tax_no = ''.join([i for i in text if i.isdigit()])
                        elif text.lower().startswith("email"):
                            label = "Merchant_Email"
                            merchant_email = text.replace("Email:", "").strip()
                        elif text.lower().startswith("tel"):
                            label = "Merchant_Tel"
                            merchant_tel = ''.join([i for i in text if i.isdigit()])
```

```
                else:
                    label = "Merchant_Address"
                    if(len(merchant_address) == 0):
                        merchant_address = text
                    else:
                        merchant_address = merchant_address + ", " + text
            elif (ymin > title_ymin) & (ymin < description_ymin-5):
                if ("Invoice No" in text) | ("Invoice Na" in text) | ("Invaice No" in text) | ("Doc No"
in text) | ("Receipt #" in text):
                    result = df[(df.ymin >= ymin-5) & (df.ymin <= ymin+5) & (df.xmin >
xmax)]["text"]
                    invoice_no = result.values[0]
                    label = "Invoice_No_Label"
                    invoice_no_value_index = result.index[0]
                elif ("Datetime" in text) | ("Date" in text) | ("Time" in text):
                    # Regex - date | time | datetime
                    datetime_match = re.search("\d{2}\/\d{2}\/\d{4} |
\d{2}\:\d{2}\:\d{2}(\s*([AaPp][Mm]))? |
(\d{2}\/\d{2}\/\d{4}\s*\d{2}\:\d{2}\:\d{2}(\s*([AaPp][Mm])))", text)
                    datetime_val = ""
                    if datetime_match is not None:
                        datetime_val = datetime_match.group()
                    else:
                        result = df[(df.ymin >= ymin-5) & (df.ymin <= ymin+5) & (df.xmin >
xmax)]["text"]
                        if len(result) > 0:
                            datetime_val = result.values[0]
                        datetime_value_index = result.index[0]
                    if ("Datetime" in text):
                        datetime = datetime_val
                        label = "Datetime_Label"
                    elif ("Date" in text):
                        date = datetime_val
                        label = "Date_Label"
```

```
                elif ("Time" in text):
                    time = datetime_val
                    label = "Time_Label"
            elif (ymin >= description_ymin-5) & (ymin < total_ymin):
                if text.lower() in descriptions:
                    label = "Table_Description_Label"
                    desc_values = df[(df.xmin >= xmin-10) & (df.xmin <= xmin+10) & (df.ymin > description_ymax) & (df.ymin < total_ymin)]
                    for desc_value in desc_values.iterrows():
                        desc_ymin = desc_value[1].ymin
                        desc_text = desc_value[1].text
                        qty = df[(df.xmin >= qty_xmin-10) & (df.xmin <= qty_xmin+30) & (df.ymin >= desc_ymin-10) & (df.ymin < desc_ymin+10)]["text"].values[0]
                        price = df[(df.xmin >= price_xmin-10) & (df.xmin <= price_xmin+20) & (df.ymin >= desc_ymin-10) & (df.ymin < desc_ymin+10)]["text"].values[0]
                        amount = df[(df.xmin >= amount_xmin-20) & (df.xmin <= amount_xmin+70) & (df.ymin >= desc_ymin-10) & (df.ymin < desc_ymin+10)]["text"].values[0]
#                         print("==========> ", desc_text, qty, price, amount)
                        item_row = pd.Series({"description": desc_text, "qty": qty, "price": price, "amount": amount})
                        item_rows = pd.concat([item_rows, pd.DataFrame([item_row], columns=item_row.index)]).reset_index(drop=True)
            elif ymin >= total_ymin:
#                 print("----------> ", text)
                if "Total Sales (Excluding GST)" in text:
                    total_excluding_gst = df[(df.xmin > xmax) & (df.ymin >= ymin-10) & (df.ymin < ymin+10)]["text"].values[0]
                elif "Total GST" in text:
                    total_gst = df[(df.xmin > xmax) & (df.ymin >= ymin-10) & (df.ymin < ymin+10)]["text"].values[0]
                elif "Total Sales (Inclusive of GST)" in text:
                    total_inclusive_gst = df[(df.xmin > xmax) & (df.ymin >= ymin-10) & (df.ymin < ymin+10)]["text"].values[0]
```

```
        df.at[x, "label"] = label
    #            print(x, " ===> ", text, " / ", baw_ratio, " / [", label, "]")

# Special updates which cannot be achieved inside the loop
df.at[invoice_no_value_index, "label"] = "Invoice_No_Value"
df.at[datetime_value_index, "label"] = "Datetime_Value"

# Corrections phase
# Correct Merchant_Name and Merchant_Address
if(len(df[df["label"] == "Merchant_Name"]) == 0):
    address_df = df[df["label"] == "Merchant_Address"]
    if(len(address_df) > 1):
        df.at[address_df.index[0], "label"] = "Merchant_Name"
        merchant_name = address_df.iloc[0]["text"]
        merchant_address = merchant_address.replace(merchant_name + ", ", "")

# print(df[df.confidence > CONFIDENCE_THRESHOLD][["text", "label"]])

print("\n*** Final result ***\n")

print("\n--- Merchant Information ---\n")
print("Name = ", merchant_name)
print("Address = ", merchant_address)
print("Reg No = ", merchant_reg_no)
print("Tax No = ", merchant_tax_no)
print("Email = ", merchant_email)
print("Tel = ", merchant_tel)

print("\n--- Receipt Information ---\n")
print("Receipt No = ", invoice_no)
print("Datetime = ", datetime if datetime else date + " " + time)

print("\n--- Receipt Items ---\n")
```

```
print(item_rows)

print("\n--- Totals ---\n")
print("Total Sales (Excluding GST) = ", total_excluding_gst)
print("Total GST = ", total_gst)
print("Total Sales (Inclusive of GST) = ", total_inclusive_gst)
```

Appendix I

```
215,215,720,215,720,255,215,255,SAINT HEART PASTRY
342,269,590,269,590,314,342,314,(001980264-H)
347,319,581,319,581,358,347,358,29,JLN SJ 17 ,
263,372,666,372,666,410,263,410,TMN SELAYANG JAYA,
285,421,643,421,643,461,285,461,68100 BATU CAVES,
363,469,568,469,568,504,363,504,SELANGOR
305,515,625,515,625,549,305,549,TEL : 03-61372830
279,562,649,562,649,595,279,595,GST ID : 001661329408
238,638,694,638,694,671,238,671,SIMPLIFIED TAX INVOICE
44,679,140,679,140,711,44,711,CASH
50,760,199,760,199,800,50,800,RECEIPT #:
50,806,133,806,133,843,50,843,STAFF:
48,849,178,849,178,885,48,885,CASHIER:
255,761,457,761,457,791,255,791,CS00254837
253,810,384,810,384,839,253,839,AISHAH
252,854,382,854,382,886,252,886,AISHAH
550,760,642,760,642,794,550,794,TABLE:
552,808,632,808,632,842,552,842,DATE:
550,852,631,852,631,884,550,884,TIME:
697,757,739,757,739,791,697,791,45
694,806,872,806,872,844,694,844,25/03/2018
702,851,835,851,835,885,702,885,10:56:00
51,919,238,919,238,961,51,961,DESCRIPTION
48,961,428,961,428,990,48,990,JUMBO SAUSAGE CHEESE
48,1006,424,1006,424,1036,48,1036,JUMBO SAUSAGE CHEESE
```

47,1053,282,1053,282,1080,47,1080,GARLIC CHEESE
491,917,546,917,546,962,491,962,QTY
503,963,521,963,521,993,503,993,1
502,1006,522,1006,522,1036,502,1036,1
501,1053,518,1053,518,1082,501,1082,1
611,895,674,895,674,924,611,924,PRICE
616,925,673,925,673,954,616,954,(RM)
604,963,662,963,662,992,604,992,3.10
602,1008,663,1008,663,1038,602,1038,3.10
603,1054,662,1054,662,1082,603,1082,2.00
749,896,800,896,800,922,749,922,AMT
738,926,799,926,799,955,738,955,(RM)
737,964,796,964,796,994,737,994,3.10
738,1007,798,1007,798,1039,738,1039,3.10
738,1056,796,1056,796,1084,738,1084,2.00
822,919,884,919,884,956,822,956,TAX
840,963,879,963,879,994,840,994,SR
839,1010,880,1010,880,1038,839,1038,SR
840,1055,879,1055,879,1082,840,1082,SR
284,1095,367,1095,367,1125,284,1125,TOTAL:
500,1096,523,1096,523,1124,500,1124,3
731,1095,802,1095,802,1128,731,1128,8.20
205,1152,658,1152,658,1191,205,1191,TOTAL SALES (EXCLUDING GST) :
493,1203,658,1203,658,1238,493,1238,DISCOUNT :
391,1261,656,1261,656,1298,391,1298,SERVICE CHARGE :
472,1311,658,1311,658,1344,472,1344,TOTAL GST :
484,1363,656,1363,656,1403,484,1403,ROUNDING :
140,1417,624,1417,624,1459,140,1459,TOTAL SALES (INCLUSIVE OF GST)
524,1475,626,1475,626,1506,524,1506,CASH
474,1518,625,1518,625,1553,474,1553,CHANGE
730,1152,802,1152,802,1191,730,1191,7.74
731,1204,805,1204,805,1237,731,1237,0.00
733,1259,801,1259,801,1289,733,1289,0.00
733,1313,802,1313,802,1342,733,1342,0.46

730,1367,800,1367,800,1397,730,1397,0.00
733,1416,801,1416,801,1453,733,1453,8.20
729,1474,801,1474,801,1505,729,1505,8.20
162,1624,443,1624,443,1661,162,1661,GST SUMMARY
173,1700,328,1700,328,1732,173,1732,TAX CODE
222,1742,276,1742,276,1773,222,1773,SR
375,1699,403,1699,403,1733,375,1733,%
379,1743,401,1743,401,1774,379,1774,6
434,1703,586,1703,586,1736,434,1736,AMT (RM)
515,1741,583,1741,583,1773,515,1773,7.74
515,1798,584,1798,584,1838,515,1838,7.74
614,1702,760,1702,760,1741,614,1741,TAX (RM)
694,1742,760,1742,760,1775,694,1775,0.46
692,1801,765,1801,765,1838,692,1838,0.46
99,1955,829,1955,829,1989,99,1989,GOODS SOLD ARE NOT RETURNABLE, THANK YOU.
366,2010,560,2010,560,2044,366,2044,RE-PRINT
304,1802,408,1802,408,1835,304,1835,TOTAL :
733,1523,800,1523,800,1550,733,1550,0.00

REFERENCES

Anaconda | Anaconda Distribution [WWW Document], n.d. . Anaconda. URL https://www.anaconda.com/products/distribution (accessed 11.18.22).

Detect text in images | Cloud Vision API | Google Cloud [WWW Document], n.d. URL https://cloud.google.com/vision/docs/ocr (accessed 11.17.22).

Downloads - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition [WWW Document], n.d. URL https://rrc.cvc.uab.es/?ch=13&com=downloads (accessed 2.28.21).

Du, J., 2018. Understanding of Object Detection Based on CNN Family and YOLO. J. Phys. Conf. Ser. 1004, 012029. https://doi.org/10.1088/1742-6596/1004/1/012029

General Data Protection Regulation (GDPR) Compliance Guidelines [WWW Document], n.d. . GDPR.eu. URL https://gdpr.eu/ (accessed 1.12.22).

Huang, Z., Chen, K., He, J., Bai, X., Karatzas, D., Lu, S., Jawahar, C.V., 2019. ICDAR2019 Competition on Scanned Receipt OCR and Information Extraction, in: 2019 International Conference on Document Analysis and Recognition (ICDAR). pp. 1516–1520. https://doi.org/10.1109/ICDAR.2019.00244

IEEE Xplore - Conference Table of Contents [WWW Document], n.d. URL https://ieeexplore.ieee.org/xpl/conhome/8961318/proceeding (accessed 2.27.21).

Lin, T.-Y., Goyal, P., Girshick, R., He, K., Dollár, P., 2018. Focal Loss for Dense Object Detection.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., Berg, A.C., 2016. SSD: Single Shot MultiBox Detector. ArXiv151202325 Cs 9905, 21–37. https://doi.org/10.1007/978-3-319-46448-0_2

Mahdi, F.P., Motoki, K., Kobashi, S., n.d. OPEN Optimization technique combined with deep learning method for teeth recognition in dental panoramic radiographs. Sci. Rep. 13.

Pagero | Digitalise and streamline your business processes, 2018. . Pagero. URL https://www.pagero.com/ (accessed 2.27.21).

Patel, S., Bhatt, D., 2020. Abstractive Information Extraction from Scanned Invoices (AIESI) using End-to-end Sequential Approach. ArXiv200905728 Cs.

Pettagam Tharindu Rukshan Ubewikkrama, 2020. Automatic invoice Data identification with relations.

Project Jupyter [WWW Document], n.d. URL https://jupyter.org (accessed 11.18.22).

Redmon, J., Divvala, S., Girshick, R., Farhadi, A., 2016. You Only Look Once: Unified, Real-Time Object Detection, in: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, Las Vegas, NV, USA, pp. 779–788.

https://doi.org/10.1109/CVPR.2016.91

Redmon, J., Farhadi, A., n.d. YOLOv3: An Incremental Improvement 6.

Results - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition [WWW Document], n.d. URL https://rrc.cvc.uab.es/?ch=13&com=evaluation&task=1 (accessed 2.27.21).

Roboflow: Give your software the power to see objects in images and video [WWW Document], n.d. URL https://roboflow.com/ (accessed 11.17.22).

Smith, R., 2007. An Overview of the Tesseract OCR Engine, in: Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2. Presented at the Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2, IEEE, Curitiba, Parana, Brazil, pp. 629–633. https://doi.org/10.1109/ICDAR.2007.4376991

Staudemeyer, R.C., Morris, E.R., 2019. Understanding LSTM -- a tutorial into Long Short-Term Memory Recurrent Neural Networks. ArXiv190909586 Cs.

Tasks - ICDAR 2019 Robust Reading Challenge on Scanned Receipts OCR and Information Extraction - Robust Reading Competition [WWW Document], n.d. URL https://rrc.cvc.uab.es/?ch=13&com=tasks (accessed 2.27.21).

Tesseract OCR, 2022.

ultralytics/yolov5, 2022.

Wang, C.-Y., Bochkovskiy, A., Liao, H.-Y.M., 2022. YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors.

Yin, W., Kann, K., Yu, M., Schütze, H., 2017. Comparative Study of CNN and RNN for Natural Language Processing.