

**DEVELOPMENT OF IMAGE  
RECOGNITION SYSTEM FOR ANALOGUE  
METER'S READING DETECTION**

**CHONG YUE JIET**

**UNIVERSITI TUNKU ABDUL RAHMAN**

**DEVELOPMENT OF IMAGE RECOGNITION SYSTEM FOR  
ANALOGUE METER'S READING DETECTION**

**CHONG YUE JIET**


**A project report submitted in partial fulfilment of the  
requirements for the award of Bachelor of Engineering  
(Honours) Electrical and Electronic Engineering**

**Lee Kong Chian Faculty of Engineering and Science  
Universiti Tunku Abdul Rahman**

**April 2022**

**DECLARATION**

I hereby declare that this project report is based on my original work except for citations and quotations which have been duly acknowledged. I also declare that it has not been previously and concurrently submitted for any other degree or award at UTAR or other institutions.

Signature :   
\_\_\_\_\_

Name : Chong Yue Jiet  
\_\_\_\_\_

ID No. : 17UEB02386  
\_\_\_\_\_

Date : 22 April 2022  
\_\_\_\_\_

**APPROVAL FOR SUBMISSION**

I certify that this project report entitled **DEVELOPMENT OF IMAGE RECOGNITION SYSTEM FOR ANALOGUE METER'S READING DETECTION** was prepared by **CHONG YUE JIET** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic at Universiti Tunku Abdul Rahman.

Approved by,

Signature : *Chua Kein Huat*  
\_\_\_\_\_  
Supervisor : Assoc. Prof. Ir. Dr. Chua Kein Huat  
\_\_\_\_\_  
Date : 22 April 2022  
\_\_\_\_\_

The copyright of this report belongs to the author under the terms of the copyright Act 1987 as qualified by Intellectual Property Policy of Universiti Tunku Abdul Rahman. Due acknowledgement shall always be made of the use of any material contained in, or derived from, this report.

© 2022, Chong Yue Jiet. All right reserved.

## ACKNOWLEDGEMENTS

First and foremost, I would like to express my sincere gratitude towards my FYP supervisor, Assoc. Prof. Ir. Dr. Chua Kein Huat, for giving me an opportunity to collaborate with him for my final year project and his patience, guidance, as well as recommendations during the project period. Besides, Dr. Chua has provided a lot of efforts and assistance in the success of publishing a conference paper for this project.

In addition, I would like to thanks my family members and friends who have given me encouragements and technical supports which led to my successful completion of this project.

## ABSTRACT

Digitization is one of the major components in Industrial Revolution 4 (IR 4.0). It provides a lot of benefits to the industrial, such as increasing productivity, better data visualisation, simplifying parameters control, etc. Although many meters nowadays come with the smart Internet of Things (IoT) features, which provides real-time monitoring and data storing, many industries still prefer to continue using the existing analogue meters in their manufacturing plants as replacing the existing analogue meters with the cloud-connected digital meters can be very costly especially for industrial grade meters. In this project, a cost-effective image recognition system to capture and digitize the analogue meter's readings using deep learning model (SSD MobileNet) as well as optical character recognition (Tesseract) was demonstrated. The deep learning model has been trained with a dataset of 750 images and was used to detect the region of interest (meter's readings). The OCR is used to convert the readings to string datatype. Besides, the image processing techniques via OpenCV library has been implemented for enhancing the quality of the ROI. The programme developed has been transferred and executed on the Raspberry Pi microcomputer with camera module attached to an analogue water meter. The results show that the accuracies of the deep learning model and OCR are 95% and 91%, respectively. In addition, the memory occupation of the deep learning model is about 10 MB, which is suited the embedded system with limited memory capacity.

## TABLE OF CONTENTS

<b>DECLARATION</b>		<b>ii</b>
<b>APPROVAL FOR SUBMISSION</b>		<b>iii</b>
<b>ACKNOWLEDGEMENTS</b>		<b>v</b>
<b>ABSTRACT</b>		<b>vi</b>
<b>TABLE OF CONTENTS</b>		<b>vii</b>
<b>LIST OF TABLES</b>		<b>x</b>
<b>LIST OF FIGURES</b>		<b>xi</b>
<b>LIST OF SYMBOLS / ABBREVIATIONS</b>		<b>xiv</b>
<b>LIST OF APPENDICES</b>		<b>xv</b>
<b>CHAPTER</b>		
<b>1</b>	<b>INTRODUCTION</b>	<b>1</b>
1.1	General Introduction	1
1.2	Importance of the Study	2
1.3	Problem Statement	2
1.4	Aim and Objectives	3
1.5	Project Overview	3
1.6	Scope and Limitation of the Study	4
1.7	Contribution of the Study	4
1.8	Outline of the Report	4
<b>2</b>	<b>LITERATURE REVIEW</b>	<b>6</b>
2.1	Introduction	6
2.2	Image Recognition Concepts	6
2.2.1	General Concept	6
2.2.2	Digital Image Capturing	7
2.2.3	Methods of Image Recognition	8
2.2.4	General Algorithms	8
2.3	Deep Learning	9
2.3.1	Introduction	9



	2.3.2	Convolutional Neural Network (CNN)	10
	2.3.3	Training Mechanism of CNN	12
2.4		Image Processing Techniques	14
	2.4.1	OpenCV	14
	2.4.2	Grayscale	15
	2.4.3	Image Filtering	15
	2.4.4	Thresholding	16
	2.4.5	Morphological Transformation	18
	2.4.6	Edge Detection	19
	2.4.7	Contour Detection	20
	2.4.8	Orientation Correction	20
	2.4.9	ROI Extraction	22
2.5		Character Recognition	22
2.6		Summary	23
<b>3</b>		<b>METHODOLOGY AND WORK PLAN</b>	<b>24</b>
	3.1	Introduction	24
	3.2	Methodology	24
	3.3	Software	25
	3.3.1	Overview	25
	3.3.2	Platforms	25
	3.3.3	Deep Learning Model	28
	3.3.4	Extraction of ROI	31
	3.3.5	Image Processing	32
	3.3.6	Optical Character Recognition	32
	3.3.7	Uploading Data to Cloud	33
	3.3.8	Additional Libraries	33
	3.4	Hardware	34
	3.4.1	Configuration	34
	3.4.2	Raspberry Pi 4 Model B	35
	3.4.3	Camera Module	36
	3.5	Cost of Components	37
	3.6	Planning and Milestones	38
	3.6.1	Project Milestones	38
	3.6.2	Project Schedule and Gantt Chart	38

3.7	Summary	40
<b>4</b>	<b>RESULTS AND DISCUSSION</b>	<b>41</b>
4.1	Introduction	41
4.2	Software Simulation	41
4.2.1	Deep Learning Model	41
4.2.2	Extraction of ROI	43
4.2.3	Image Processing	46
4.2.4	Character Recognition	47
4.3	Hardware Implementation	47
4.3.1	Setup	47
4.3.2	Installation on Meter	49
4.4	Cloud and Real-time Database	50
4.5	Accuracy Evaluation	50
4.6	Summary	52
<b>5</b>	<b>CONCLUSION AND RECOMMENDATIONS</b>	<b>53</b>
5.1	Conclusion	53
5.2	Recommendations for future work	54
	<b>REFERENCES</b>	<b>55</b>
	<b>APPENDICES</b>	<b>59</b>

**LIST OF TABLES**

Table 3.1: Specifications of Raspberry Pi 4 Model B	36
Table 3.2: Cost of Components	37
Table 3.3: Project Milestones	38
Table 3.4: Project Schedule for FYP Part 1	38
Table 3.5: Project Schedule for FYP Part 2	39
Table 4.1: Results of Accuracy Evaluation	51

**LIST OF FIGURES**

Figure 2.1: Image Represented as Arrays of Number	6
Figure 2.2: CMOS Sensor Block Diagram	7
Figure 2.3: General Algorithm for Image Recognition	8
Figure 2.4: Structure of Perceptron	10
Figure 2.5: Supervised Learning	10
Figure 2.6: Architecture of CNN	11
Figure 2.7: Backpropagation Algorithm	12
Figure 2.8: Convolution Operation	13
Figure 2.9: Pooling Operation	13
Figure 2.10: Illustration of Grayscale vs RGB in Arrays	15
Figure 2.11: Process of 2D Convolution	16
Figure 2.12: Difference in Intensity between Background and Object	17
Figure 2.13: Erosion Operation	18
Figure 2.14: Dilation Operation	18
Figure 2.15: CHAIN_APPROX_NONE vs CHAIN_APPROX_SIMPLE	20
Figure 2.16: Parameters Returned by minAreaRect	21
Figure 2.17: Mask for ROI Extraction	22
Figure 3.1: Flowchart of the System Development	24
Figure 3.2: Operations of the Meter's Readings Detection System	25
Figure 3.3: Spyder IDE	26
Figure 3.4: LabelImg Annotation Software	27
Figure 3.5: Firebase Console	27
Figure 3.6: Examples of Meter Images in the Dataset	28

Figure 3.7: Flowchart of Deep Learning Model Training via TensorFlow	30
Figure 3.8: XML file of the Labelled Image	31
Figure 3.9: Block Diagram of the System	35
Figure 3.10: Raspberry Pi 4 Model B	35
Figure 3.11: Raspberry Pi Camera Module	37
Figure 3.12: Camera Connection with CSI Port	37
Figure 3.13: Gantt Chart for FYP Part 1	39
Figure 3.14: Gantt Chart for FYP Part 2	40
Figure 4.1: Training Process of Deep Learning Model on Google Colab	42
Figure 4.2: Graphs of various Loss Functions	42
Figure 4.3: Detection Box Visualizing Function	43
Figure 4.4: Inferencing of various Types of Meters	43
Figure 4.5: Inferenced Meter Image with Label Hidden	44
Figure 4.6: Output of <i>inRange</i> function	44
Figure 4.7: Mask for ROI Extraction	45
Figure 4.8: Output of <i>bitwise AND</i> Function	45
Figure 4.9: ROI Extracted	45
Figure 4.10: ROI being Enlarged	46
Figure 4.11: Grey-scaled ROI	46
Figure 4.12: Thresholded ROI	46
Figure 4.13: Output of Tesseract OCR	47
Figure 4.14: Hardware Setup	48
Figure 4.15: Hardware Connected to Power Bank	48
Figure 4.16: Hardware Installation on Water Meter	49
Figure 4.17: Data Collected in Firebase	50

Figure 4.18: Data Tabulated using Microsoft Excel

**LIST OF SYMBOLS / ABBREVIATIONS**

$G_x$	1 <sup>st</sup> derivative in x-axis direction
$G_y$	1 <sup>st</sup> derivative in y-axis direction
ADC	analogue to digital converter
AI	artificial intelligence
ASCII	American standard code for information interchange
CMOS	complementary metal oxide semiconductor
CNN	convolutional neural network
CPS	cyber physical system
CSI	camera serial interface
HDMI	high-definition multimedia interface
HPF	high pass filter
IDE	integrated development environment
IoT	internet of things
I/O	input or output
IR 4.0	industrial revolution 4.0
KNN	K nearest neighbour
LPF	low pass filter
MP	megapixel
OCR	optical character recognition
OpenCV	open source computer vision
OS	operating system
PIL	python imaging library
ReLU	Rectified Linear Unit
RGB	red, green and blue colourspace
ROI	region of interest
SCADA	supervisory control and data acquisition
SNR	signal to noise ratio
USB	universal serial bus

**LIST OF APPENDICES**

APPENDIX A: Acceptance of Conference Paper	59
APPENDIX B: Code of Deep Learning Model Training (Google Colab)	60
APPENDIX C: Code of TensorBoard	61
APPENDIX D: Code of Image Recognition Software Developed	62



## CHAPTER 1

### INTRODUCTION

#### 1.1 General Introduction

Industrial Revolution is the transition of agricultural and handicraft economy to industrial and machine-manufacturing economy. It has been divided into four periods, namely the 1<sup>st</sup> revolution for steam power (1760-1820), the 2<sup>nd</sup> revolution for electrical power usage (1820-1900), the 3<sup>rd</sup> revolution for electronic and computing (1900s) and the 4<sup>th</sup> revolution for cyber-physical system in manufacturing (IR 4.0).

The ultimate goal of IR 4.0 is to attain a fully automated smart factory, which the manufacturing processes are fully automated and the number of operators is minimized. The backbone of a smart factory is the Cyber Physical System (CPS). The CPS integrates the parameters of the processes in physical world with the software components via digitization to achieve the machine-to-machine and machine-to-human interactions (Zhou, Liu and Zhou, 2016). This system mainly consists of the latest technologies and advancements, namely the Internet of Things, Big Data Analytics and Artificial Intelligence.

Internet of Things (IoT) is defined as the connections between machines, which are embedded with instrumentation, sensors, actuators and antenna, via the internet to the cloud and SCADA system (Oracle, n.d.). The IoT provides a channel for the linkage of physical world to the digital world which enables data collection, data exchange and real-time update and monitoring. Due to the centralization of data, the operators can visualize the conditions of the manufacturing processes conveniently, without inspecting the machines one after another; thus, increasing the working efficiency.

Big Data Analytics is the process of evaluating and interpreting a tremendous amount of data, either in raw or structured, to unveil the patterns, trends and correlations among the data (tableau, n.d.). As the technology advances, the speed of processor and the volume of the memory storage increase rapidly, thus making the big data analytics, in the size of gigabytes to zettabytes possible.

Artificial Intelligence (AI) describes the capability of an electronic device to execute tasks associated with intelligent beings. When a system is integrated with AI, it has the potential of learning, reasoning, generalizing and deciding (Copeland, 2020). Thus, in IR 4.0 and smart factory, the AI is expected to make precise decisions based on the results from big data analytics to control the operations of the machines and enhance the automation within the factory.

## **1.2 Importance of the Study**

Industrial Revolution 4.0 and Digitization are the current trends in industrial and manufacturing sectors. They provide a lot of benefits to the industry. According to Caylar and Noterdaeme (2016), IR 4.0 can increase the productivity up to 45% through automation, reduce machine downtime up to 50%, increase forecast accuracy to 85% and reduce production cost up to 40%.

In Malaysia, the Government of Malaysia initiated the TN50 plan for digitization and forecasted that the country's economy will be uplifted to 2 trillion USD in 8 years (Mohammad et al., 2018). However, according to a study done by Idris (2019), only 40% of the respondents implied that Malaysia is ready to embrace the IR 4.0. The challenges faced by the companies include large investment, technical and compatibility issues and insufficient qualification of workers.

This research primarily focuses on providing an easier and more feasible way for digitization in the factory. Besides, the proposed solution, in terms of hardware and software, will be in minimized cost to ensure that the implementation of Internet of Things (IoT) is economical and in large scale to promote IR 4.0.

## **1.3 Problem Statement**

Digitization is one of the most important steps towards Industry 4.0 transformation. Although many meters nowadays are equipped with IoT features that enable the digitation of readings and real-time monitoring, many industries are still preferably using analogue meters for their manufacturing plants. One of the reasons is that some IoT meters are still very costly and that may hinder the owners or directors from the intention to implement the transition.

Besides, various types of smart meters from different suppliers are required for different manufacturing processes. A divergence in the standards of electronic communication and protocols used by each meter becomes a challenge to integrate them to an already-built control system of the factory. It is not economical for the factory to undergo a major overhaul.

Moreover, most of the workers in the factory are already used to deal with the analogue meters and they are lack of the knowledge for using and calibrating some complex smart meters. Thus, the requirement of extra training and investment on the workers turns out to be an obstacle for the progress towards IR 4.0 (Rymarczyk, 2020).

#### **1.4 Aim and Objectives**

The aim of this research is to develop a cost-effective image recognition system to capture and monitor the analogue meter's readings and send the data to the cloud system of the factory as a part of digitization and automation.

The objectives of the project are as follows:

- i. To investigate the feasibility of image recognition system for analogue meter's reading detection.
- ii. To develop the image recognition system.
- iii. To evaluate the functions and performance of the system.

#### **1.5 Project Overview**

There is a wide variety of tools available for deep learning and image recognition such as Python, MATLAB, TensorFlow, etc. Besides, there are many different libraries and techniques involved. In the first stage, different approaches were investigated and the most suitable approach, in terms of cost effectiveness, complexity and usability would be implemented.

In the second stage, the software for image recognition was developed and trained. Next, the performance and accuracy of the image recognition were assessed and optimized during the simulation process.

Subsequently, the integration of hardware and software was performed. Lastly, the image recognition system developed in previous stage was tested and installed at the real meter to observe its real-world performance.

## **1.6 Scope and Limitation of the Study**

This project predominantly focuses on detecting and extracting the Region of Interest (ROI), i.e. the analogue meter's readings, recognizing and converting the numerical information to digital format using deep learning, image processing technique and optical character recognition engine.

The limitation of the study is on the diversity of the meters being recognized. Due to the constraint in various parameters involving the image processing, computer vision and character recognition, the system developed is only applicable to detect and convert the numerical reading. Thus, the image recognition system is not universal.

## **1.7 Contribution of the Study**

This project demonstrates the method of digitizing an analogue meter's reading using deep learning, image processing and optical character recognition. Besides, the solution proposed is cost effective and easy to be implemented which eventually enhances the digitization and adoption of IR 4.0 in a factory.

## **1.8 Outline of the Report**

The project report consists of 5 main chapters, namely introduction, literature review, methodology, results and discussion and conclusion.

The first chapter, introduction, mainly focuses on providing the details and benefits of digitization, problem statement, aim and objectives as well as overview of the project.

The second chapter is literature review. This chapter provides technical reviews and interpretations on the topics related to image recognition. Besides, in-depth reviews and explanations on the functions and working principles involved in the deep learning and image processing are delivered.

Next, the third chapter, methodology, provides the details regarding the configuration of the proposed system, the processes and flowchart of the system, the software and hardware involved, as well as the list and cost of components. In addition, the work plan, project milestones, schedule of each activity and the Gantt Charts are illustrated.

The fourth chapter is results and discussion. This chapter mainly illustrates and explains the outcomes of each sub-system as mentioned in the

methodology. In-depth analysis have been performed on the results, in terms of software, hardware and accuracy evaluation, to verify and ensure that the solutions provided and system developed are able to achieve the aim and objectives of this project.

Lastly, the fifth chapter is conclusion. This chapter provides an overall summary of the project as well as the conclusion to the work carried out and results obtained during the research. Moreover, the recommendations for future work is delivered.

## CHAPTER 2

### LITERATURE REVIEW

#### 2.1 Introduction

This chapter provides technical reviews and interpretations on the topics related to image recognition. Firstly, the general concept, methods and algorithms of the image recognition are introduced and elaborated. Subsequently, the tools and software available for the image and character recognition are explored. An in-depth reviews and explanations on the functions involved in the image recognition processes and their working principles are investigated.

#### 2.2 Image Recognition Concepts

##### 2.2.1 General Concept

Image recognition refers to the ability of an electronic device in detecting and identifying certain objects based on their features and properties. Its working principles are analogous to that of how humans or animals' brains identify and recognize an object. The visual performance of humans is exceptional due to the capability of our brain in features extraction, contours detection, classification, contextual knowledge and paralleling processing (Mujtaba, 2020). A digital image is represented in arrays of number, called pixels. Thus, the computer analyses the characteristics of an image depending on the pattern and correlation among the numerical data using mathematical function. Figure 2.1 shows an image represented as arrays of number.

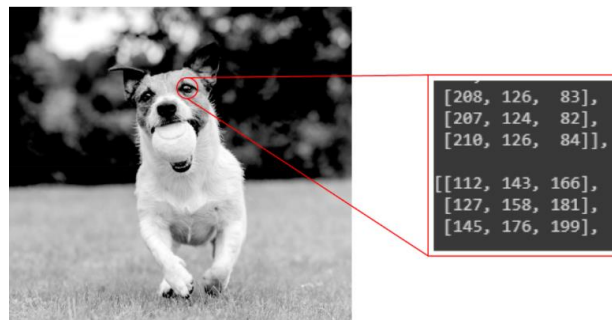


Figure 2.1: Image Represented as Arrays of Number (Mujtaba, 2020)

### 2.2.2 Digital Image Capturing

An image sensor is used to carry out the digital image capturing process. It is an electronic device that consists of photosensitive elements which can convert the analogue optical signal to digital signal. The operation of an image sensor is based on a physics phenomenon known as photoelectric effect, which electrons are emitted during the bombardment of photons on the surface of the photosensitive material. The relationship of the magnitudes of light signal and electrical signal is proportional. Nowadays, the photosensitive element is made up of Complementary Metal Oxide Semiconductor (CMOS). Figure 2.2 shows the block diagram of a CMOS sensor.

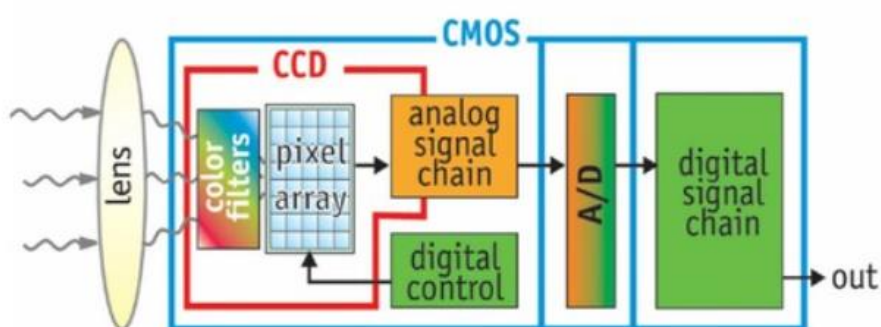


Figure 2.2: CMOS Sensor Block Diagram (University of Maine, n.d.)

A CMOS image sensor mainly consists of colour filter, pixel cell array, 2D array driver, control logic, analogue-to-digital converter (ADC) and output data bus. Firstly, the environmental light is focused by an optical lens placed above the sensor. Next, the light passes through the colour filter and illuminates the pixel array in the CMOS. Only the light with similar colour to the colour filter's segment is able to penetrate through, thus the colour of the image is represented by each respective pixel. The photodiode of each pixel performs the conversion of light signal to electrical signal.

Subsequently, the 2D array driver will scan the pixel array row by row to extract the electrical signal and transmit the respective signal to the amplifier to increase the signal-to-noise (SNR) ratio. Thereafter, the signal undergoes digitization by the ADC and finally output as binary data. (Utmel Electronic, 2020)

### 2.2.3 Methods of Image Recognition

There are two major approaches for image recognition: image processing and deep learning. The image processing method comprises of extracting key features from an image via several mathematical functions and input parameters. It is very effective in pixel-based recognition utilizations such as template matching, colour-based detection, image segmentation, shape extraction and blob analysis.

On the other hand, deep learning incorporates the neural network, for instance Convolutional Neural Network (CNN) to enable self-executive learning and training process of a model based on a collection of images and data set to such an extent that the features of an image can be automatically identified with satisfactory accuracy (MathWorks, n.d.).

Generally, the image processing technique will be implemented first to determine whether the results and accuracy can fulfil the requirement as the deep learning approach requires a huge number of data set (in terms of thousands) and complicated framework to achieve decent accuracy.

### 2.2.4 General Algorithms

The basic procedures involve in image recognition and region detection are shown in Figure 2.3. The algorithm starts with capturing the image of interest. A camera module with high resolution, e.g. 5 MPs and above, is preferred to ensure the readability of the image captured in different conditions and thus the accuracy for character recognition is reliable.

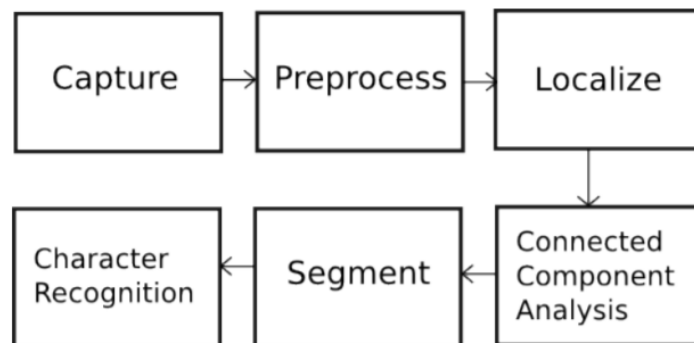


Figure 2.3: General Algorithm for Image Recognition (Sajjad, 2012)



Secondly, the image captured will undergo pre-processing. It comprises a set of algorithms to remove unwanted noise and provide enhancement to the quality of the image. Besides, it involves conversion of the image's format, i.e. from colourful to grayscale and to binary as the algorithms associated in the subsequent procedure can only take in single channel formatted parameters and arrays.

Next, localization will be performed to detect and filter the contours or borders of the components within the image. Generally, the contours of the useful components will be in connected or grouped orientation; thus, connected components analysis is implemented for identifying the region of interest (ROI). After the ROI has been identified, segmentation process is executed for extracting the ROI by cropping and the background or unwanted components are eliminated (Sajjad, 2012).

Finally, the character recognition is performed on the ROI which contains the characters or numbers, to digitize them into ASCII format for further usage, e.g. mathematical calculation. There are two different solutions for character recognition. The first solution is the well-developed Optical Character Recognition (OCR) engine such as Tesseract which was developed by HP and Google. The second solution is via the self-develop deep learning model using neural network such as Convolutional Neural Network (CNN) and K Nearest Neighbour (KNN).

## **2.3 Deep Learning**

### **2.3.1 Introduction**

Deep Learning is a subtype of machine learning which provides the computers with ability to mimic how human thinks, making decisions as well as recognizing and classifying objects. The general structure of deep learning algorithm is motivated by human's brain and neurons. It consists of multiple layers of neural networks, with each basic unit called Perceptron (Figure 2.4). The perceptrons are connected to each other via mathematical method, as analogous to the connections of brain's neurons (Oppermann, 2019).

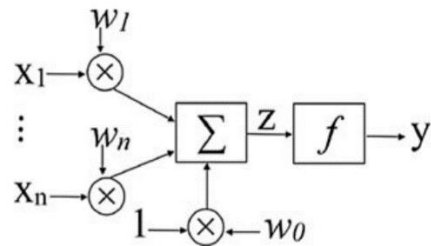


Figure 2.4: Structure of Perceptron (Taud and Mas, 2018)

The deep learning algorithm for image recognition is generally classified as supervised learning. It means both the input data and their labels are provided to the learning algorithm. The labels act as referencing answers to the algorithm which the prediction from the deep learning model is compared with the labels. Subsequently, the error between the label and the prediction is calculated and this value will be used to fine-tune the model for improving its accuracy, also known as training process (IBM Cloud Education, 2020). Figure 2.5 illustrates the idea of supervised learning.

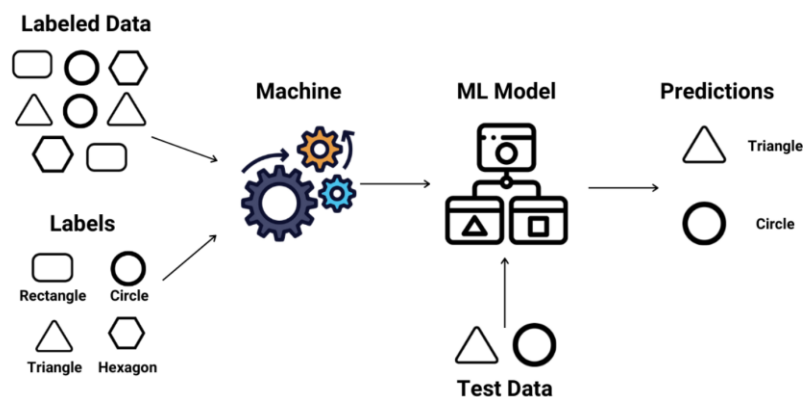


Figure 2.5: Supervised Learning (Kumar, 2021)

### 2.3.2 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a neural network architecture under deep learning. It has remarkable performance and accuracy on the tasks involving image classification and object detection. Besides, it possesses the ability of automatic feature extraction; meaning, the features of an object inside an image are identified and extracted automatically via the convolution operation based on the entire pixels of that image. Thus, the time-consuming

manual feature extraction procedure is eliminated and the image pre-processing required is greatly reduced (IBM Cloud Education, 2020).

The general structure of the CNN comprises of image input, convolutional layers, pooling layers, fully-connected layers and predicted outputs. Figure 2.6 illustrates the architecture and the components of CNN.

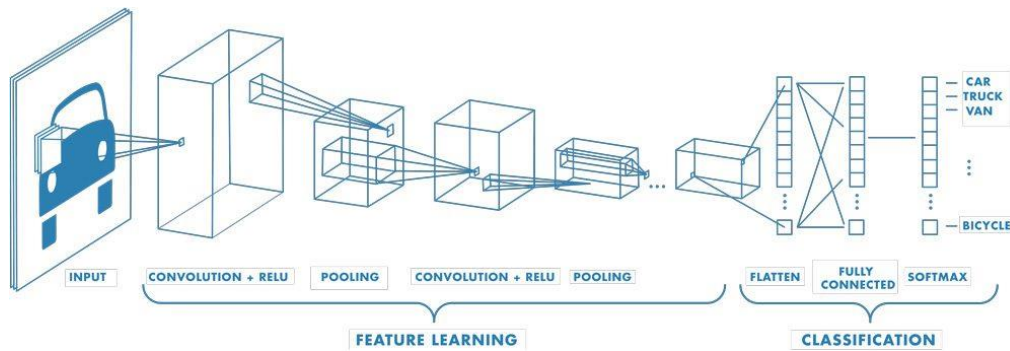


Figure 2.6: Architecture of CNN (MathWorks, n.d.)

The convolutional layer mainly consists of feature extractor, made up of two-dimensional (2D) kernel or filter which can be mathematically expressed as matrix. The kernel is being convoluted with the input image pixels which returns the feature map that contains the extracted features, such as edges, colour, brightness, etc. In addition, each convolutional layer can identify one type of features. Thus, for the model to detect various types of objects with different characteristics, multiple layers, in terms of hundreds or thousands are required (MathWorks, n.d.).

The pooling layer performs downsampling on the convoluted feature which reduces its dimension and number of parameters while retaining the dominant features. The benefits of downsampling include decreasing the computational power required for model training and reducing the size of memory required. The pooling techniques are categorized into two types, namely max pooling and average pooling. In max pooling, the maximum value of the pixels under the kernel is extracted whereas in average pooling, the average value among the pixels is extracted (Saha, 2018).

The fully-connected layer is the layer that learns and performs classification based on the features extracted in the previous layers. In this layer, each perceptron in one layer is fully connected with the perceptrons in both its

previous and next layers. In addition, weights are present between the connections. They are values, typically real numbers, that will be multiplied with the output of each perceptron before sending the outputs to next layer. These weights are to be fine-tuned during the training process (IBM Cloud Education, 2020).

### 2.3.3 Training Mechanism of CNN

The technique being implemented to perform training on a CNN deep learning model is known as the backpropagation algorithm. This algorithm consists of two processes, namely forward propagation and backward propagation.

During the forward propagation, the input parameters pass through all the layers of the CNN architecture and a prediction in terms of probability will be generated as output. Next, the predicted output is compared with the labels provided and returns the error values.

Subsequently, the error values will be used to update and fine-tune the weights of the neural network in the fully-connected layers. The backpropagation algorithm is repeated for multiple times and when each time the weights are updated, it is considered as one step. The training process will be terminated when the pre-defined criteria are fulfilled, such as constant loss function value (McGonagle, et al., n.d.). Ultimately, the accuracy of the model is maximized whereas the error is minimized. Figure 2.7 illustrates the backpropagation algorithm.

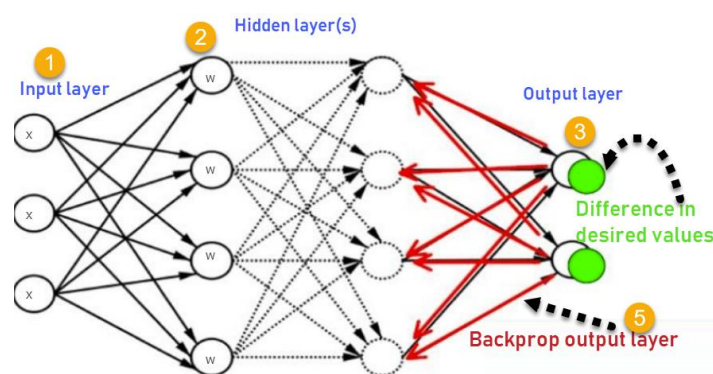


Figure 2.7: Backpropagation Algorithm (Johnson, 2022)

### 2.3.3.1 Forward Propagation

Forward propagation refers to the process of generating a prediction based on the input parameters by passing through every layer in the neural network. In the forward propagation, the input image will first undergo convolution operation based on the kernel defined. The output of the convolution is the dot product between the kernel and the input image's region covered by the kernel as shown in Figure 2.8. After this operation has been executed, one particular feature of the image is extracted.

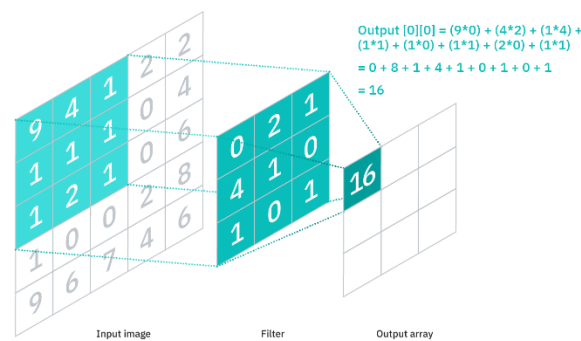


Figure 2.8: Convolution Operation (IBM Cloud Education, 2020)

Next, the pooling operation will be performed on the convoluted output to reduce its spatial size. Generally, the max pooling technique is implemented, which the maximum pixel value under the kernel is retained whereas the others are eliminated as shown in Figure 2.9. In addition to reduce of size, the max pooling technique provides noise suppressing to the image (Saha, 2018). The convolution and pooling operations work in pair and multiple pairs are present to extract different types of feature from the image.

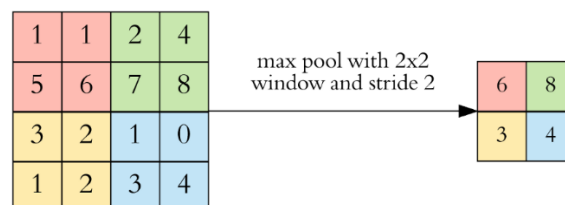


Figure 2.9: Pooling Operation (Dertat, 2017)

Subsequently, the 2D matrix is converted to a column vector which is then fed into the fully-connected layers of neural network. The neural network,

comprises of perceptrons, will classify the image and generate the prediction based on the weights and activation function such as ReLU. This mechanism of generating the prediction with respect to probability is known as Softmax Classification technique (Saha, 2018) .

### **2.3.3.2 Backward Propagation**

The backward propagation is the mechanism to fine-tune the weights of the neural network based on the error values. Error is defined as the difference between the targeted value and the actual value. It is calculated by comparing the value of the predicted output from the model with the value from the labels or reference answers.

After obtaining the error values, the gradient descent algorithm is implemented to compute the amount of value change applied on the current weights. The gradient descent algorithm is known as optimizer in machine learning as it calculates the optimal values to the weights which maximize the accuracy of the model's prediction (Johnson, 2022).

Ultimately, after consecutive steps of training on the deep learning model, the difference between the predicted value and the reference value is minimized and the model is capable to make reliable predictions based on the input image.

## **2.4 Image Processing Techniques**

### **2.4.1 OpenCV**

Open Source Computer Vision Library, known as OpenCV, is a free and open source library developed by Intel from the year 2000. It incorporates diverse algorithms for computer vision and machine learning which are mainly used for image processing and object identification. The OpenCV library comprises of more than 2500 programming functions to perform various tasks such as colour space conversion, edge detection, orientation and angle modification, contours detection, etc.

Besides, the functions and algorithms of OpenCV library are written in C++ and well optimized. It also provides various wrappers for multiple programming languages such as Java, MATLAB and Python; thus, the coding

developed by the programmers can be executed on different platforms, including low-processing power application, in a rapid manner with minimized execution time and power consumption. (OpenCV Organisation, n.d.)

### 2.4.2 Grayscale

An image captured by the CMOS image sensor is in RGB (Red, Green, Blue) colour space due to the presence of colour filter as discussed in Section 2.3.1. However, most of the OpenCV functions for image processing can only take in the single channel image (Gray) instead of three channels image (RGB); thus, the initial step will be the grayscale.

Figure 2.10 shows that each pixel in grayscale is represented by single digit which denotes the intensity. Contrarily, each pixel of a colour image consists of 3 digits representing the intensity of the respective RGB colour channels. Therefore, by implementing the equation (2.1), the colour space conversion can be achieved (OpenCV Organisation, 2020).

$$Y = 0.299R + 0.587G + 0.114B \quad (2.1)$$



Figure 2.10: Illustration of Grayscale vs RGB in Arrays (Rune, 2020)

### 2.4.3 Image Filtering

Filtering can be performed on an image via two-dimensional (2D) convolution operation. When an image is filtered by a low pass filter (LPF), the noise within the image can be eliminated and the image is smoothed; whereas when it is filtered by a high pass filter (HPF), the edges in the image are sharpen. To perform 2D convolution, a kernel is needed to represent the type of filter. It is an  $n \times n$  matrix as shown below.

$$\begin{aligned} \text{Kernel} &= K \\ &= \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \end{aligned} \quad (2.2)$$

The equation for the 2D convolution is

$$g(x, y) = \sum_{i=-a}^a \sum_{j=-b}^b K(i, j) f(x + i, y + j) \quad (2.3)$$

where

$x$  and  $y$  are the coordinate of the pixel

$i$  and  $j$  are the row and column element of the kernel respectively

Basically, the operation of convolution is to sum up the product of each element of the kernel and the respective element of the image which is encircled by the kernel. Next, the sum will be the value for the centre pixel as compounded by the kernel for the filtered image. The process is illustrated in Figure 2.11.

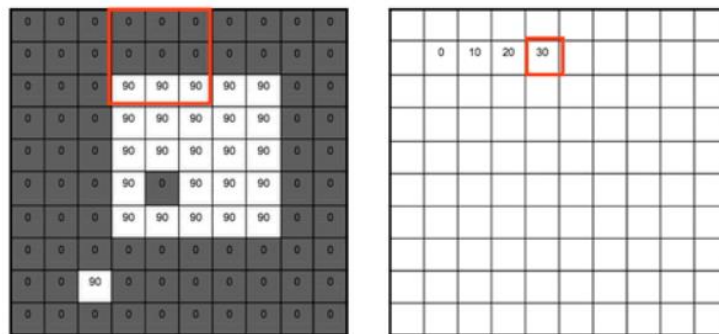


Figure 2.11: Process of 2D Convolution (Stanford University, n.d.)

#### 2.4.4 Thresholding

Thresholding is the process of binarizing a gray-scaled image, i.e. converting the value of each pixel to either 0 or maximum depending on the threshold value defined, as represented by the equation (2.4). The purpose of thresholding is to fade out or remove unnecessary elements within the image such as background



and reflection. This can be achieved as there is a variation between the background and object in terms of pixel intensity as illustrated by Figure 2.12.

$$f(x) = \begin{cases} 255, & x > \text{threshold} \\ 0, & x < \text{threshold} \end{cases} \quad (2.4)$$

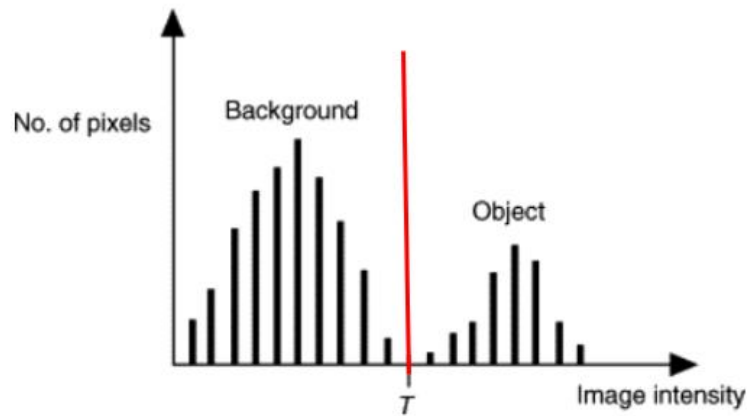


Figure 2.12: Difference in Intensity between Background and Object (Rogowska, 2009)

Besides, there are two types of thresholding, i.e. simple thresholding and adaptive thresholding. In simple thresholding, a threshold with constant value is defined and it is applied to every pixel of the image. However, the simple thresholding's result is inferior when the image lighting condition is fluctuating. On the other hand, the adaptive thresholding uses an algorithm to vary the threshold value for each pixel depends on its surrounding region. Thus, the thresholding result is optimized for different illumination. The Otsu's binarization algorithm is utilized to determine the threshold value,  $t$ , that keeps the weighted within-class variance in minimum (equation 2.5) (OpenCV Organisation, n.d.).

$$\sigma_w^2(t) = q_1(t)\sigma_1^2(t) + q_2(t)\sigma_2^2(t) \quad (2.5)$$

where

$\sigma$  is the variance of each class

$q$  is the probability of each class

### 2.4.5 Morphological Transformation

Morphological transformation is a set of algorithms to process an image based on the shape of the components. It mainly consists of four operations which are erosion, dilation, closing and opening. The algorithms perform comparison on the pixel in the input image with its surrounding pixels to decide the value of the corresponding pixel in the output image (Sreedhar, 2012).

The erosion operation adjusts the pixel of the output image to zero (black) if any of its surrounding pixels under the kernel is zero. Thus, the overall white pixels in the image will be reduced and converted to black pixels as illustrated in Figure 2.13. This operation is effective for the removal of white noises.

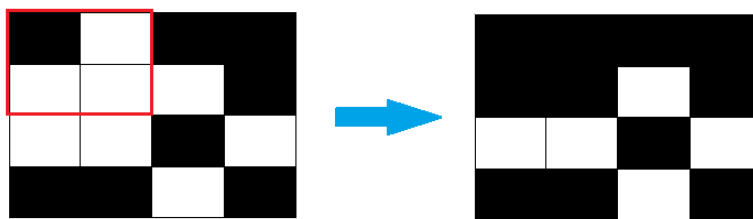


Figure 2.13: Erosion Operation

The dilation operation is opposite to the erosion. It sets the pixel of the output image to one (white) if any of the adjacent pixels within the kernel is one. Therefore, the white pixels in the image will be increased after dilation which can be observed in Figure 2.14.

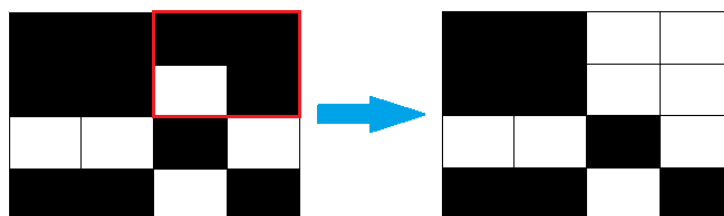


Figure 2.14: Dilation Operation

Closing and opening are the operations that combine both erosion and dilation. Closing is erosion comes after dilation whereas opening is dilation comes after erosion. These two operations are very effective and convenient for extending the shape, fixing the broken lines and patching up small holes in the

image. The operation should be selected depending on the background of the image (white or black).

#### 2.4.6 Edge Detection

The Canny edge detector developed by John F. Canny is considered as one of the most popular edge detectors used in image processing due to its accuracy and flexibility. There are five stages in the detection algorithms, namely smoothing, finding derivatives, calculating gradient magnitude and orientation, non-maximum suppression and hysteresis.

Smoothing is the first stage as the edge detection is vulnerable to noise which can cause errors or inaccuracy. In the Canny's algorithm, a Gaussian kernel (in  $n \times n$  matrix) is used to filter the noise (equation 2.6).

$$K_{i,j} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}} \quad (2.6)$$

where

$$x = i - (k + 1), 1 \leq i \leq 2k + 1$$

$$y = j - (k + 1), 1 \leq j \leq 2k + 1$$

$$k = \frac{n-1}{2}$$

Next, the 1<sup>st</sup> derivatives for the edges are being calculated for the horizontal direction and vertical direction, represented as  $G_x$  and  $G_y$ . From these values, the edge gradient and direction can be calculated using the equations 2.7 and 2.8, respectively.

$$Gradient = \sqrt{G_x^2 + G_y^2} \quad (2.7)$$

$$\theta = \tan^{-1} \left( \frac{G_y}{G_x} \right) \quad (2.8)$$

Subsequently, every pixel is checked to determine whether it is a local maximum within the adjacent pixels in the gradient direction. The pixel which is local maximum will be examined next stage whereas the other will be suppressed. Lastly, in the hysteresis stage, a maximum threshold and minimum

threshold value will be defined. If an edge with gradient larger than the maximum threshold, it is classified as an edge, whereas if the gradient is smaller than the minimum threshold, it is eliminated (Teli, 2019).

#### 2.4.7 Contour Detection

Contours detection is used to detect the borders of a component or the outline of a shape in the image. It is the fundamental for various applications including object recognition, image classification and region of interest extraction. The contour of a specific object is defined as the boundary pixels with identical characteristics such as intensity and colour. Thus, by comparing a pixel with its adjacent pixels based on their characteristics, the contour can be detected (Gong et al., 2018).

There are two different algorithms for contour detection, namely the CHAIN\_APPROX\_SIMPLE algorithm and CHAIN\_APPROX\_NONE algorithm. The CHAIN\_APPROX\_NONE will detect the contour of a shape and store all the points of that particular contour; thus, the contour will be indicated with line segments. On the other hand, the CHAIN\_APPROX\_SIMPLE method will truncate the contour's line segments and preserve their vertices. Therefore, this method indicates the contour with discrete points, utilizes less memory and executes faster (Mallick, n.d.). The comparison of both methods is illustrated in Figure 2.15. The former method is more suitable for irregular shape whereas the latter for regular shape.



Figure 2.15: CHAIN\_APPROX\_NONE vs CHAIN\_APPROX\_SIMPLE

#### 2.4.8 Orientation Correction

The orientation of an image may not be upright and straight all the time due to the misplacement of camera. With the image tilted or rotated in different directions, the accuracy of the object detection or character recognition is

greatly affected. Thus, it is necessary to apply correction to the orientation of that particular image. There are two elements involve in this process, namely the angle detection and affine transformation.

Firstly, a reference component such as rectangle or square in the image is selected via the contour detection. Next, by implementing a shaping function such as the `minAreaRect`, the characteristics of the rectangle such as the size, tilted angle and position of centre can be obtained as illustrated in Figure 2.16 below.

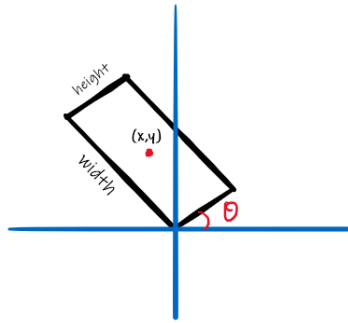


Figure 2.16: Parameters Returned by `minAreaRect`

Subsequently, a rotation matrix (equation 2.9) is to be defined using the parameters obtained from the previous step to perform the rotation. As in programming aspect, the mathematical function is executed in matrix calculation. Lastly, the `warpAffine` function performs matrix multiplication of the rotation matrix and the vector of the initial point to complete the geometric transformation (Mallick, n.d.).

$$M = \begin{bmatrix} \alpha & \beta & (1 - \alpha)c_x - \beta c_y \\ -\beta & \alpha & \beta c_x + (1 - \alpha)c_y \end{bmatrix} \quad (2.9)$$

where

$$\alpha = k \cos(\theta)$$

$$\beta = k \sin(\theta)$$

$c_x$  and  $c_y$  are the coordinates of the centre

$k$  is the scaling factor

### 2.4.9 ROI Extraction

Once the region of interest (ROI) has been identified, it is to be extracted and cropped to appropriate size (depending on the application) for further operation such as character recognition. A mask is required to perform the ROI extraction. It is an image arrays with identical size to the original image and contains the area of the ROI in white (255) and the external region in black (0) as shown in Figure 2.17.

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	255	255	255	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0

Figure 2.17: Mask for ROI Extraction

Next, the bitwise AND operation will be performed on the original image with the mask. The bitwise AND operation returns the pixel value of the original image if the pixel of the mask is not 0, else the output pixel will be 0 (black) as shown in the equation 2.10. Thus, the ROI in the original image is retained and the external region is replaced with black pixel. The ROI is said to be extracted.

$$output(x, y) = \begin{cases} input(x, y), & \text{if } mask(x, y) \neq 0 \\ 0, & \text{otherwise} \end{cases} \quad (2.10)$$

where

$x$  and  $y$  are the coordinates of the pixel

## 2.5 Character Recognition

Optical Character Recognition (OCR) is a technology being utilized for converting the characters in an image to the machine language such as ASCII format. This conversion is necessary for the digitization purpose and it enables further operations to be performed on the characters, for examples mathematical calculation, text manipulation and uploading to cloud.

Tesseract is a free and open source OCR engine developed by Hewlett Packet (HP) and Google using neural network. This engine can recognize more than 100 languages. Moreover, it is written and compiled in C and C++, thus it can be executed in various platforms such as Linux, Windows and MacOS (Patel et al, 2012). Besides, to run the Tesseract engine in Python, a library wrapper known as PyTesseract is required. It consists of the Tesseract class in Python language and is able to deal with various type of images, for instance jpg, png, etc as supported by the Pillow library.

## **2.6 Summary**

In this chapter, the theories, concepts and working principles regarding the image recognition have been discussed. There are many tools and functions provided by the OpenCV library which can be utilized to determine and extract the region of interest. Subsequently, the ROI can be processed by the optical character recognition engine to obtain the digitized readings. The knowledge and idea acquired from this chapter will be applied into the project subsequently.

## CHAPTER 3

### METHODOLOGY AND WORK PLAN

#### 3.1 Introduction

This chapter provides the details regarding the configuration of the proposed system, the processes and flowchart of the system, the software and hardware involved, as well as the list and cost of components. Besides, the work plan, project milestones, schedule of each activity and the Gantt Charts are illustrated subsequent to the methodology. They deliver the information about the progress and tasks that have been achieved during the execution of this project.

#### 3.2 Methodology

This section describes the general procedures and tools utilized in developing the meter's reading detection system. The development process is illustrated by a flowchart as shown in Figure 3.1.

Firstly, the images of various types of analogue meters had been collected and labelled manually to form the dataset for the training of the deep learning model. Next, the deep learning model, namely SSD MobileNet V2, was trained using the TensorFlow architecture via Google Colab.

Subsequently, the software for the system, comprised of image inferencing, region of interest (ROI) extraction, image processing, optical character recognition and clouding was developed. Thereafter, the software was transferred to the Raspberry Pi microcomputer for execution. It was installed on the real meter for readings digitization and accuracy evaluation as real-world application.

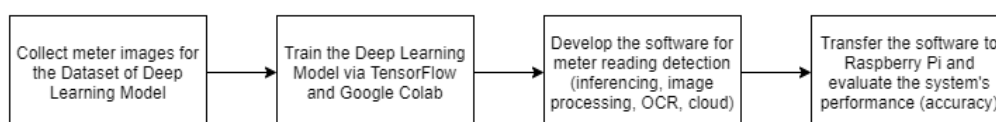


Figure 3.1: Flowchart of the System Development



### 3.3 Software

#### 3.3.1 Overview

The software of meter's reading detection system consisted of six major operations as shown in Figure 3.2. Firstly, the self-trained deep learning model was loaded into the system's RAM. Next, the image of the meter can either be captured directly using the camera integrated on the microcomputer or loaded from the storage. The deep learning model loaded previously was used to perform inference on the image to localize the meter reading's area. Subsequently, the region of interest (ROI) containing the readings was extracted. Thereafter, the ROI underwent image processing for noise removal and quality enhancement. Lastly, the optical character recognition (OCR), namely Tesseract, was used to recognise the digits and convert them to ASCII format which would then be uploaded to the cloud (Firebase).

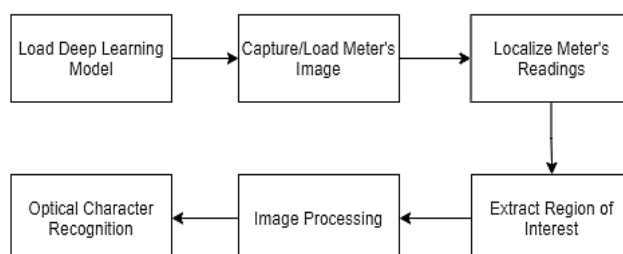


Figure 3.2: Operations of the Meter's Readings Detection System

#### 3.3.2 Platforms

##### 3.3.2.1 Python

The programming language used in this project is Python. It is a high-level object-oriented programming language which offers user-friendly and high code readability experience. Python supports multiple libraries, modules and packages for various applications such as image processing, data analysis, hardware integration, etc. Besides, the Python codes are standardized in different platforms. Thus, the Python programme developed in the Windows platform can be directly migrated and executed in the Raspberry Pi OS without extra modifications.

### 3.3.2.2 Google Colab

The Google Colab is a Python development environment, provided freely by Google Inc that runs entirely on the cloud which the programmer can access the Colab notebook using internet browsers. Due to its cloud-based characteristic, many benefits are available. Firstly, it enables auto-saving on cloud so that the programme being developed are always up-to-date. Besides, it is integrated with pre-installed libraries, especially for deep learning such as TensorFlow and Keras. Moreover, it provides high amount of RAM and powerful GPU which enables acceleration and optimisation for the model training process (Nelson and Hoover, 2020).

### 3.3.2.3 Spyder IDE

Spyder is an integrated development environment (IDE) specifically for Python programming language. It is an open-source and cross-platform software which provides comprehensive development tools and functionalities such as variable explorer, auto error detection, keywords suggestion, codes documentation and debugging console. The Spyder IDE user interface is shown in Figure 3.3.

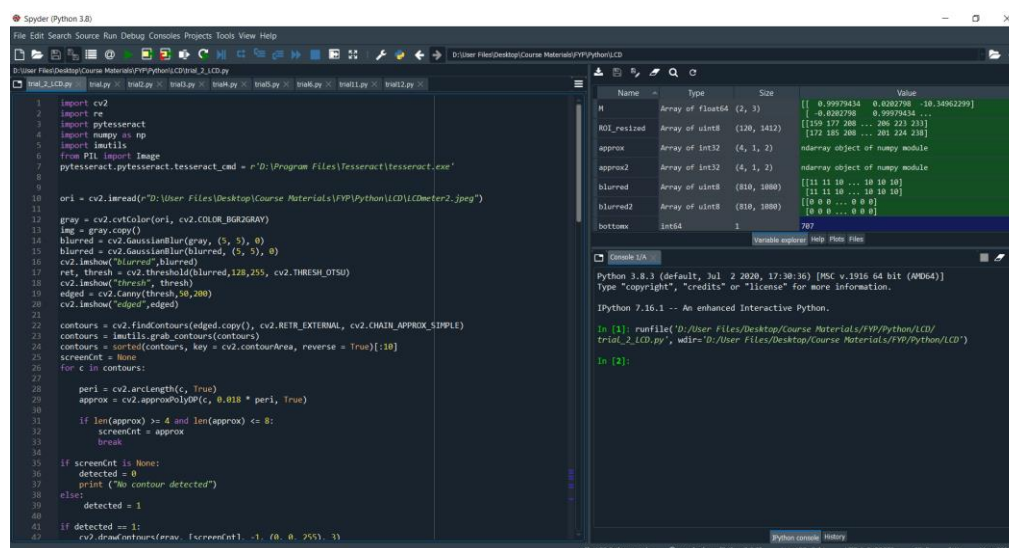


Figure 3.3: Spyder IDE

### 3.3.2.4 LabelImg

LabelImg is a free image annotation software developed by Tzutalin in 2015. It has a simple and user-friendly interface which enables quick image labelling

and class defining. Besides, it supports two different labelling formats, namely PASCAL VOC and YOLO (Tzatalin, 2015). In this project, the TensorFlow architecture used the PASCAL VOC format. Figure 3.4 shows the user interface of the LabelImg software.



Figure 3.4: LabelImg Annotation Software

### 3.3.2.5 Firebase

Firebase is an online platform developed by Google, primarily for cloud services. It provides various functions such as real-time database, web hosting, cloud storage and users analytics (Moroney, 2017). The programme developed in this project used the Firebase's real-time database for uploading and storing the digitized meter's readings which enabled the real-time monitoring function. Figure 3.5 shows the console of the Firebase platform.

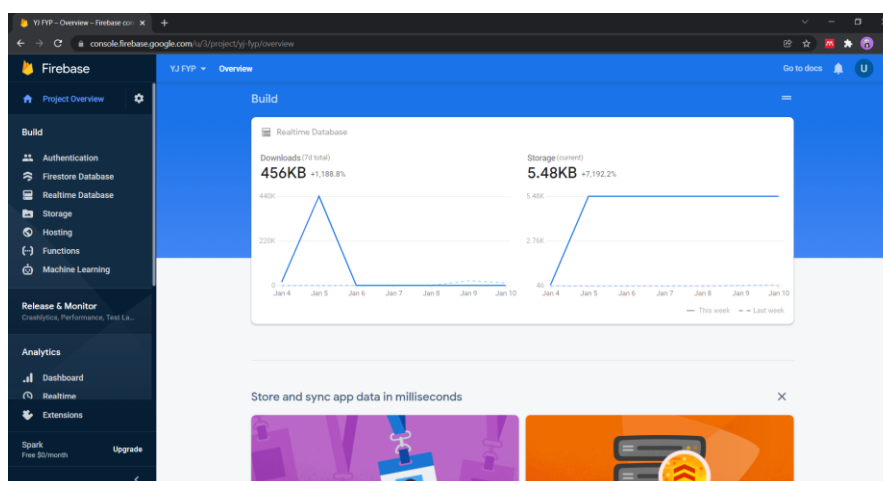


Figure 3.5: Firebase Console

### 3.3.3 Deep Learning Model

The deep learning model used in the image recognition system was based on the SSD MobileNet V2. This model was chosen because it required lesser processing power and storage than other conventional models such as VGG, ResNet, Inception and AlexNet (Howard et al., 2017). It was suitable to be deployed in resource-limited devices, such as a low-cost mobile phone or embedded system, e.g., Raspberry Pi.

#### 3.3.3.1 Dataset

A dataset containing 750 meter images has been prepared and then labelled with the labelling software. A class of ‘readings’ has been defined. A portion of the images are generated with augmentation techniques to increase the variability. The dataset is divided with 8:2 ratio: 600 images for training and 150 images for validation. Figure 3.6 shows the examples of meter images within the dataset.



Figure 3.6: Examples of Meter Images in the Dataset

#### 3.3.3.2 Model Training

The deep learning model was trained using the TensorFlow library via Google Colab with GPU accelerator. The GPU accelerator was activated as it provided 10 times faster training speed for each step of the model as compared to using CPU only. Figure 3.7 shows the overall flowchart of the training procedures.

Firstly, the dataset for the deep learning model had been prepared as discussed in previous subsection. Each image being labelled with the `labelImg` software came with a XML file which contained the information about the image and the coordinate of the labelling box, as shown in Figure 3.8.

Next, all the XML files of the images within the dataset were combined into one file by converting them into CSV format. Subsequently, a `tf.record` file was generated based on the CSV file created previously. The `tf.record` file is a binary formatted file specialised for the TensorFlow architecture. It has been optimised for the TensorFlow model's training usage which improves the training performance and reduces the dataset file size (Gamauf, 2018).

Besides, a label map, the file used to define the name of classes in the dataset, was created. In this project, the label map contained only one class, namely "readings".

Next, the configuration file (`pipeline.config`) for the training process was set up. It contained the model configuration and hyperparameters such as batch size, learning rate, activation function, `tf.record` file location and training checkpoint file location. In this project, the batch size considered was 32 due to the size of RAM provided by Google Colab.

In addition, the transfer learning approach was implemented using the pre-trained object detection model based on the COCO dataset to save the training time, reduce the size of dataset and improve the performance of the neural network, as it transfers the knowledges gained from previous trainings to current task (Torrey and Shavlik, n.d.).

After all the requirements had been fulfilled, the training of the model was started. During the training process, new checkpoint file which contained the latest progress was created for every 1000 training steps. The TensorBoard was used to convert the numerical information in the checkpoint file to graphical information in terms of charts. Thus, the training parameters such as various types of losses, learning rate and steps per second can be easily visualized.

When the total loss of the model had reached a constant value, i.e. 0.2 in this case, the deep learning model was considered saturated and well-trained. Thus, the training process was interrupted and the training steps were around 15000. The training was stopped at this moment to prevent the over-fitting issue.

Lastly, the latest checkpoint file was converted to the final model (*saved\_model.pb*) using the model exporter programme. The *saved\_model.pb* file can be used to perform inferencing on the meter images and thus localizing the reading region.

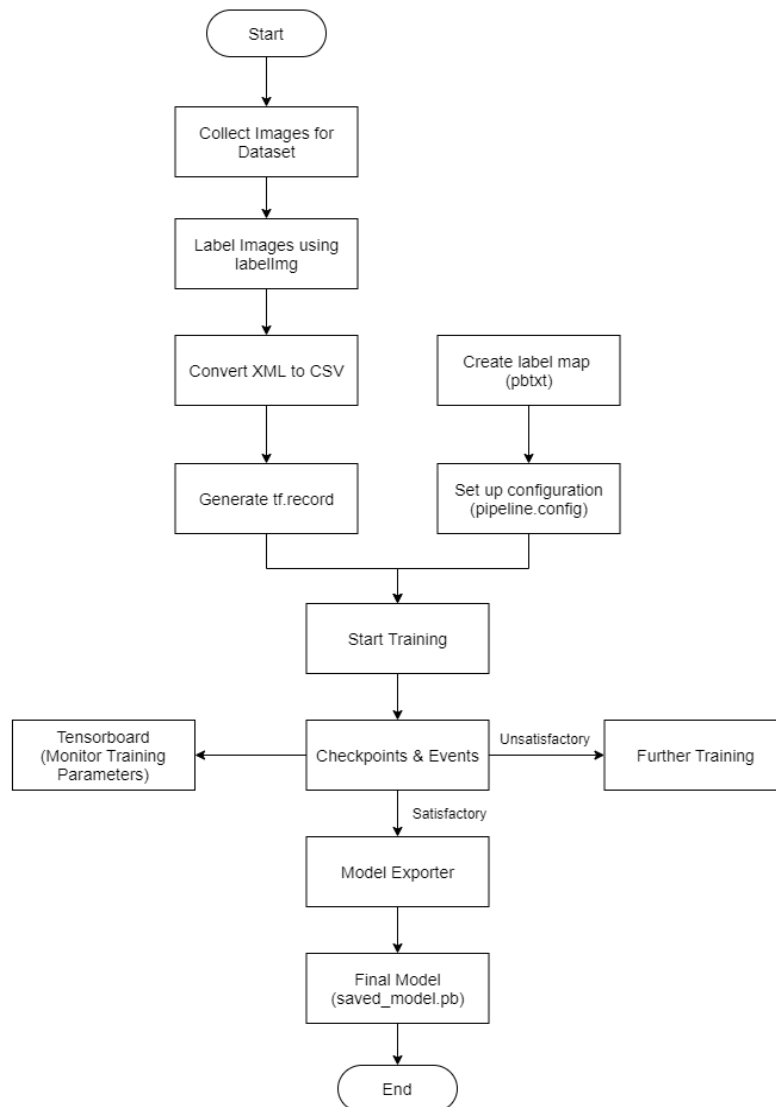


Figure 3.7: Flowchart of Deep Learning Model Training via TensorFlow

```

<annotation>
  <folder>train</folder>
  <filename>1.jpg</filename>
  <path>D:\User Files\Desktop\readings\train\1.jpg</path>
  <source>
    <database>Unknown</database>
  </source>
  <size>
    <width>530</width>
    <height>624</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>readings</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
      <xmin>164</xmin>
      <ymin>279</ymin>
      <xmax>287</xmax>
      <ymax>319</ymax>
    </bndbox>
  </object>
</annotation>

```

Figure 3.8: XML file of the Labelled Image

### 3.3.4 Extraction of ROI

The region of interest (ROI) is the area within the detection box where the meter's reading located. It is necessary to extract the ROI by removing the area outside the detection box while retaining the area within the box so that unintended characters would not be recognized during the OCR process. This was achieved using the contour detection functions of OpenCV. By setting the *skip\_labels* parameter to be true during the inference, the 'readings' labelling can be removed.

First, the meter image that had been inferenced by the deep learning model was converted to HSV format using the colour space conversion. As the detection box was green, the lower and upper limits of the HSV values were defined as (40,200,200) and (50,255,255) respectively. Next, the image was compared with the limits defined using the *inRange* function to obtain a mask containing only the box outline and position.

Thereafter, the *findContours* function was used to locate the box and a new mask with the region of interest in white (255) was created using *drawContours*. Next, the *bitwise AND* operation was performed on the meter image with the new mask. Subsequently, the positions of vertices of the ROI were obtained using the NumPy min and max functions. These vertices values were essential for cropping out the ROI.

### 3.3.5 Image Processing

Image processing is to remove the noise and enhance the quality of the extracted ROI. This step is crucial to improve the accuracy of the OCR. According to the documentation (Tesseract, n.d.), the Tesseract OCR engine prefers the characters in black and white, and the characters should be resized to at least 300 dpi of resolution.

The ROI was first enlarged to three times its original size using the *resize* function in OpenCV. The superior interpolation method of *INTER\_CUBIC* was used to enlarge the image. It performed bicubic interpolation over a 4×4-pixel neighbourhood (Asthana, 2014) and outputted a clear and large image as desired.

Next, the enlarged ROI was converted to grayscale using the *cvtColor* function. It was then thresholded to binarize the grey-scaled image into black and white only by converting the value of each pixel to either 0 or maximum.

Moreover, the sharpening process can be carried out using the *filter2D* function if necessary, by defining an appropriate matrix utilizing the *numpy.array* function.

### 3.3.6 Optical Character Recognition

Optical Character Recognition, OCR is a technology being utilized for converting the characters in an image to the machine language such as ASCII format. In this paper, a free version of OCR engine, known as Tesseract is used. To run the Tesseract engine in Python, a library wrapper known as PyTesseract, which consists of the Tesseract class in Python language is required.

The Tesseract engine can only operate with the image in PIL format. However, the ROI as extracted and processed in previous steps was represented in arrays form. Thus, it was mandatory to convert the ROI format using the *Image.fromarray* function from the PIL library. Thereafter, the *pytesseract.image\_to\_string* function was executed with the ROI in PIL format as an input parameter and the function would return the recognized characters as a string.

There were various configurations that can be defined for the *pytesseract.image\_to\_string* function to achieve better performance depending on the usage and character orientation. First was the *lang* parameter which refers to the Tesseract dataset being used for the recognition process. Officially, there



are three types of datasets (.traindata file) with different sizes, accuracy and execution speed (Tesseract, 2017). Thus, the user can choose to use either type according to the requirements. Next was the segmentation mode which depends on the orientation of the characters (Tesseract, n.d.), in this case the *psm 7* was used, meaning the characters were in the format of single line of text. Furthermore, for recognizing digits only as in the meter's readings, the *outputbase digits* config was employed. In addition, the *re.sub* function was used to eliminate unnecessary spacing and symbols recognized.

### 3.3.7 Uploading Data to Cloud

The meter's reading after being digitized, would be uploaded to the Firebase's real-time database for real-time monitoring. In Python, an additional library namely *python-firebase*, was installed to perform the Firebase cloud accessing. The data being uploaded to the cloud were in the following format:

Date: DD-MM-YYYY, Time: hh:mm:ss, Reading: XXXX

where

DD is day; MM is month; YYYY is year

hh is hours; mm is minutes; ss is second

XXXX is the digitized meter's reading

### 3.3.8 Additional Libraries

#### 3.3.8.1 Imutils

Imutils is the supplemental package that aids the operation of OpenCV. It provides various functions to deal with the output from OpenCV such as contour grabbing, contour sorting and transformation.

#### 3.3.8.2 Numpy

Numpy is a library that handles arrays and matrices with multiple dimensions. It provides algorithms for array creation, transposing, reshaping, obtaining min and max values in an array, etc. This library is essential in image processing as

the images are represented by different dimensions of numeric arrays in Python and OpenCV. (Numpy Organisation, 2021)

### **3.3.8.3 PIL**

PIL stands for Python Imaging Library. It is the main image manipulation library for Python which provides functions such as image opening, enhancing, format conversion and exporting. The Tesseract character recognition engine only supports the image in PIL format, thus this library is necessary to convert the image array from OpenCV format.

### **3.3.8.4 re**

Regular Expression (re) is a module that allows the user to define certain rules or patterns for the string. It is mainly used to search, split or replace the particular characters in a string with reference to the patterns defined. For example, unnecessary spacing in a string can be eliminated using the re function.

### **3.3.8.5 Pytesseract**

Pytesseract is a library wrapper for the Tesseract engine. It enables the execution of Tesseract engine within the Python IDE using a sets of python coding and simplifies the work of a developer.

## **3.4 Hardware**

### **3.4.1 Configuration**

The microcomputer employed in the image recognition system was the Raspberry Pi 4 Model B. It was powered by a 240 V AC to 5 V DC Adapter with 15 W power rating. A camera module was attached to the microcomputer via the CSI port for capturing the image of the analogue meter. Besides, the microcomputer was connected to WiFi and upload the digitized readings to the cloud system, namely Firebase. Moreover, a monitor together with a pair of keyboard and mouse were connected to the microcomputer during the set up and debugging process to allow the developer interfacing with the microcomputer. Figure 3.9 shows the overall block diagram of the system proposed.

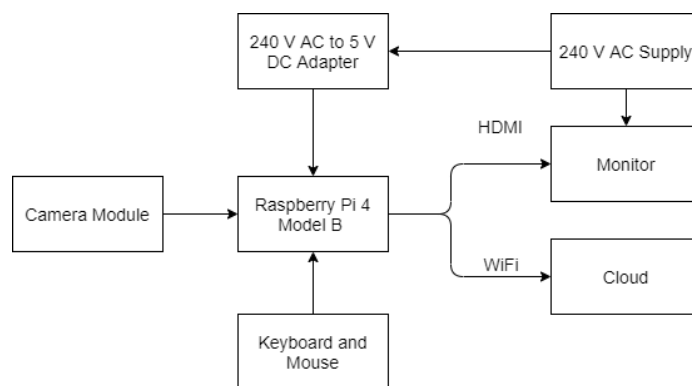


Figure 3.9: Block Diagram of the System

### 3.4.2 Raspberry Pi 4 Model B

The Raspberry Pi 4 Model B is a microcomputer developed by the UK-based Raspberry Pi Foundation. It was released in June 2019 and is currently the latest version in the Pi series. This model is integrated with its own processor, memory chip (RAM), connectivity modules, various I/O ports and camera peripheral. It can offer computing performance comparable to a typical entry-level laptop and supports Python programming. A micro SD card is used as a hard drive to store the operating system, software and files of the microcomputer.

Besides, the Model B supports both Bluetooth and WiFi wireless connections. The Bluetooth version supported is 5.0 and the WiFi is dual-band 2.4/5 GHz. Thus, it is very convenience for the online cloud access and internet-of-thing (IoT) purposes. The power consumption of the Model B is 15 W and is supplied by an AC to DC adapter with output of 5 V/3 A. To configure and control the microcomputer, a set of USB keyboard and mouse are required.

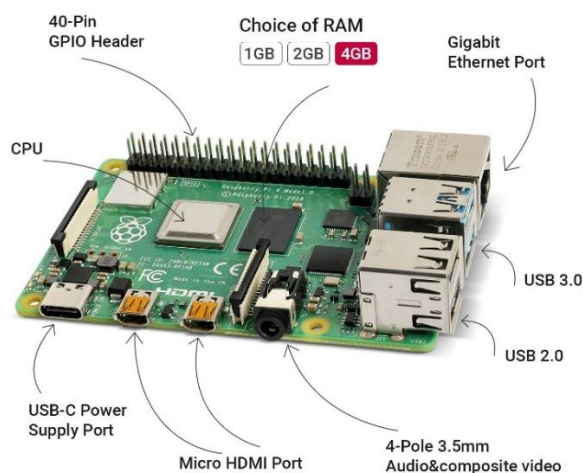


Figure 3.10: Raspberry Pi 4 Model B

Table 3.1: Specifications of Raspberry Pi 4 Model B (RaspberryPi, 2020)

Components	Details
Processor	Broadcom BCM2711 (ARM v.8)
Memory	LPDDR4 (1/2/4/8 GB)
Connectivity	2.4/5 GHz IEEE 802.11b/g/n/ac wireless Bluetooth v5.0 4x USB ports Gigabits Ethernet 40 GPIO pins
Video & Sound	Micro HDMI, camera peripheral, audio port
Power supply	15 W, 5 V 3 A DC
Environment	0 – 50 °C

### 3.4.2.1 Operating System

The official operating system (OS) for Raspberry Pi 4 Model B is the Raspberry Pi OS. It is a Debian-based OS developed by the Raspberry Pi Foundation and is highly optimized for the ARM CPU. The OS has a built-in Package Manager which consists of various applications such as web browser, document editor, etc. The Python IDE such as Spyder is also supported on the Raspberry Pi OS. The OS is to be downloaded into an SD Card using Windows and subsequently installed on the microcomputer.

### 3.4.3 Camera Module

The Raspberry Pi Camera Module is a camera board officially released by the Raspberry Pi Foundation. It consists of a 5 MP OV5647 CMOS sensor developed by OmniVision. The module is weighted at 3 g and with a size of 6 cm<sup>2</sup>. Besides capturing a 5 MP image, it is also capable of recording a video with 1080p30 and 720p60 resolutions. The camera module is connected to the Raspberry Pi 4 Model B microcomputer via the high data rate CSI port integrated on the board.

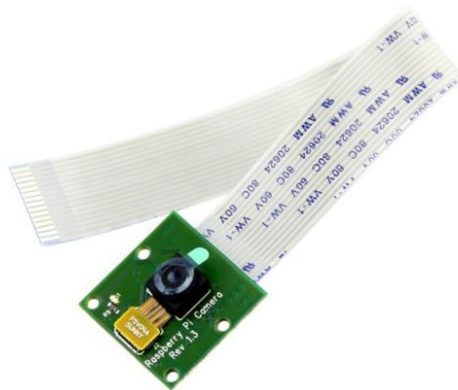


Figure 3.11: Raspberry Pi Camera Module

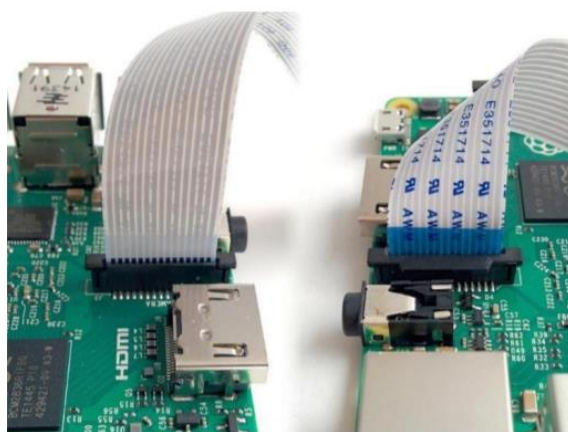


Figure 3.12: Camera Connection with CSI Port

### 3.5 Cost of Components

Table 3.2 shows the cost of each component for this project.

Table 3.2: Cost of Components

Component	Quantity	Price (RM)
Raspberry Pi 4 Model B (4 GB RAM)	1	240.00
Power Adapter (240 V AC to 5V 3A DC)	1	35.00
MicroSD card (32 GB)	1	28.90
Raspberry Pi Camera Module	1	29.90
<b>Total Price (RM)</b>		<b>333.80</b>

### 3.6 Planning and Milestones

#### 3.6.1 Project Milestones

There are 6 milestones for the project as shown in Table 3.3

Table 3.3: Project Milestones

Label	Milestone
M1	Preliminary investigation for customer's requirements and problem statement
M2	Literature review and research
M3	Data collection
M4	Model Development
M5	Model Evaluation and Optimization
M6	Integration of Software and Hardware

#### 3.6.2 Project Schedule and Gantt Chart

Table 3.4 shows the project schedule and activities for FYP part 1 and Figure 3.13 shows the Gantt Chart for Part 1.

Table 3.4: Project Schedule for FYP Part 1

Milestone	Activities	Duration (weeks)
M1	A1 Discuss with supervisor and understand the scope of project	1
M1	A2 Preliminary investigation and project planning	1
M2	A3 Perform literature review and research on related topics for image processing	6
M3	A4 Collect the required data and images	2
M4	A5 Develop the software model for image recognition system	7
M5	A6 Initial model testing and obtaining preliminary results	3
-	A7 Documentation and report writing	2

Activities	Weeks													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A1	■													
A2		■												
A3			■	■	■	■	■	■	■					
A4							■	■	■					
A5							■	■	■	■	■	■		
A6											■	■	■	
A7													■	■

Figure 3.13: Gantt Chart for FYP Part 1

Table 3.5 shows the project schedule and activities for FYP part 2 and Figure 3.14 shows the Gantt Chart for Part 2.

Table 3.5: Project Schedule for FYP Part 2

Milestone	Activities	Duration (weeks)
M5	A8 Software model testing and optimization	8
M6	A9 Obtain the hardware components	3
M6	A10 Build the hardware prototype	2
M6	A11 Integrate the software with hardware	4
M6	A12 On-site testing and results collection	5
-	A13 Documentation and final report writing	3

Activities	Weeks													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A8	■	■	■	■	■	■	■	■						
A9		■	■	■										
A10					■	■								
A11						■	■	■	■					
A12							■	■	■	■	■	■		
A13												■	■	■

Figure 3.14: Gantt Chart for FYP Part 2

### 3.7 Summary

This chapter has elaborated the details of methodology involved in the project such as the system configuration, execution procedures, flowchart, software, hardware components and cost of materials. Besides, the workplan, milestones, activities and their respective schedule and duration were listed out. The author carried out the project based on the above information to ensure that it could be successfully completed and all the objectives were able to be achieved.



## CHAPTER 4

### RESULTS AND DISCUSSION

#### 4.1 Introduction

This chapter mainly illustrates and explains the outcomes of each sub-system as mentioned previously in the methodology section. In-depth analysis have been performed on the results, in terms of software, hardware and accuracy evaluation, to verify and ensure that the solutions provided and system developed are able to achieve the aim and objectives of this project.

#### 4.2 Software Simulation

##### 4.2.1 Deep Learning Model

The deep learning model was based on the SSD MobileNet V2 achitecture. The size of the model, with training steps of 15000, was about 10 MB. It was appropriate for resource-limited applications due to small sized.

Figure 4.1 illustrates the on-going training process of the deep learning model on Google Colab with TensorFlow. It can be observed that the time taken for each training step was about 0.75 seconds as accelerated by the GPU. Besides, the loss parameters were updated for every 100 training steps.

Moreover, Figure 4.2 shows the graphs of various loss functions for the training process of the deep learning model. It can be observed that the curve of the total loss function was decreasing and approaching constant as the training steps increased. Thus, the training process had been stopped to avoid overfitting.

```

+ Code + Text
Notebook
Loss/total_loss: 0.39922482,
'learning_rate': 0.7340352}
INFO:tensorflow:Step 11000 per-step time 0.737s
10920 02:30:18.990163 140486802888576 model_lib_v2.py:700] Step 11000 per-step time 0.737s
INFO:tensorflow:{"loss/classification_loss": 0.16098078,
'loss/localization_loss': 0.099271215,
'loss/regularization_loss': 0.13897282,
'loss/total_loss': 0.39922482,
'learning_rate': 0.7325879}
10920 02:30:18.990575 140486802888576 model_lib_v2.py:701] {"loss/classification_loss": 0.16098078,
'loss/localization_loss': 0.099271215,
'loss/regularization_loss': 0.13897282,
'loss/total_loss': 0.39922482,
'learning_rate': 0.7325879}
INFO:tensorflow:Step 11100 per-step time 0.745s
10920 02:31:33.514796 140486802888576 model_lib_v2.py:700] Step 11100 per-step time 0.745s
INFO:tensorflow:{"loss/classification_loss": 0.14676838,
'loss/localization_loss': 0.08512042,
'loss/regularization_loss': 0.1396063,
'loss/total_loss': 0.3714951,
'learning_rate': 0.73112625}
10920 02:31:33.515183 140486802888576 model_lib_v2.py:701] {"loss/classification_loss": 0.14676838,
'loss/localization_loss': 0.08512042,
'loss/regularization_loss': 0.1396063,
'loss/total_loss': 0.3714951,
'learning_rate': 0.73112625}
Traceback (most recent call last):
File "model_main_tf2.py", line 121, in <module>
tf.compat.v1.app.run()
File "/usr/local/lib/python3.7/dist-packages/tensorflow/python/platform/app.py", line 40, in run
_run(main=main, argv=argv, flags_parser=_parse_flags_tolerate_undef)
File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 303, in run
_run_main(main, args)
File "/usr/local/lib/python3.7/dist-packages/absl/app.py", line 251, in _run_main
sys.exit(main(argv))

```

Figure 4.1: On-going Training Process of Deep Learning Model on Google Colab

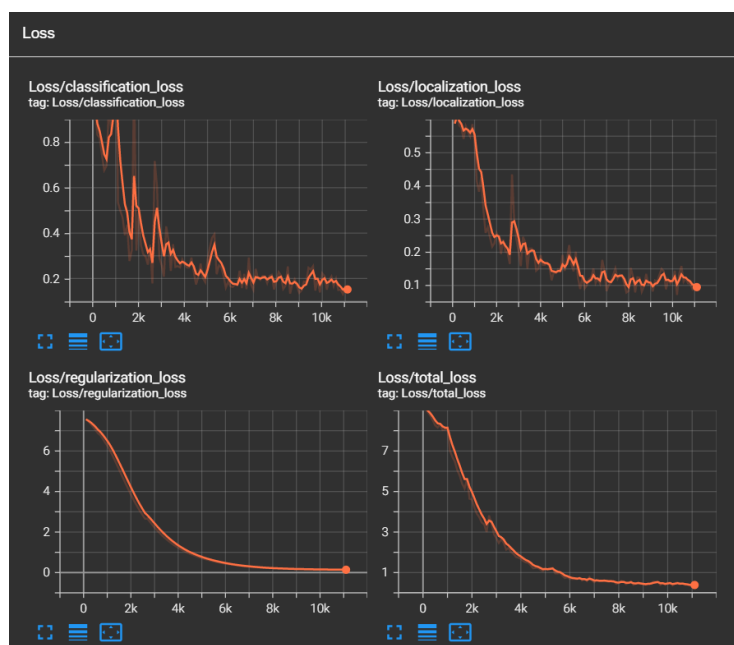


Figure 4.2: Graphs of various Loss Functions

Both the deep learning model and meter image were loaded into the system memory. The meter image was then being inferred and drawn with detection box using the `viz_utils.visualize_boxes_and_labels_on_image_array` function as shown in Figure 4.3. Figure 4.4 shows the inferecing of various

types of meters by the deep learning model. It can be observed that the meter's readings area had been framed with detection box and labelled as readings.

```
viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_detections,
    detections['detection_boxes'],
    detections['detection_classes'],
    detections['detection_scores'],
    category_index,
    skip_scores=True,
    use_normalized_coordinates=True,
    max_boxes_to_draw=1,
    min_score_thresh=0.2,
    agnostic_mode=False)
```

Figure 4.3: Detection Box Visualizing Function



Figure 4.4: Inference of various Types of Meters

#### 4.2.2 Extraction of ROI

The Region of Interest (ROI), containing the meter's reading, was extracted by removing the area outside the detection box and retaining the area within the box so that any unintended characters would not be recognized during the OCR. This process was achieved using the contour detection and masking methods.

Figure 4.5 shows the meter image that had been inferred using the deep learning model where the "readings" labelling was hidden with the *skip\_labels* parameter in the *visualization\_utils* function as it was unnecessary for the ROI extraction.



Figure 4.5: Inferred Meter Image with Label Hidden

Figure 4.6 shows the output of the *inRange* function with lower and upper limits of HSV values of (40,200,200) and (50,255,255) respectively. It can be observed that the outline of the detection box is extracted in white (bits of 255) whereas the other region became black (bits of 0).

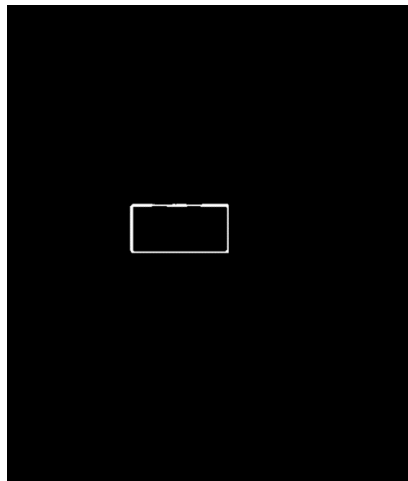


Figure 4.6: Output of *inRange* function

Figure 4.7 shows the mask created using the *findContours* and *drawContours* functions. Firstly, the *findContours* detects the outline extracted. Next, the *drawContours* makes the area within the box to be white. The mask is essential for the extraction of ROI which will be implemented with the *bitwise AND* operation.

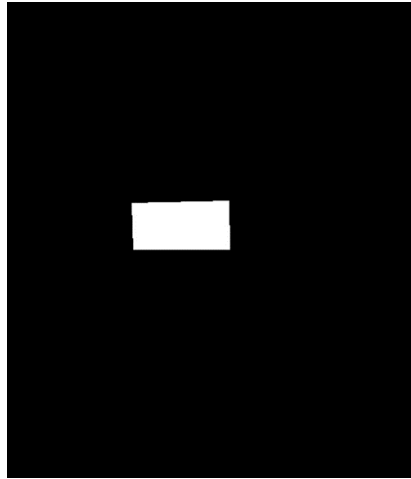


Figure 4.7: Mask for ROI Extraction

Figure 4.8 shows the ROI containing the meter's reading which is obtained by applying the *bitwise AND* function on the mask as well as the meter image. Figure 4.9 shows the excessive region in black was cropped using the positions of vertices of the ROI, obtained from the NumPy min and max functions.



Figure 4.8: Output of *bitwise AND* Function

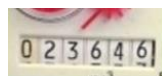


Figure 4.9: ROI Extracted

### 4.2.3 Image Processing

The ROI extracted had undergone image processing to remove the noise and enhance its quality. This step was essential as to improve the accuracy of the optical character recognition.

Firstly, the ROI was enlarged using the *resize* function of the OpenCV with the *INTER\_CUBIC* interpolation method as shown in Figure 4.10. It can be observed that the quality of the image, in terms of clearness, can still be retained after enlarging.

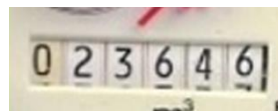


Figure 4.10: ROI being Enlarged

Next, grey-scaling was performed on the ROI using the colour space conversion function. This step was necessary as the thresholding function of OpenCV in the next step can only take in a single-channel (Grey) image instead of three-channels (RGB) image. Figure 4.11 shows the ROI in grey-scale.



Figure 4.11: Grey-scaled ROI

Afterwards was the thresholding operation. Figure 4.12 illustrates the thresholding of the ROI in which the image being represented in binary form, which was black (0) and white (max). It can be observed that some of the noises and unnecessary components are removed.

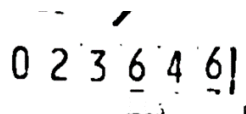


Figure 4.12: Thresholded ROI

#### 4.2.4 Character Recognition

Finally, the ROI after being processed, was performed with optical character recognition using the Tesseract OCR engine. The numerical characters in the ROI are digitized and converted to string datatype as shown in Figure 4.13. Further operations such as uploading to the cloud can be performed using this string data.

Name	Type	Size	Value
recognized	str	1	023646

Figure 4.13: Output of Tesseract OCR

### 4.3 Hardware Implementation

#### 4.3.1 Setup

The Raspberri Pi microcomputer was setup with its operating system, namely Raspberry Pi OS, as well as all the essential libraries as mentioned in the methodology section. Next, the camera module was intalled on the CSI peripheral and being activated.

Subsequently, the microcomputer with camera module was fixed on a plastic holder with clipping function as shown in Figure 4.14. Moreover, the power of the hardware was supplied by a power bank to enable the its portability as illustrated in Figure 4.15. The image recognition software developed in previous steps is transferred to the Raspberry Pi microcomputer for execution and validation.

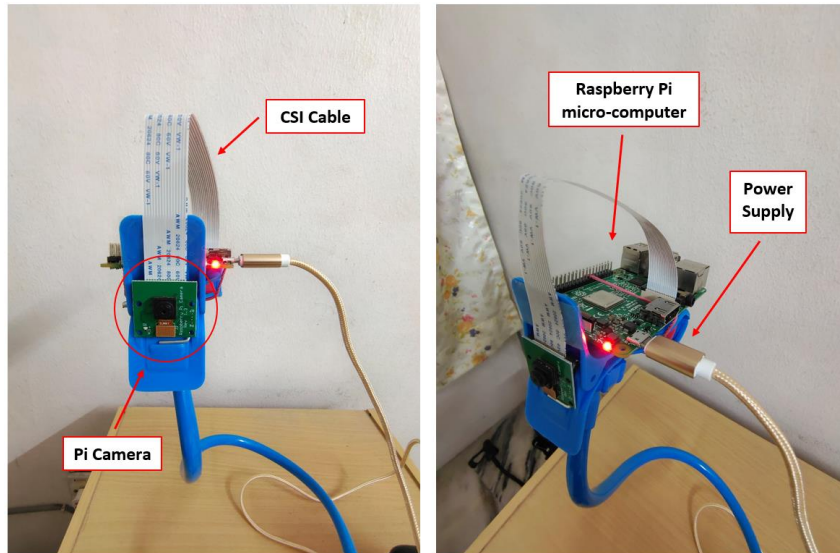


Figure 4.14: Hardware Setup

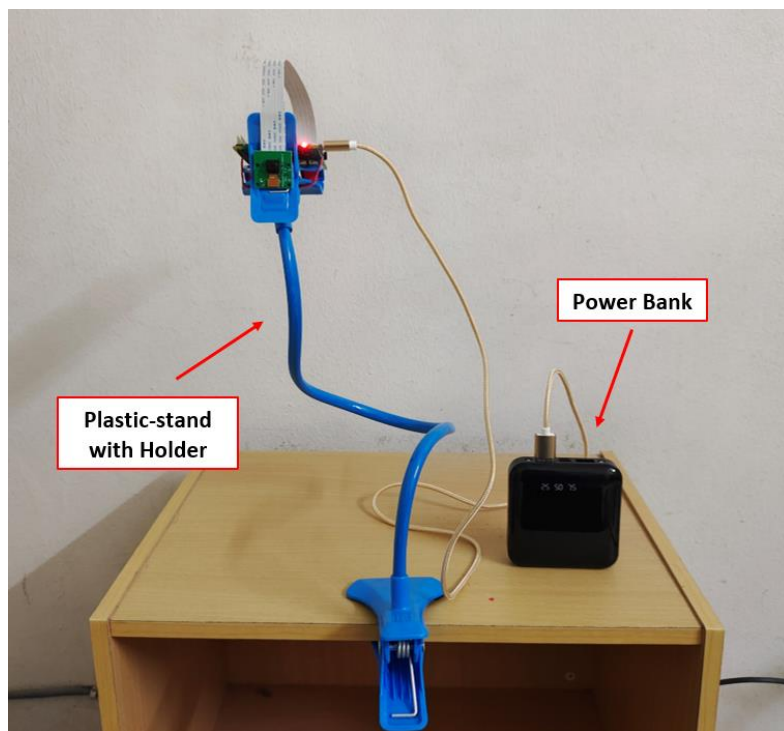


Figure 4.15: Hardware Connected to Power Bank



### 4.3.2 Installation on Meter

After the Raspberry Pi microcomputer had been set up and the image recognition software was successfully executed on the microcomputer, the hardware was then installed on the water meter for practicality testing and real-time monitoring. The hardware installation on the water meter is shown in Figure 4.16.



Figure 4.16: Hardware Installation on Water Meter

#### 4.4 Cloud and Real-time Database

The digitized meter's reading was uploaded to the Firebase Real-time Database to enable the cloud storing and real-time monitoring functions. The interval of data upload had been configured to two minutes; thus, the data in the Firebase would be updated for every two minutes.

The real-time database can be accessed using internet browser, by logging in to the Firebase's website with URL of *console.firebase.google.com*. Figure 4.17 shows the data being collected and stored in the Firebase Real-time Database. It can be observed that each data consists of information in date, time and reading digitized.

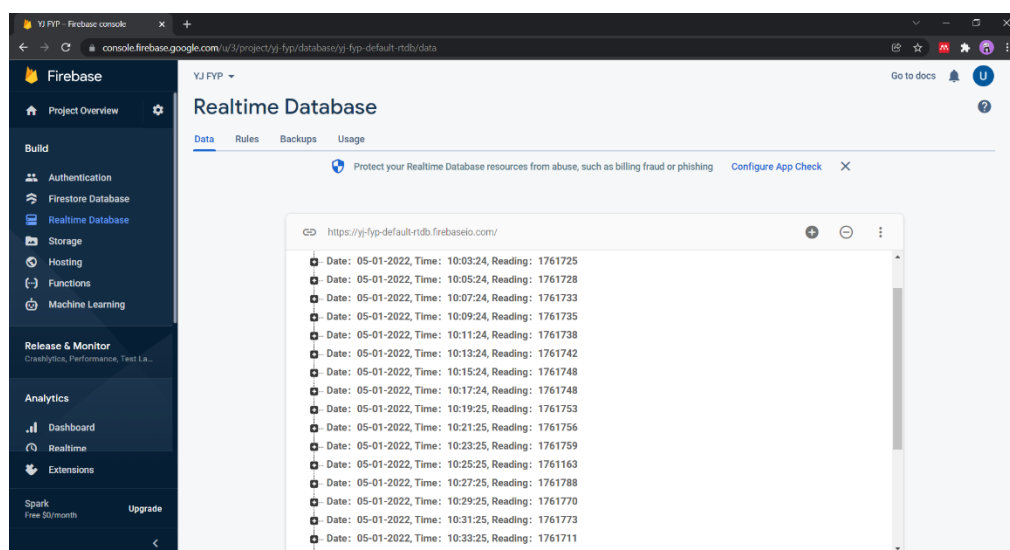


Figure 4.17: Data Collected in Firebase

#### 4.5 Accuracy Evaluation

The accuracy of the system was evaluated in terms of two criteria, namely the accuracy of meter's reading region localization by the deep learning model and the average accuracy of optical character recognition on the meter's reading. Figure 4.18 shows the data being collected, tabulated and compared using Microsoft Excel. In addition, Equation 4.1 was used to calculate the accuracy of the optical character recognition. Besides, the overall results of the accuracy evaluation are shown in Table 4.1.

	A	B	C	D	E	F	G
1		Aspect 1: DLM	Accuracy	Aspect 2: OCR		Accuracy	
2		Detected		Actual	Detected		
3		Yes	1	1761725	1761725	100	
4		Yes	1	1761728	1761728	100	
5		Yes	1	1761732	1761733	85.7	
6		Yes	1	1761735	1761735	100	
7		Yes	1	1761739	1761738	85.7	
8		Yes	1	1761742	1761742	100	
9		Yes	1	1761746	1761748	85.7	
10		Yes	1	1761749	1761748	85.7	
11		Yes	1	1761753	1761753	100	
12		Yes	1	1761756	1761756	100	
13		Yes	1	1761759	1761759	100	
14		Yes	1	1761763	1761163	85.7	
15		Yes	1	1761166	1761788	57.1	
16		Yes	1	1761770	1761770	100	
17		Yes	1	1761773	1761773	100	
18		Yes	1	1761777	1761711	71.4	
19		Yes	1	1761780	1761780	100	
20		Yes	1	1761783	1761783	100	
21		Yes	1	1761787	1761781	85.7	
22		Yes	1	1761791	1761791	100	
23		Yes	1	1761794	1761794	100	
24		Yes	1	1761797	1767797	85.7	
25		Yes	1	1761801	1761601	85.7	
26		Yes	1	1761804	1761804	100	
27		Yes	1	1761808	1761808	100	
28		Yes	1	1761811	1767817	71.4	
29		Yes	1	1761814	1761814	100	

Figure 4.18: Data Tabulated using Microsoft Excel

$$\% \text{ Accuracy} = \frac{\text{Num of Correct Digits}}{\text{Total Num of Digits}} \times 100 \quad (4.1)$$

Table 4.1: Results of Accuracy Evaluation

Aspects	Values
Accuracy of meter's readings region detection by Deep Learning Model	95%
Average accuracy of character recognition by Tesseract OCR	91%

From the evaluation process, it is noticed that the accuracy of the meter's readings region detection by deep learning model is affected by the size and resolution of the meter images. When the meter faces are small, which is occupying less than 30% of the whole image, there is higher chance that the deep learning model detects the false area. Thus, it is recommended to place the camera near and centred to the meter face.

Besides, the Tesseract OCR engine is sensitive to the skewing of the characters. Hence, it is suggested that during the capturing process of the image,

the camera should be placed correctly with respect to the meter-face orientation to optimise the performance of the Tesseract.

Moreover, the sharpness and quality of the image taken are affected by the vibration caused by wind or movement of vehicles. When the vibration occurs at the camera module, the photo capture is blurry and thus, the performance and accuracy of the optical character recognition are greatly reduced. In order to mitigate this issue, the camera module is suggested to be installed on a stable and enduring platform.

#### **4.6 Summary**

The results obtained from each sub-module of the software, as well as the hardware and accuracy evaluation have been presented in this chapter. In-depth analysis had been performed and detailed explanations were provided along with each result to ensure that the information on operating principles and functions of each sub-system are clearly delivered.

## CHAPTER 5

### CONCLUSION AND RECOMMENDATIONS

#### 5.1 Conclusion

Digitization is one of the major components in IR 4.0. It provides a lot of benefits to the industrial, such as increasing productivity, better data visualisation, simplifying parameters control, etc. One of the important steps in digital transformation is to make all the instrumental devices connected to the cloud. However, replacing the existing analogue meters with the cloud-connected digital meters can be very costly especially for industrial grade meters.

The aim of this research is to develop a cost-effective image recognition system to capture and monitor the analogue meter's readings and send the data to the cloud system of the factory as a part of digitization and automation. The objectives of this project, namely to investigate the feasibility of image recognition system for analogue meter's reading detection, to develop the image recognition system and to evaluate the functions and performance of the system have been achieved accordingly.

The system developed consists of a deep learning model based on SSD MobileNet V2 and optical character recognition engine referring to the Tesseract. The deep learning model was trained with a dataset of 750 meters' images, and it is used to detect the region of interest where the meter's readings are located. The OCR is used to convert the readings to string datatype. Besides, the image processing techniques via the OpenCV library are implemented for enhancing the quality of the ROI.

The programme developed is executed on Raspberry Pi microcomputer with camera module installed and its performance has been evaluated. The results show that the deep learning model and OCR accuracies are 95% and 91%, respectively. Moreover, the data were successfully uploaded to the cloud-based service platform, namely Firebase.

## **5.2 Recommendations for future work**

The limitation of this project is on the diversity of the meters being recognized. Due to the constraint in various parameters involving the image processing, computer vision and character recognition, the system developed is only applicable to detect and convert the numerical reading (digits). Thus, the image recognition system is not universal. In future, the deep learning model can be further trained to recognize and digitize the analogue meter with arrows or pointers so that the system is applicable to more variety of meters.

Besides, the current hardware system uses WiFi connection for internet access and uploading the data to cloud. Due to the absence of WiFi signal at certain places, the cloud access will fail. Thus, mobile connectivity such as 3G or 4G technology can be added to the system to address the problem.

## REFERENCES

Asthana, S.R., 2014. Enhanced Camera Calibration for Machine Vision using OpenCV. *IAES International Journal of Artificial Intelligence (IJ-AI)*, 3(3), p.136

Caylar, P. and Noterdaeme, O., 2016. Digital in industry: From buzzword to value creation. (Exhibit 1), pp.1–9.

Copeland, B., 2020. *Artificial intelligence*. [Online]  
Available at: <<https://www.britannica.com/technology/artificial-intelligence>>  
[Accessed 10 July 2021].

Gong, X.Y., Su, H., Xu, D., Zhang, Z.T., Shen, F. and Yang, H. Bin, 2018. An Overview of Contour Detection Approaches. *International Journal of Automation and Computing*, 15(6), pp.656–672.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications..

IBM Cloud Education, 2020. *Convolutional Neural Networks*. [Online]  
Available at: <<https://www.ibm.com/cloud/learn/convolutional-neural-networks>> [Accessed 17 Jan 2022].

IBM Cloud Education, 2020. *Supervised Learning*. [Online]  
Available at: <<https://www.ibm.com/cloud/learn/supervised-learning>>  
[Accessed 10 Jan 2022].

Idris, R., 2019. Industrial Revolution 4.0: An Overview of Readiness and Potential Economic Effects in Malaysia from Millennial’s Perspective. *World Scientific News*, 118(January), pp.273–280.

Johnson, D., 2022. *Back Propagation Neural Network: What is Backpropagation Algorithm in Machine Learning?*. [Online]  
Available at: <<https://www.guru99.com/backpropagation-neural-network.html>>  
[Accessed 20 Jan 2022].

Kumar, R., 2021. *Supervised, Unsupervised, And Semi-Supervised Learning*. [Online]  
Available at: <<https://medium.com/enjoy-algorithm/supervised-unsupervised-and-semi-supervised-learning>>  
[Accessed 10 Jan 2022].

Mallick, S., n.d. *Contour Detection using OpenCV*. [Online]  
Available at: <<https://learnopencv.com/contour-detection-using-opencv-python-c/>> [Accessed 19 July 2021].

Mallick, S., n.d. *Image Rotation and Translation Using OpenCV*. [Online]  
Available at: <<https://learnopencv.com/image-rotation-and-translation-using-opencv/>> [Accessed 20 July 2021].

MathWorks, n.d. Convolutional Neural Network. [Online]  
Available at: <<https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>>  
[Accessed 17 Jan 2022].

MathWorks, n.d. *Image Recognition*. [Online]  
Available at: <<https://www.mathworks.com/discovery/image-recognition-matlab.html>> [Accessed 7 July 2021].

McGonagle, J. et al., n.d. *Backpropagation*. [Online]  
Available at: <<https://brilliant.org/wiki/backpropagation/>>  
[Accessed 20 Jan 2022].

Mohamad, E., Sukarma, L., Mohamad, N.A., Salleh, M.R., Rahman, M.A.A., Rahman, A.A.A. and Sulaiman, M.A., 2018. Review on Implementation of Industry 4.0 Globally and Preparing Malaysia for Fourth Industrial Revolution. *The Proceedings of Design & Systems Conference*, 2018.28(0), p.2203.

Moroney, L., 2017. The firebase realtime database. In *The Definitive Guide to Firebase* (pp. 51-71). Apress, Berkeley, CA.

Mujtaba, H., 2020. *What is Image Recognition and How it is Used?*. [Online]  
Available at: <<https://www.mygreatlearning.com/blog/image-recognition/>>  
[Accessed 17 July 2021].

Nelson, M.J. and Hoover, A.K., 2020. Notes on Using Google Colaboratory in AI Education. *Annual Conference on Innovation and Technology in Computer Science Education, ITiCSE*, pp.533–534.

Numpy Organisation, 2021. *What is NumPy?*. [Online]  
Available at: <<https://numpy.org/doc/stable/user/whatisnumpy.html>>  
[Accessed 2 August 2021].

OpenCV Organisation, 2020. *Color conversions*. [Online]  
Available at: <[https://docs.opencv.org/4.4.0/de/d25/imgproc\\_color\\_conversions.html](https://docs.opencv.org/4.4.0/de/d25/imgproc_color_conversions.html)> [Accessed 15 July 2021].

OpenCV Organisation, n.d. *About*. [Online]  
Available at: <<https://opencv.org/about/>> [Accessed 14 July 2021].

OpenCV Organisation, n.d. *Image Thresholding*. [Online]  
Available at: <[https://docs.opencv.org/4.5.2/d7/d4d/tutorial\\_py\\_thresholding.html](https://docs.opencv.org/4.5.2/d7/d4d/tutorial_py_thresholding.html)> [Accessed 16 July 2021].



Oppermann, A., 2019. *What is Deep Learning and How does it work?*. [Online] Available at: <<https://towardsdatascience.com/what-is-deep-learning-and-how-does-it-work>> [Accessed 10 Jan 2022].

Oracle, n.d. *What is IoT?*. [Online] Available at: <<https://www.oracle.com/internet-of-things/what-is-iot/>> [Accessed 9 July 2021].

Patel, C., Patel, A. and Patel, D., 2012. Optical Character Recognition by Open source OCR Tool Tesseract: A Case Study. *International Journal of Computer Applications*, 55(10), pp.50–56.

RaspberryPi, 2020. Raspberry Pi 4. *Raspberry Pi Foundation*, (May).

Rogowska, J., 2009. Overview and Fundamentals of Medical Image Segmentation. In: I. N. Bankman, ed. *Handbook of Medical Image Processing and Analysis*. s.l.:Academic Press.

Rune, 2020. *Understand How Color to Gray Scale Works Using OpenCV*. [Online] Available at: <<https://www.learnpythonwithrune.org/understand-how-color-to-gray-scale-works-using-opencv/>> [Accessed 15 July 2021].

Rymarczyk, J., 2020. Technologies, opportunities and challenges of the industrial revolution 4.0: Theoretical considerations. *Entrepreneurial Business and Economics Review*, 8(1), pp.185–198.

Saha, S., 2018. *A Comprehensive Guide to Convolutional Neural Networks*. [Online] Available at: <<https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53>> [Accessed 20 Jan 2022].

Sajjad, K.M., 2012. Automatic License Plate Recognition using Python and OpenCV. *Department of Computer Science and Engineering MES College of Engineering.*, pp.1–5.

Sreedhar, K., 2012. Enhancement of Images Using Morphological Transformations. *International Journal of Computer Science and Information Technology*, 4(1), pp.33–50.

Stanford University, n.d. *Image Processing*. [Online] Available at: <<https://ai.stanford.edu/~syeyeung/cvweb/tutorial1.html>> [Accessed 17 July 2021].

Tableau, n.d. *Big Data Analytics: What It Is, How It Works, Benefits, And Challenges*. [Online] Available at: <<https://www.tableau.com/learn/articles/big-data-analytics>> [Accessed 9 July 2021].

Taud, H. and Mas, J.F., 2018. Multilayer perceptron (MLP). In *Geomatic approaches for modeling land change scenarios* (pp. 451-455). Springer, Cham.

Teli, M. N., 2019. Canny Edge Detection. *University of Maryland*.

Tesseract, n.d. *Improving the quality of the output*. [Online] Available at: <<https://tesseract-ocr.github.io/tessdoc/ImproveQuality>> [Accessed 9 Jan 2022]

Tesseract, 2017. *Traineddata Files for Version 4.00 +*. [Online] Available at: <<https://tesseract-ocr.github.io/tessdoc/Data-Files.html>> [Accessed 9 Jan 2022]

Torrey, L. and Shavlik, J., n.d. Transfer learning. *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques* (pp. 242-264). IGI global.

Tzutalin, 2015. LabelImg. Git code. Available at: <<https://github.com/tzutalin/labelImg>>

University of Maine, n.d. *CMOS Sensor*. [Online] Available at: <[http://www.optique-ingenieur.org/en/courses/OPI\\_ang\\_M05\\_C06/co/Contenu\\_20.html](http://www.optique-ingenieur.org/en/courses/OPI_ang_M05_C06/co/Contenu_20.html)> [Accessed 15 July 2021].

Utmel Electronic, 2020. *Image Sensor: How do CCD and CMOS Sensors work?*. [Online] Available at: <<https://www.utmel.com/blog/categories/sensors/image-sensor-how-do-ccd-and-cmos-sensors-work>> [Accessed 15 July 2021].

Zhou, K., Liu, T. and Zhou, L., 2016. Industry 4.0: Towards future industrial opportunities and challenges. *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery, FSKD 2015*, pp.2147–2152.

## APPENDICES

### APPENDIX A: Acceptance of Conference Paper

#### Name of Conference:

12<sup>th</sup> IEEE Symposium on Computer Applications & Industrial Electronics  
(ISCAIE 2022)

#### Date of Conference:

22 May 2022

#### Title of Conference Paper:

Deep Learning and Optical Character Recognition for Digitization of Meter  
Reading

#### Notification of Acceptance:

[ISCAIE2022] Your submission has been accepted! External Inbox x



ISCAIE2022 <IEEE.iscaie@gmail.com>  
to me

Thu, Mar 24, 3:54 PM ★ ↶ ⋮

\*Please ignore this email if you have completed your registration and payment\*

Dear Yue Jiet,

Congratulations!

On behalf of the 2022 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE 2022), I am pleased to inform you that your submission titled: "Deep Learning and Optical Character Recognition for Digitization of Meter Reading" has been ACCEPTED for Virtual Presentation.

#### Reviewers' Feedback/Comment

We have included the reviewers' feedback at the end of this message. Please read comments from reviewers and make the necessary corrections and amendments before the final camera-ready submission. Please address these comments while preparing your camera-ready version. Your paper may still be rejected if this is not followed.

#### Registration and Payment:

Please complete the registration and payment latest by 15th April 2022 at this registration link:  
[https://www.ihanyassin.com/iscaie2022\\_reg/](https://www.ihanyassin.com/iscaie2022_reg/)

#### IEEE No Show Policy

Please take note that IEEE has a strict policy on "NO-SHOW" status. The IEEE has implemented a policy to exclude authors who do not present their accepted submission from further publication distribution, such as exclusion from IEEE Xplore. Therefore, one of the authors or their representatives MUST submit the video presentation and upload it. Please take note of IEEE Terms & Conditions here:  
<https://sites.google.com/view/iscaie/ieee-no-show-policy?authuser=0>

#### Video Presentation

Please upload your presentation video (max of 15 minutes) in the YouTube latest by 20th April 2022 as either Unlisted or Public.

Once again, Congratulations and Stay Safe. Take Care.

Regards,  
Technical Program Chair  
ISCAIE 2022  
[IEEE.iscaie@gmail.com](mailto:IEEE.iscaie@gmail.com)

## APPENDIX B: Code of Deep Learning Model Training (Google Colab)

```
%cd /content/drive/MyDrive/Colab Notebooks/models/research

!protoc object_detection/protos/*.proto --python_out=.

!export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim

!apt-get install -qq protobuf-compiler python-pil python-
lxml python-tk

!pip install -
qq Cython contextlib2 pillow lxml matplotlib pycocotools

cd /content/drive/MyDrive/Colab Notebooks/models

pip install --user -r official/requirements.txt

pip install tf_slim

pip install dataclasses

pip install tensorflow-addons

pip install tensorflow-text-nightly

pip install utils

pip install lvis

import os
os.environ['PYTHONPATH']+="/content/drive/MyDrive/Colab Not
ebooks/models"
os.environ['PYTHONPATH']+="/content/drive/MyDrive/Colab Not
ebooks/models/research"

!python generate_tfrecord.py --
csv_input=data/test_labels.csv --
output_path=tf_record/test.record --image_dir=test_images

!python model_main_tf2.py \
  --model_dir=trained-checkpoint --num_train_steps=50000 \
  --sample_1_of_n_eval_examples=1 \
  --pipeline_config_path=pipeline.config \
  --alsologtostderr

!python exporter_main_v2.py \
```

```
--input_type image_tensor \  
--pipeline_config_path pipeline.config \  
--trained_checkpoint_dir trained-checkpoint \  
--output_directory exported-model
```

### APPENDIX C: Code of TensorBoard

```
cd /content/drive/MyDrive/Colab Notebooks/models  
  
import tensorflow as tf  
import datetime, os  
  
%load_ext tensorboard  
  
%tensorboard --logdir trained-checkpoint
```

## APPENDIX D: Code of Image Recognition Software Developed

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'    # Suppress TensorFlow logging (1)
import pathlib
import tensorflow as tf
import cv2
import numpy
import numpy as np
from PIL import Image
tf.get_logger().setLevel('ERROR')         # Suppress TensorFlow logging (2)
import re
import pytesseract
import imutils
from firebase import firebase
from datetime import datetime
from time import sleep
firebase = firebase.FirebaseApplication('https://yj-fyp-default-rtdb.firebaseio.com/', None)

# Enable GPU dynamic memory allocation
gpus = tf.config.experimental.list_physical_devices('GPU')
for gpu in gpus:
    tf.config.experimental.set_memory_growth(gpu, True)

pytesseract.pytesseract.tesseract_cmd = r'D:\Program Files\Tesseract\tesseract.exe'
```

```
PATH_TO_LABELS = r'D:\User Files\Desktop\readings\label_map.pbtxt'

import time
from object_detection.utils import label_map_util
from object_detection.utils import visualization_utils as viz_utils

PATH_TO_SAVED_MODEL = r'D:\User Files\Desktop\readings\My model\exported-model_ver1\saved_model'

print('Loading model...', end='')
start_time = time.time()

# Load saved model and build the detection function
detect_fn = tf.saved_model.load(PATH_TO_SAVED_MODEL)

end_time = time.time()
elapsed_time = end_time - start_time
print('Done! Took {} seconds'.format(elapsed_time))

category_index = label_map_util.create_category_index_from_labelmap(PATH_TO_LABELS, use_display_name=True)

import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore') # Suppress Matplotlib warnings

def load_image_into_numpy_array(path):
```

```
    return np.array(Image.open(path))

for i in range(100):
    image_path = r'D:\User Files\Desktop\{}.jpg'.format(i+1)

    print('Running inference for {}... '.format(image_path), end='')

    image_np = load_image_into_numpy_array(image_path)

    # Condition based:
    # Flip horizontally
    # image_np = np.fliplr(image_np).copy()

    # Convert image to grayscale
    # image_np = np.tile(
    #     np.mean(image_np, 2, keepdims=True), (1, 1, 3)).astype(np.uint8)

    # The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
    input_tensor = tf.convert_to_tensor(image_np)
    # The model expects a batch of images, so add an axis with `tf.newaxis`.
    input_tensor = input_tensor[tf.newaxis, ...]

    # input_tensor = np.expand_dims(image_np, 0)
    detections = detect_fn(input_tensor)
```



```
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
              for key, value in detections.items()}
detections['num_detections'] = num_detections

detections['detection_classes'] = detections['detection_classes'].astype(np.int64)

image_np_with_detections = image_np.copy()

viz_utils.visualize_boxes_and_labels_on_image_array(
    image_np_with_detections,
    detections['detection_boxes'],
    detections['detection_classes'],
    detections['detection_scores'],
    category_index,
    skip_scores=True,
    skip_labels=True,
    use_normalized_coordinates=True,
    max_boxes_to_draw=1,
    min_score_thresh=0.2,
    agnostic_mode=False)

print('Done')
print(detections['detection_classes'])
print(detections['detection_scores'])
im_PIL = Image.fromarray(image_np_with_detections)
```

```
im_cv = cv2.cvtColor(numpy.array(im_PIL), cv2.COLOR_RGB2BGR)

inferenced = im_cv
inferenced_copy = inferenced.copy()

inferenced_HSV = cv2.cvtColor(inferenced, cv2.COLOR_BGR2HSV)
lower_green = np.array([40,200,200])
upper_green = np.array([50,255,255])
green_box_detect = cv2.inRange(inferenced_HSV, lower_green, upper_green)
contours = cv2.findContours(green_box_detect, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
contours = imutils.grab_contours(contours)
contours = sorted(contours, key = cv2.contourArea, reverse = True)[:10]
screenCnt = None
for c in contours:

    peri = cv2.arcLength(c, True)
    approx = cv2.approxPolyDP(c, 0.018 * peri, True)

    if len(approx) >= 4 and len(approx) <=6:
        screenCnt = approx
        break

if screenCnt is None:
    detected = 0
    print ("No contour detected")
else:
```

```

    detected = 1

    mask = np.zeros(green_box_detect.shape,np.uint8)
    mask = cv2.drawContours(mask,[screenCnt],0,255,-1,)
    extracted = cv2.bitwise_and(inferenced,inferenced,mask=mask)
    (x, y) = np.where(mask == 255)
    (topx, topy) = (np.min(x), np.min(y))
    (bottomx, bottomy) = (np.max(x), np.max(y))
    readings_ROI = inferenced[topx+5:bottomx-5, topy:bottomy]

    scale_percent = 300 # percent of original size
    width = int(readings_ROI.shape[1] * scale_percent / 100)
    height = int(readings_ROI.shape[0] * scale_percent / 100)
    dim = (width, height)
    readings_ROI_resized = cv2.resize(readings_ROI, dim, interpolation = cv2.INTER_CUBIC)
    readings_ROI_resized_gray = cv2.cvtColor(readings_ROI_resized, cv2.COLOR_BGR2GRAY)

    ret, readings_ROI_resized_thresh = cv2.threshold(readings_ROI_resized_gray,128,255, cv2.THRESH_BINARY)
    ROI_PIL = Image.fromarray(readings_ROI_resized_thresh)
    text = pytesseract.image_to_string(ROI_PIL,lang = 'eng',config = '--psm 7 outputbase digits')
    converted = re.sub('[^0-9]','', text)

    now_time = (datetime.now()).strftime("%d-%m-%Y, %H:%M:%S")
    data = {'Reading': converted}
    name = "Date: {}, Time: {}, Reading: {}".format(now_time[0:10], now_time[12:], converted)
    result = firebase.patch(name,data)

```

```
sleep(5)
```