

SMART SECURITY CAMERA WITH DEEP LEARNING

WONG YEE CHENG


**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 2020

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 : WONG YEE CHENG

ID No. : 15UEB05621

Date : 26th April 2020

APPROVAL FOR SUBMISSION

I certify that this project report entitled “**SECURITY CAMERA WITH DEEP LEARNING**” was prepared by **WONG YEE CHENG** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Engineering (Honours) Electrical and Electronic Engineering at Universiti Tunku Abdul Rahman.

Approved by,

Signature

:



Supervisor

:

DR. THAM MAU LUEN

Date

:

26th April 2020

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.

© 2020, Wong Yee Cheng. All right reserved.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Tham Mau Luen for his invaluable advice, guidance and his enormous patience throughout the development of the research. In every phase of the project, his supervision and guidance shaped this project to be completed perfectly.

In addition, I would also like to express my gratitude to my loving parents and friends who had helped and given me full support in this project. I appreciate the encouragement given by my supervisor, loving parents, and friends.

ABSTRACT

Recent statistics indicate that home burglary remains to be one of the most common property crimes. Video surveillance is one of the mainstream crime prevention measures used around the globe. Driven by the success of machine learning, this study aims to develop a real-time smart security camera system. Different from existing works, the designed deep learning model is deployed on a resource limited Raspberry Pi 3 B+, inference of which is handled by an accelerator called Intel Movidius Neural Compute Stick 2 (NCS2). Darknet-53 of YOLO v3 is selected as the feature extractor and it undergoes transfer learning and fine tuning, in order to detect three types of weapons, namely, gun, helmet, and knife. Furthermore, data augmentation is applied to overcome the scarcity of datasets, which is collected and labelled via LabelImg. The training platform is Google Colab whereas the testing environment using NCS2 requires the pre-trained model to be converted into TensorFlow and subsequently intermediate representation (IR). The performance of crime detection is evaluated in terms of precision, recall, mean average precision (mAP) and frame per second (FPS). Experimental results confirm the effectiveness of using NCS2 in an embedded platform.

TABLE OF CONTENTS

DECLARATION		i
APPROVAL FOR SUBMISSION		ii
ACKNOWLEDGEMENTS		iv
ABSTRACT		v
TABLE OF CONTENTS		vi
LIST OF TABLES		viii
LIST OF FIGURES		ix
LIST OF SYMBOLS / ABBREVIATIONS		xii
LIST OF APPENDICES		xiv
 CHAPTER		
1	INTRODUCTION	1
1.1	General Introduction	1
1.1.1	Artificial Intelligence (AI)	1
1.1.2	Introduction of Deep Learning (DL)	2
1.1.3	Application of Deep Learning	6
1.2	Importance of the Study	7
1.3	Problem Statement	9
1.4	Aim and Objectives	9
1.5	Scope and Limitation of the Study	10
2	LITERATURE REVIEW	11
2.1	Introduction	11
2.2	Applications using CNNs	11
2.3	A Searching Process in 3D Space for Human Detection using Camera and Scene Knowledge	16
2.4	Applications using YOLO v3 Algorithm	20
2.5	Summary	24
3	METHODOLOGY AND WORK PLAN	25
3.1	Introduction	25
3.2	Design Requirements	25

3.2.1	Hardware Requirements	25
3.2.2	Software Requirements	27
3.3	Setup of Operating System (OS), Dependencies and Software into a RP3	28
3.4	Model Selection for Training Dataset	29
3.5	Procedures for Training and Testing YOLO v3 Model	31
3.6	Conversion of YOLO v3 model into Intermediate Representation (IR) to load in Inference Engine (IE)	35
3.7	Flowchart of System Operation	37
3.8	Evaluation Performance of the Model	38
3.8.1	Measurement of Precision, Recall and Mean Average Precision	38
3.8.2	Measurement of Object Detection and FPS	40
3.9	Summary	40
4	RESULTS AND DISCUSSION	41
4.1	Introduction	41
4.2	System Overview	41
4.3	Evaluation Precision and Recall of the Model	42
4.4	Evaluation Average Precision (AP), Mean Average Precision (mAP) and Intersection over Union Threshold (IoU) of the Model	46
4.5	Crime Detections	50
4.6	Real-time and Input Video Performance	52
4.7	Summary	52
5	CONCLUSIONS AND FUTURE WORK	53
5.1	Conclusions	53
5.2	Future Work	53
	REFERENCES	54
	APPENDICES	56

LIST OF TABLES

Table 1.1: Crime rate by type of crimes in Malaysia, 2016-2018 (Department of Statistics Malaysia, 2019).	8
Table 2.1: Comparison of different deep learning architecture and their performance (Norouzzadeh et al., 2018).	14
Table 2.2: Comparison of <i>mLeNet</i> and <i>mAlexNet</i> in CNN architecture (Amato et al., 2016).	15
Table 2.3: Experiment Results of a single camera and multi-camera in <i>mLeNet</i> and <i>mAlexNet</i> (Amato et al., 2016).	16
Table 2.4: Failure modes and Colours for 3D search on CAVIAR INRIA datasheet.	19
Table 2.5: mAP and speed of detection for PCB component detection using Different methods (Li et al., 2019).	21
Table 2.6: Comparison of the Performance of the System using Different Methods (Shakil, Rajjak and Kureshi, 2020).	23
Table 2.7: Comparison of Accuracy of System in Different Methods (Shakil, Rajjak and Kureshi, 2020).	23
Table 4.1: Precision and Recall for All Classes and Gun Detection in Different Confidence Scores.	42
Table 4.2: Precision and Recall for Helmet and Knife Detection in Different Confidence Score.	43
Table 4.3: mAP or AP for Overall System, Gun, Helmet and Knife Detection in Different IoU thresholds.	46
Table 4.4: Testing FPS with Different Devices in Input Video and Real-time.	52

LIST OF FIGURES

Figure 1.1: Sample Case of AI in the Automotive Industry.	2
Figure 1.2: A simple Venn diagram illustrates DL is a subset of ML (Goodfellow, Bengio and Courville, 2016).	2
Figure 1.3: Comparison between different parts of AI system with their own different disciplines (Goodfellow, Bengio and Courville, 2016).	3
Figure 1.4: Deep Neural Network (DNN).	4
Figure 1.5: A CNN Sequence to categorize the handwritten digits.	5
Figure 1.6: Pet and Human Detection using AI Security Camera.	6
Figure 1.7: Smart Security Camera System from Athena Security identify guns using AI and Cloud Technology.	7
Figure 1.8: Crime rate per 100,000 population between the states of Malaysia, 2016-2018 (Department of Statistics Malaysia, 2019).	8
Figure 1.9: Crime rate per 100,000 population in Singapore, 2016-2018 (R. Hirschmann, 2020).	8
Figure 2.1: A CNN architecture (Krizhevsky, Sutskever and Hinton, 2017).	11
Figure 2.2: Eight tested images with Top Five Predictions.	12
Figure 2.3: A CNN architecture proceed with an input layer to an output layer (Willi et al., 2019).	13
Figure 2.4: Animals can be identified, counted, and described in camera-trap images successfully using DNN (Norouzzadeh et al., 2018).	14
Figure 2.5: Pedestrian Detection Results in 2D search and 3D search (Li, Wu and Nevatia, 2008).	17
Figure 2.6: An Approach Overview (Li, Wu and Nevatia, 2008).	17
Figure 2.7: A searching process using 3D Search Grid from datasheet (PETS 2007) (Li, Wu and Nevatia, 2008).	18
Figure 2.8: Sample detection results for 2D search and 3D search on datasheet (PETS 2007)(Li, Wu and Nevatia, 2008).	18
Figure 2.9: Sample detection results for 2D search and 3D search on datasheet (CAVIAR INRIA)(Li, Wu and Nevatia, 2008).	19

Figure 2.10: General Flowchart of PCB detection (Li et al., 2019).	20
Figure 2.11: Implementation of System using YOLO v3 pre-trained model (Shakil, Rajjak and Kureshi, 2020).	22
Figure 2.12: Object Detection and Tracking as cars (Shakil, Rajjak and Kureshi, 2020)	22
Figure 3.1: Raspberry Pi 3 Model B+ (RP3)	26
Figure 3.2: Intel Movidius Neural Compute Stick 2 (NCS2)	26
Figure 3.3: Camera Board for RP3 with 5MP	26
Figure 3.4: Micro SD Card with 128GB and 100MB/s	27
Figure 3.5: Model of Darknet-53 (Redmon and Farhadi, 2018).	29
Figure 3.6: Comparison of Backbone (Redmon and Farhadi, 2018).	30
Figure 3.7: COCO Dataset Testing with mAP@0.5 (Redmon and Farhadi, 2018)	31
Figure 3.8: Example of Labelling Image using LabelImg.	32
Figure 3.9: Annotation in Text File Format.	32
Figure 3.10: Data file, <i>weapon.data</i> .	32
Figure 3.11: Command for training the dataset.	33
Figure 3.12: Process Flow Diagram of Training and Testing YOLO v3 model.	34
Figure 3.13: Overview of the structure of OpenVINO in DL (Intel, 2019).	35
Figure 3.14: Conversion of YOLO v3 into IR, then loads into IE.	36
Figure 3.15: Flowchart of System Operation.	37
Figure 4.1: Overall System	41
Figure 4.2: Prototype of RP3 with NCS2 and Pi Camera	42
Figure 4.3: Precision vs Recall Curve for All Classes.	43
Figure 4.4: Precision vs Recall Curve for Gun Detection.	44
Figure 4.5: Precision vs Recall Curve for Helmet Detection.	44

Figure 4.6: Precision vs Recall Curve for Knife Detection.	45
Figure 4.7: Overall Precision vs Recall Curve in Trained Model.	45
Figure 4.8: mAP against IoU Threshold for All Classes.	47
Figure 4.9: AP against IoU Threshold for Gun Detection.	47
Figure 4.10: AP against IoU Threshold for Helmet Detection.	48
Figure 4.11: AP against IoU Threshold for Knife Detection.	48
Figure 4.12: Overall mAP or AP against IoU Threshold in Trained Model.	49
Figure 4.13: IoU calculation (Rosebrock, 2016).	49
Figure 4.14: Gun Detection	50
Figure 4.15: Helmet Detection	51
Figure 4.16: Knife Detection	51

LIST OF SYMBOLS / ABBREVIATIONS

AI	Artificial Intelligence
ANN	Artificial Neural Network
AP	Average Precision
API	Application Programming Interface
CC	Camera CATalogue
CCTV	Closed-Circuit Television
CLI	Command-line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
D53	Darknet-53
DL	Deep Learning
DNN	Deep Neural Network
Drive	Google Drive
EE	Elephant Expedition
FN	False Negative
FP	False Positive
FPGA	Field-programmable Gate Array
FPS	Frame per Second
GC	Google Colab
GPU	Graphic Processing Unit
GUI	Graphic User Interface
IE	Inference Engine
ImgNet	ImageNet
IoU	Intersection over Union
IR	Intermediate Representation
KL	Kuala Lumpur
LRN	Local Response Normalization
<i>mAN</i>	<i>mAlexNet</i>
mAP	Mean Average Precision
ML	Machine Learning
<i>mLN</i>	<i>mLeNet</i>
MO	Model Optimiser

NCS2	Intel Movidius Neural Compute Stick 2
NCSDK2	Software Development Kit for Neural Compute Stick 2
OpenCV	Open Source Computer Vision
OpenVINO	Open Visual Inference and Neural Network Optimization
OS	Operating System
P	Precision
PCB	Printed Circuit Board
R	Recall
RaspSL	Raspbian Stretch Lite
ReLU	Linear Rectification
RP3	Raspberry Pi 3 Model B+
SD	Secure Digital
SDK	Software Development Kit
SS	Snapchat Serengeti
SW	Snapshot Wisconsin
TF	TensorFlow
TP	True Positive
USB	Universal Serial Bus
VMware	VMware Workstation Player
VNC	Virtual Network Computing
VPU	Visual Processing Unit
VPU	Visual Processing Unit
Win32	Win32 Disk Imager
YOLO v2	You Only Look Once Version 2
YOLO v3	You Only Look Once Version 3

LIST OF APPENDICES

APPENDIX A: Python Script	56
---------------------------	----

CHAPTER 1

INTRODUCTION

1.1 General Introduction

In this section, concept, applications and some related terms of deep learning are discussed shortly. Also, the concept and applications of artificial intelligence (AI) are roughly discussed.

1.1.1 Artificial Intelligence (AI)

John McCarthy stated the term AI in 1955 and defined as “artificial intelligence is science and engineering of making intelligent machines”(McCarthy, 2007; Dash and Subudhi, 2016). The definition of AI for nowadays is defined as “the study and design of intelligent agents”, which means an intelligent agent is a system that detects its environment and then maximizes its success rate (Dash and Subudhi, 2016).

Besides, these processes consist of learning, reasoning, and self-correction. Furthermore, the term “artificial intelligence” was officially invented in 1956. After developing the first programmable computers, people wondered the machines might become intelligent (Lovelace, 1842). The applications of the AI are divided into five categories, which are reasoning, knowledge, planning, communication, and perception. In addition, the AI trends involve in various sectors such as data security, manufacturing, finance, automotive industry and so on. Figure 1.1 shows a sample of AI in the automotive industry.



Figure 1.1: Sample Case of AI in the Automotive Industry.

1.1.2 Introduction of Deep Learning (DL)

In this section, deep learning (DL) will be briefed. DL is a part of machine learning (ML) that learn the features and task directly from data, and the ANN which is the function of the brain (Lecun, Bengio and Hinton, 2015). A Venn diagram is shown in Figure 1.2. Also, DL is referred to as “deep neural networks”, or DNN, which means numerous layers are involved. Besides, a neural network has only a single layer of data. On the other hand, DNN consists of two or more layers.

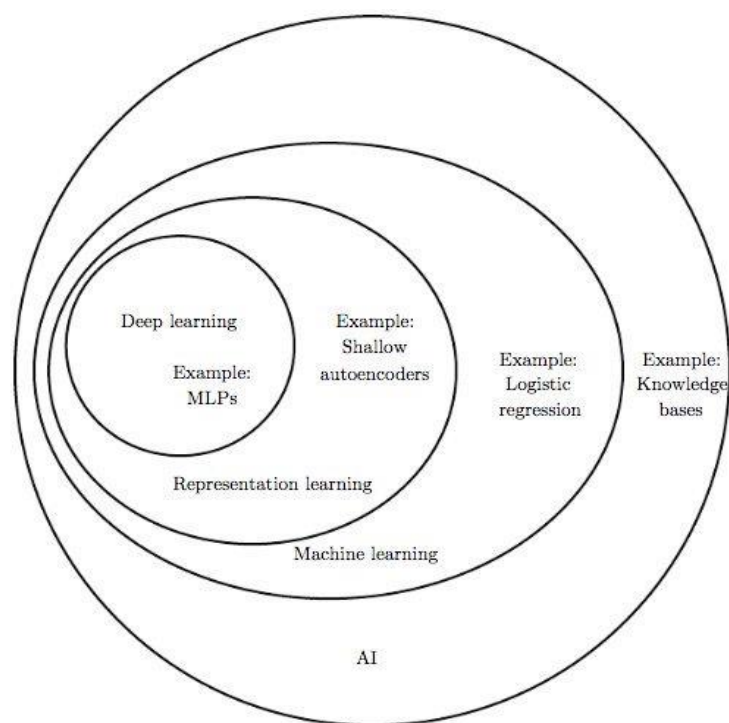


Figure 1.2: A simple Venn diagram illustrates DL is a subset of ML (Goodfellow, Bengio and Courville, 2016).

The computers are trained to work naturally as same as humans are defined as DL. For example, the classification tasks from images, sound or text can be performed using DL. Moreover, a stop sign can be detected or a pedestrian and a lamppost can be recognized by driverless cars using DL. In addition, an enormous number of data and neural network architectures that consist of several layers are used to train the models. DL is able to achieve high accuracy in recognition. Figure 1.3 illustrates the comparison between different parts of AI system with their own different disciplines.

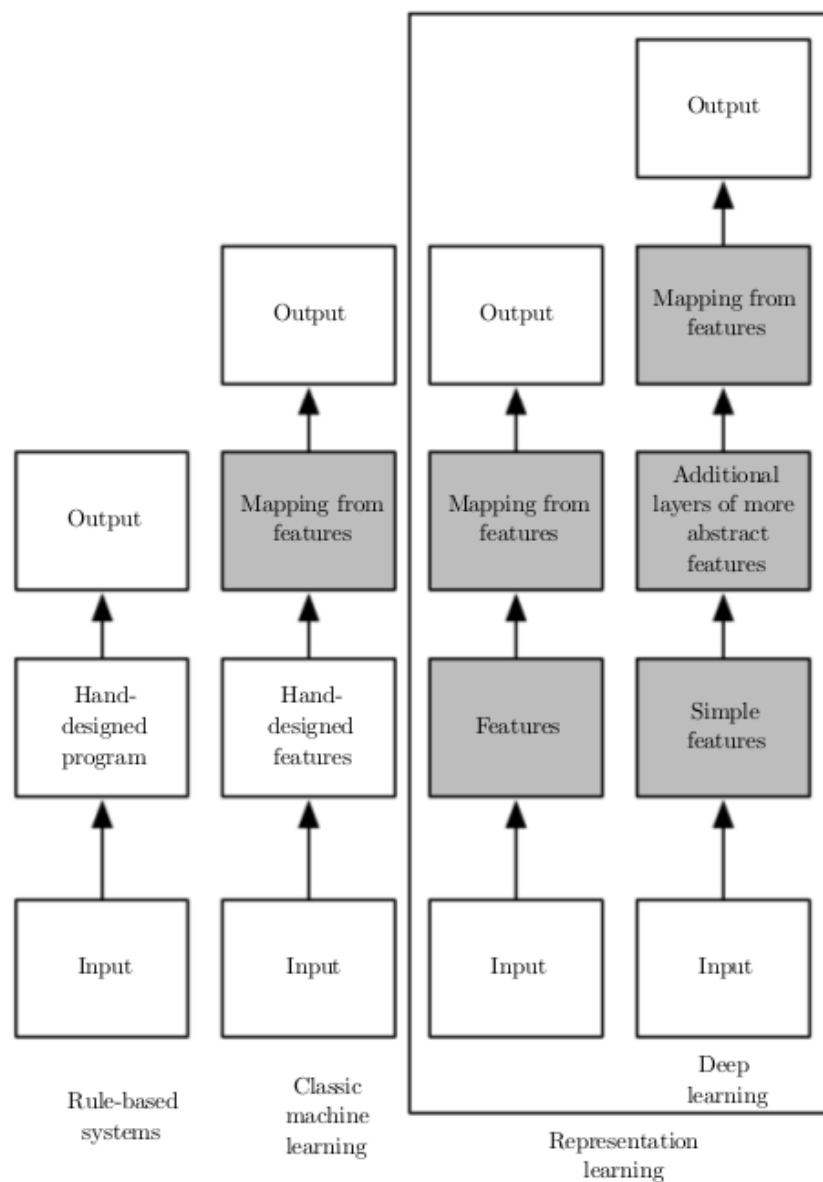


Figure 1.3: Comparison between different parts of AI system with their own different disciplines (Goodfellow, Bengio and Courville, 2016).

Google Brain that finally brought about the productization of DL technologies over numerous Google services was officially established by Andrew Ng. In the talk titled “Deep Learning, Self-taught Learning, and Unsupervised Feature Learning”, Andrew defined the concept of DL as “Using brain simulations, hope to make learning algorithms much better and easier to use; make revolutionary advances in machine learning and AI” (Ng, 2013).

1.1.2.1 Artificial Neural Networks (ANN)

In ML, artificial neural networks (ANN) are the main tools to be used. Haykin described ANN as “An extremely parallel combination of the simple processing unit which can obtain knowledge from the environment through a learning process and store the knowledge in its connections” (Haykin, 1999). ANNs have input, input layer, processing layers, processing element, output layer, and output and an entry (Guresen and Kayakutlu, 2011). Figure 1.4 shows a deep neural network.

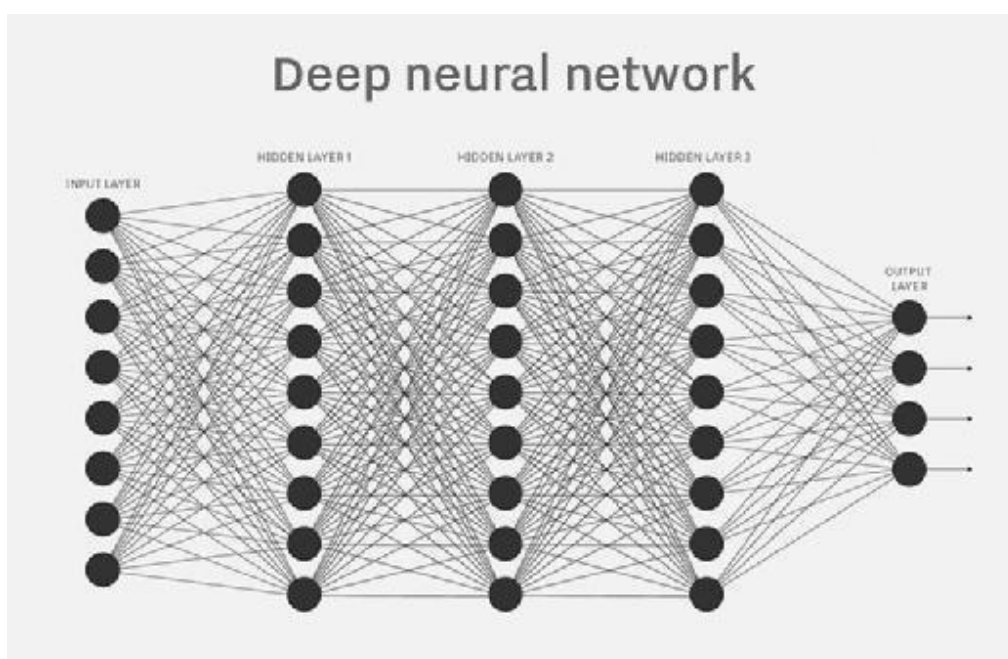


Figure 1.4: Deep Neural Network (DNN).

Additionally, there are various types of learning which are reinforcement learning, supervised learning and unsupervised learning. Meanwhile, there are various types of neural networks which have their own specific functions and

levels of complexity such as feedforward neural network, recurrent neural network, Hopfield networks, convolutional neural networks, and a variety of others.

1.1.2.2 Convolutional Neural Network (CNN)

The convolutional neural networks (CNNs) are a kind of neural network and considered the most popular among all the types of DNNs, as they process data that contain a known grid like topology (Goodfellow, Bengio and Courville, 2016). Besides, CNN eliminates the requirement for manual feature extraction. The feature extraction is automated to build the DL models to become high accuracy computer vision task such as object classification. For example, Figure 1.5 shows classifying handwritten digits using the CNN sequence. Examples can be found in LeNet, AlexNet, VGGNet, GoogLeNet, ResNet, and ZFNet (Wani et al., 2020).

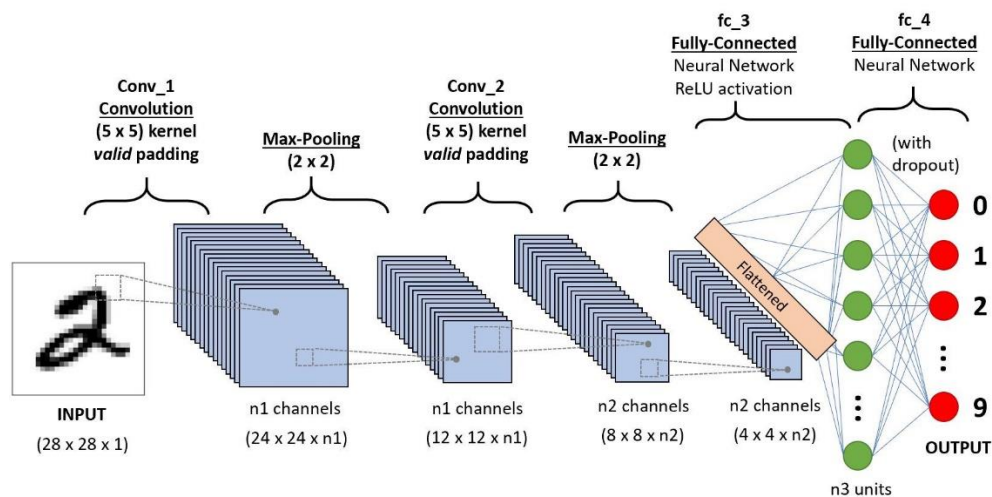


Figure 1.5: A CNN Sequence to categorize the handwritten digits.

1.1.3 Application of Deep Learning

There are a few applications for DL that will rule the world in 2020 and beyond. For example, self-driving cars, deep learning in healthcare, voice recognition, image recognition, automatic colourization, and other applications.

Have you heard about an AI security camera? An AI security camera analyses the images from video surveillance cameras by utilizing computer software programs to recognize vehicles, humans or objects. The advantages of AI home security cameras can provide people and pet detection, facial recognition, unusual behaviour detection, object tracking, and video recognition. Figure 1.6 shows the pet and human detection using a smart home AI security camera. Moreover, smart security camera system also can detect the weapon such as guns as shown in Figure 1.7. This results in that the smart security camera system can identify criminal behaviours with weapons such as guns and knives. Hence, polices can search the crimes much easier than the traditional security system.

In short, the applications in DL bring a lot of benefits and make our life smart and easier. However, applications in deep learning still can be improved.



Figure 1.6: Pet and Human Detection using AI Security Camera.

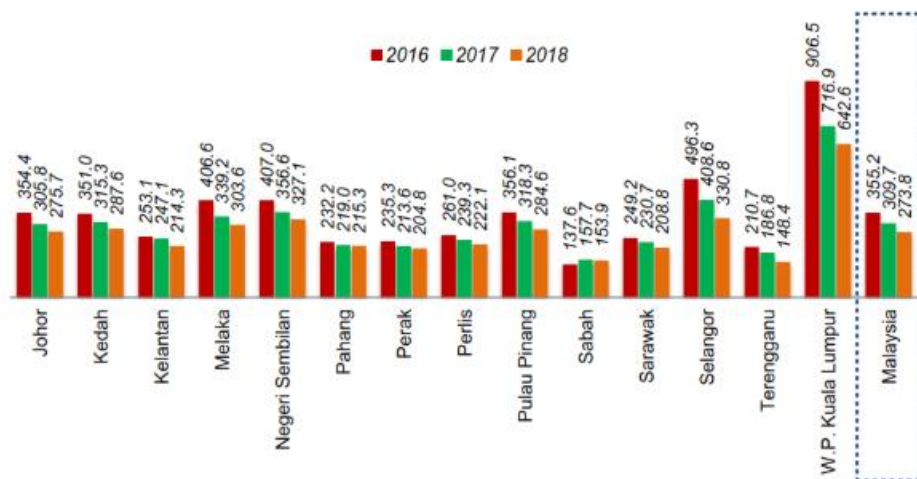


Figure 1.7: Smart Security Camera System from Athena Security identify guns using AI and Cloud Technology.

1.2 Importance of the Study

According to the Crime Statistics, Malaysia, 2019, the first publication of the Department of Statistics Malaysia, the crime rate per 100,000 population between the states of Malaysia, from 2016-2018 is shown in Figure 1.8. Meanwhile, Table 1.1 illustrates the crime rate by type of crimes in Malaysia over these three years. The total number of cases among these three years are 300,184.

With the crime rates of 906.5, 716.9 and 642.6 per 100,000 population, Kuala Lumpur (KL) ranks first for crime among the states of Malaysia in these three years. However, KL has a greater crime index than Singapore from 2016-2018 as shown in Figure 1.9. Therefore, this study aims to develop a smart security camera system that detects crime and alert owners.



Notes:
 1) Derivation ratio using estimate population
 2) W.P. Kuala Lumpur includes W.P. Putrajaya
 3) Sabah includes W.P. Labuan

Data source: Royal Malaysia Police

Figure 1.8: Crime rate per 100,000 population between the states of Malaysia, 2016-2018 (Department of Statistics Malaysia, 2019).

Table 1.1: Crime rate by type of crimes in Malaysia, 2016-2018 (Department of Statistics Malaysia, 2019).

	2016	2017	2018
Violent Crime	22,326	21,366	16,902
Property Crime	90,028	77,802	71,760

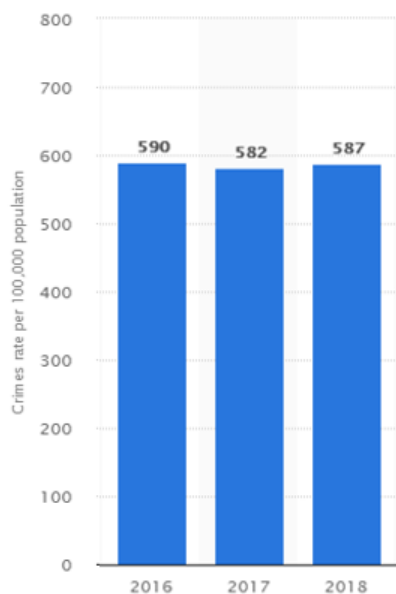


Figure 1.9: Crime rate per 100,000 population in Singapore, 2016-2018 (R. Hirschmann, 2020).

1.3 Problem Statement

A smart security camera consists of the object or human detection and also image classification using deep learning. The smart security camera also can detect crimes through their behaviours. The false alarms are raised in the event of blocking, brightness change, false detection of criminal behaviour and huge storage for footage is required due to the camera is recording continually. In addition, the error of detection in weapon also causes false alarms. Hence, the objectives of this project are to increase the accuracy and speed of detection of the system.

The size of a camera is suggested to be small enough but provides good performance and low cost. Besides, the CNN application generally is applied on the computer (PC) or laptop, but the required computational power is high. Hence, a single-board computer is considered to utilize a CNN application due to it has low power consumption and small in size.

1.4 Aim and Objectives

The purposes of this project are:

- To develop a crime detection model using transfer learning and fine tuning
- To integrate the trained model into an embedded security camera system
- To evaluate the performance of crime detection in terms of speed and accuracy

1.5 Scope and Limitation of the Study

The study is focused on crime detection on the smart security camera. The process of developing this system consists of a combination of hardware and software. The hardware used are Raspberry Pi 3 Model B+ (RP3), micro SD card, camera board for RP3 with 5MP, and Intel Movidius Neural Compute Stick 2 (NCS2). Meanwhile, the software used are Ubuntu 18.04 Operating System, PuTTY, Raspbian Operating System with Stretch Lite (RaspSL), OpenCV, Caffe, and Software Development Kit for NCS2 (NCSDK2).

The limitations are the camera provides a low quality of the image and it is difficult to detect the distant objects or targets due to the low megapixel. Additionally, switch socket outlet and adapter are required to provide power supply for the single-board computer (RP3).

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

The examples and applications of image classification using deep learning are discussed in this chapter.

2.2 Applications using CNNs

In the paper titled “ImageNet Classification with Deep Convolutional Neural Networks”, a large set of high-resolution images with the total of 1.2 million in the ImageNet LSVRC-2010 contest were categorised using deep CNN into the 1000 distinct classes (Krizhevsky, Sutskever and Hinton, 2017). A dataset of beyond 15 million high quality labelled images are categorised into around 22,000 categories is called ImageNet (ImgNet).

The outputs of adjacent groups of neurons in the similar kernel map can be concluded using pooling layers in CNNs. Figure 2.1 illustrates the architecture of CNN which consists of eight layers in total, five in which are convolutional while the following three are fully connected.

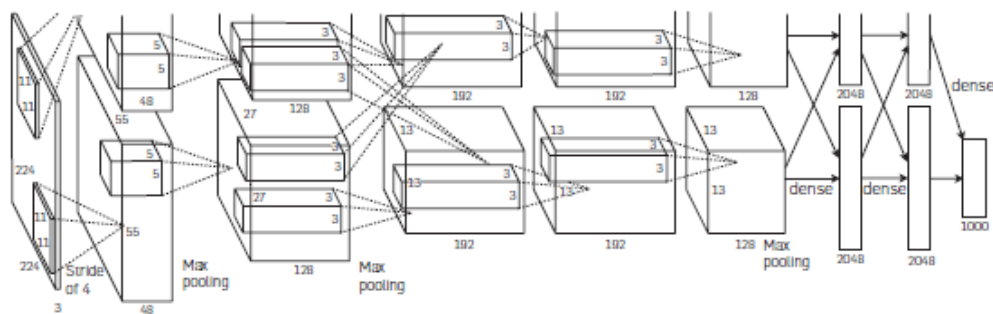


Figure 2.1: A CNN architecture (Krizhevsky, Sutskever and Hinton, 2017).

There were two types of data augmentation that involved in generation image translations and horizontal reflections, and alteration the intensities of the RGB channels to train the images.

Figure 2.2 shows the eight tested images with the top five predictions. The red colour bar resulted in the correct label.



Figure 2.2: Eight tested images with Top Five Predictions.

A camera trap images with DL was introduced in the papers titled “ Identifying animal species in camera trap images using deep learning and citizen science”, and “Automatically identifying, counting, and describing wild animals in camera-trap imaged with deep learning”.

The characteristics of the cameras must be independent gadgets, portable, activated by movement, and infrared sensors for the passing animals (Willi et al., 2019). Based on Figure 2.3, a convolutional layer is converted from an input layer, then an output layer is generated through the end of the process of a CNN architecture. The final output can be obtained from the output layer.

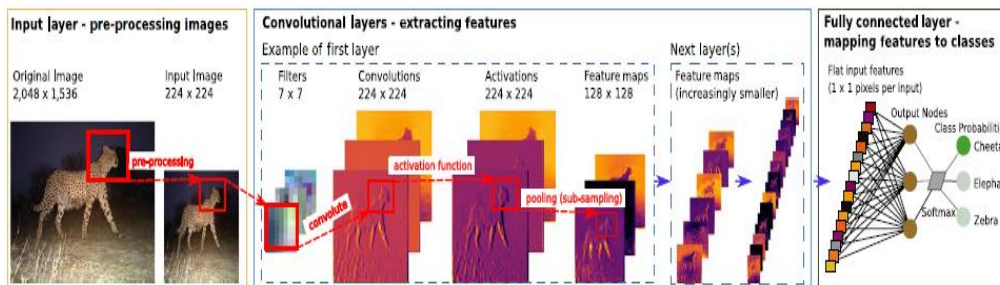


Figure 2.3: A CNN architecture proceed with an input layer to an output layer (Willi et al., 2019).

The four camera datasets used were Camera CATalogue (CC), Snapshot Serengeti (SS), Elephant Expedition (EE), and Snapshot Wisconsin (SW) (Willi et al., 2019). Moreover, transfer-learning was applied to improve the model to become more accurate essentially for small datasets and deduct the training time of the model. However, the classification can be improved in the future due to the less accurate for rare species.

There were four tasks for animal detection. The first step was to identify images that consisted of animals that were animal or empty. The second step was identifying the species of the animal with the top five predictions. After that, the number of animals in the images was counted using DL. And, the last step was defining the additional animal attributes and their behaviours. Figure 2.4 shows the animals can be identified, counted, and described in camera-trap images successfully using DNN. Table 2.1 illustrates different deep learning architectures has different performances.

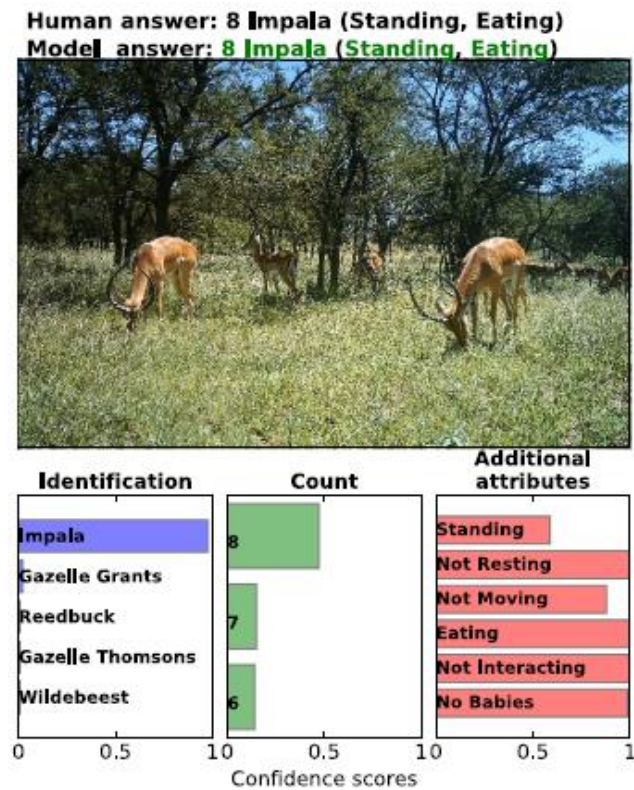


Figure 2.4: Animals can be identified, counted, and described in camera-trap images successfully using DNN (Norouzzadeh et al., 2018).

Table 2.1: Comparison of different deep learning architecture and their performance (Norouzzadeh et al., 2018).

Architecture	No. of layers	Short description
AlexNet	8	A landmark architecture for deep learning winning ILSVRC 2012 challenge (31).
NiN	16	Network in Network (NiN) is one of the first architectures harnessing innovative 1×1 convolutions (49) to provide more combinational power to the features of a convolutional layers (49).
VGG	22	An architecture that is deeper (i.e., has more layers of neurons) and obtains better performance than AlexNet by using effective 3×3 convolutional filters (26).
GoogLeNet	32	This architecture is designed to be computationally efficient (using 12 times fewer parameters than AlexNet) while offering high accuracy (50).
ResNet	18, 34, 50, 101, 152	The winning architecture of the 2016 ImageNet competition (25). The number of layers for the ResNet architecture can be different. In this work, we try 18, 34, 50, 101, and 152 layers.

In the paper titled “Age and Gender Classification using Convolutional Neural Network”, it described age and gender were classified automatically using deep CNN was introduced. The face recognition techniques can be proceeded using deep CNN (LeCun et al., 1989). Besides, the unfiltered face

images for gender and age can be classified using Adience benchmark (Eidinger, Enbar and Hassner, 2014). Hence, the accuracy of CNN design was tested using Adience benchmark. The age classification errors mostly were affected by occlusions, blur and low resolution of the system while the gender estimation errors were the group of babies and young children due to the gender were no visibility at these stages of age.

In the paper titled “Car Parking Occupancy Detection Using Smart Camera Networks and Deep Learning”, a detection for car parking occupancy using CNN. The two CNN architectures were tested which were *mAlexNet* (*mAN*) and *mLeNet* (*mLN*). The difference between *mAN* and *mLN* was that the *mAN* was based on Alexnet while *mLN* was based on LeNet-5 in the CNN architecture. The max pooling (MP) and linear rectification (ReLU) were included in the first three convolutional layers of *mAN*. However, local response normalization (LRN) was included in the first two convolutional layers only. Meanwhile, *mLN* consisted of two convolutional layers and following by MP and two fully connected layers which were based on LeNet-5. The comparison of *mLeNet* and *mAlexNet* in CNN architecture is shown in Table 2.2.

Table 2.2: Comparison of *mLeNet* and *mAlexNet* in CNN architecture (Amato et al., 2016).

<i>net</i>	<i>conv1</i>	<i>conv2</i>	<i>conv3</i>	<i>fc4</i>	<i>fc5</i>
<i>mLeNet</i>	30x11x11+4 pool 5x5+5 -	20x5x5+1 pool 2x2+2 -	-	100 ReLU	2 soft-max
<i>mAlexNet</i>	16x11x11+4 pool 3x3+2 LRN, ReLU	20x5x5+1 pool 3x3+2 LRN, ReLU	30x3x3+1 pool 3x3+2 ReLU	48 ReLU	2 soft-max

The two datasheets used were PKLot and CNRPark. Table 2.3 indicates the experimental results of a single camera and multi-camera in *mLN* and *mAN*. It showed the accuracy of *mAN* was higher than the *mLN* in single-camera and multi-camera experiments, which was 0.996 or 99.6 %. Hence, the car parking space occupancy can be detected and classified using CNN.

Table 2.3: Experiment Results of a single camera and multi-camera in *mLeNet* and *mAlexNet* (Amato et al., 2016).

SINGLE CAMERA EXPERIMENTS				
<i>train</i>	<i>test</i>	<i>net</i>	<i>base lr</i>	<i>accuracy</i>
A (even)	A (odd)	mLeNet	0.001	0.993
		mAlexNet	0.01	0.996
A (odd)	A (even)	mLeNet	0.001	0.982
		mAlexNet	0.005	0.993
B (even)	B (odd)	mLeNet	0.001	0.861
		mAlexNet	0.01	0.911
B (odd)	B (even)	mLeNet	0.001	0.893
		mAlexNet	0.005	0.898

MULTI CAMERA EXPERIMENTS				
<i>train</i>	<i>test</i>	<i>net</i>	<i>base lr</i>	<i>accuracy</i>
A	B	mLeNet	0.0001	0.843
		mAlexNet	0.001	0.863
B	A	mLeNet	0.001	0.842
		mAlexNet	0.0005	0.907

2.3 A Searching Process in 3D Space for Human Detection using Camera and Scene Knowledge

In the paper titled “Human Detection by searching in 3D Space Using Camera and Scene Knowledge”, human detection using the 3D search method instead of the 2D search method. A single camera can estimate the 3D target position. The objective of human detection was to find all the humans in a single image.

A significant category with a large number of high-performance representative human detection systems is known as sub-window classification (Li, Wu and Nevatia, 2008). Moreover, the detection was completed by computing all potential sub-windows in the 2D image. However, the detection performance may be affected by different camera settings. Hence, the new strategy was proposed.

The camera setting were assumed to perform an object search in the 3D world space rather than the 2D image space. All potential positions of objects in the view were included using 3D scanning grid. The detection results are shown in Figure 2.5(b) and Figure 2.5(c).

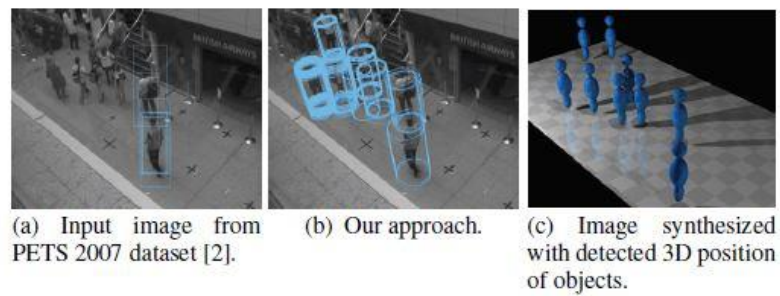


Figure 2.5: Pedestrian Detection Results in 2D search and 3D search (Li, Wu and Nevatia, 2008).

By eliminating the detection errors, a scene knowledge was applied. Then, to obtain a subset of foot positions, the homography was combined with background deduction for detection with the purpose of reducing the search space during detection. An approach overview is shown in Figure 2.6.

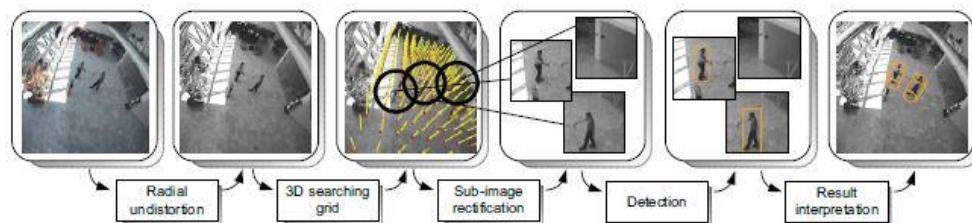


Figure 2.6: An Approach Overview (Li, Wu and Nevatia, 2008).

Figure 2.7 shows the searching process was performed using a 3D search grid. The yellow lines estimated a person at a grid point. Moreover, it also can be improved by appending scene knowledge. Typically, human detectors were practised on images with constant size and humans' sight from a distant camera depending on sub-window classification.



Figure 2.7: A searching process using 3D Search Grid from datasheet (PETS 2007) (Li, Wu and Nevatia, 2008).

The same detector was used to determine the rate of detection and false alarm for 2D and 3D from PETS 2007. The red colour labels indicated the false alarm in the 3D, reflecting the fact that the 3D search was stronger than the 2D search as shown in Figure 2.8 and Figure 2.9. It can be observed that the 3D search was stronger than the 2D search.

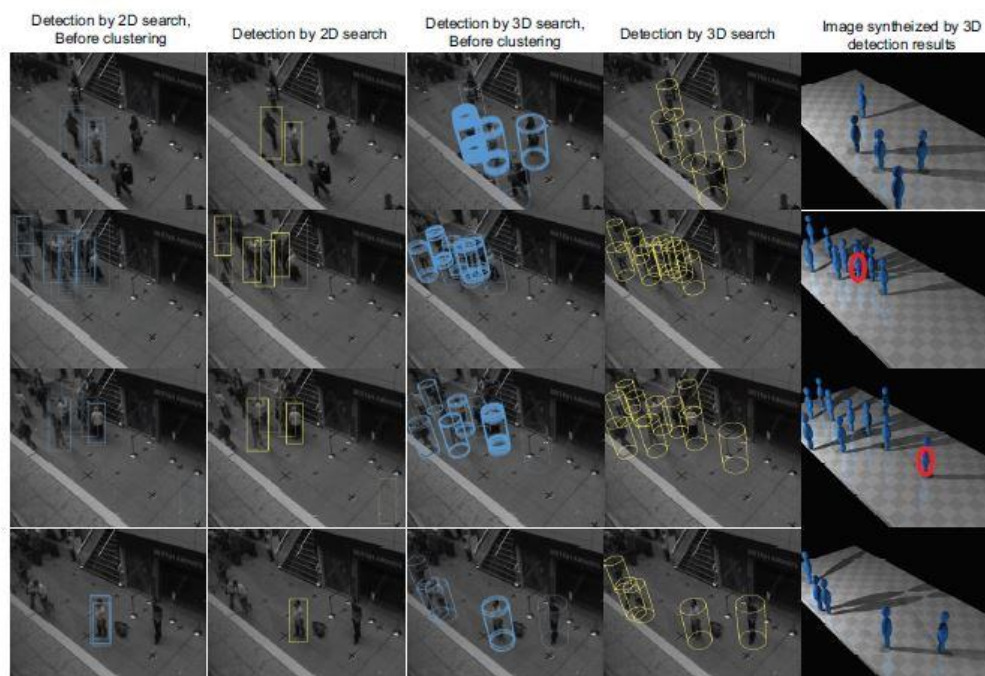


Figure 2.8: Sample detection results for 2D search and 3D search on datasheet (PETS 2007)(Li, Wu and Nevatia, 2008).

Figure 2.9 shows the sample detection results for 2D search and 3D search from datasheet CAVIAR INRIA. In addition, the different failure modes were labelled with different colours as shown in Table 2.4.

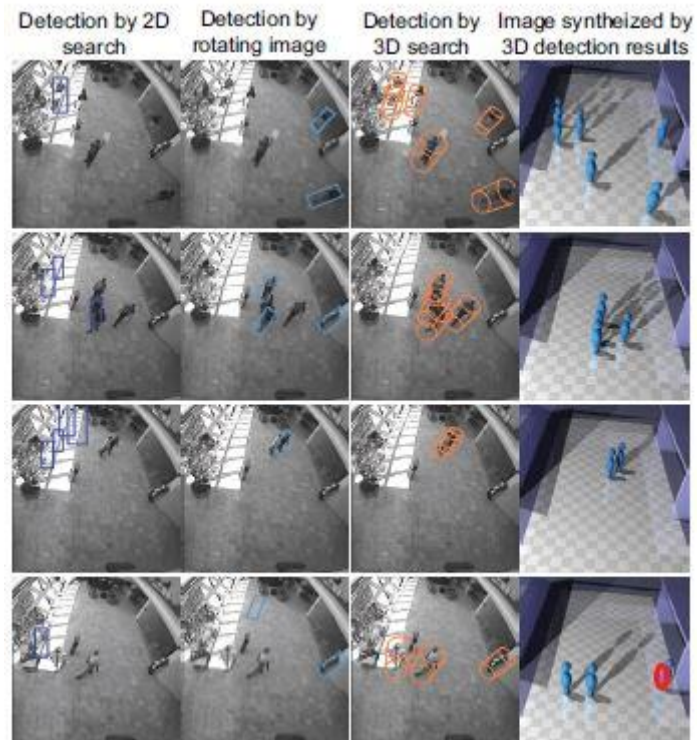


Figure 2.9: Sample detection results for 2D search and 3D search on datasheet (CAVIAR INRIA)(Li, Wu and Nevatia, 2008).

Table 2.4: Failure modes and Colours for 3D search on CAVIAR INRIA datasheet.

Failure Modes	Colour (Circles)
Walking posture	Yellow
Blocking in crowds	White
Top-down view	Blue
Crouching and Bending	Orange

The overall detection rate was low because of the occlusion, pose variation and low contrast.

In short, there were a lot of benefits to this model. It could approximate an object position in 3D. Moreover, the adversary effect of camera view was deducted on detection performance using image rectification, and it was flexible to combine with any patched-based detector (Li, Wu and Nevatia, 2008).

2.4 Applications using YOLO v3 Algorithm

In the paper titled “Application Research of Improved YOLO V3 Algorithm in PCB Electronic Component Detection”, it described the advancement of the algorithm using YOLO v3 with a real and virtual picture of PCB for the training process. The YOLO v3 network structure and the layer of YOLO were extended by removing the features of the network layer to detect the small objects. Figure 2.10 shows the flowchart of PCB detection.

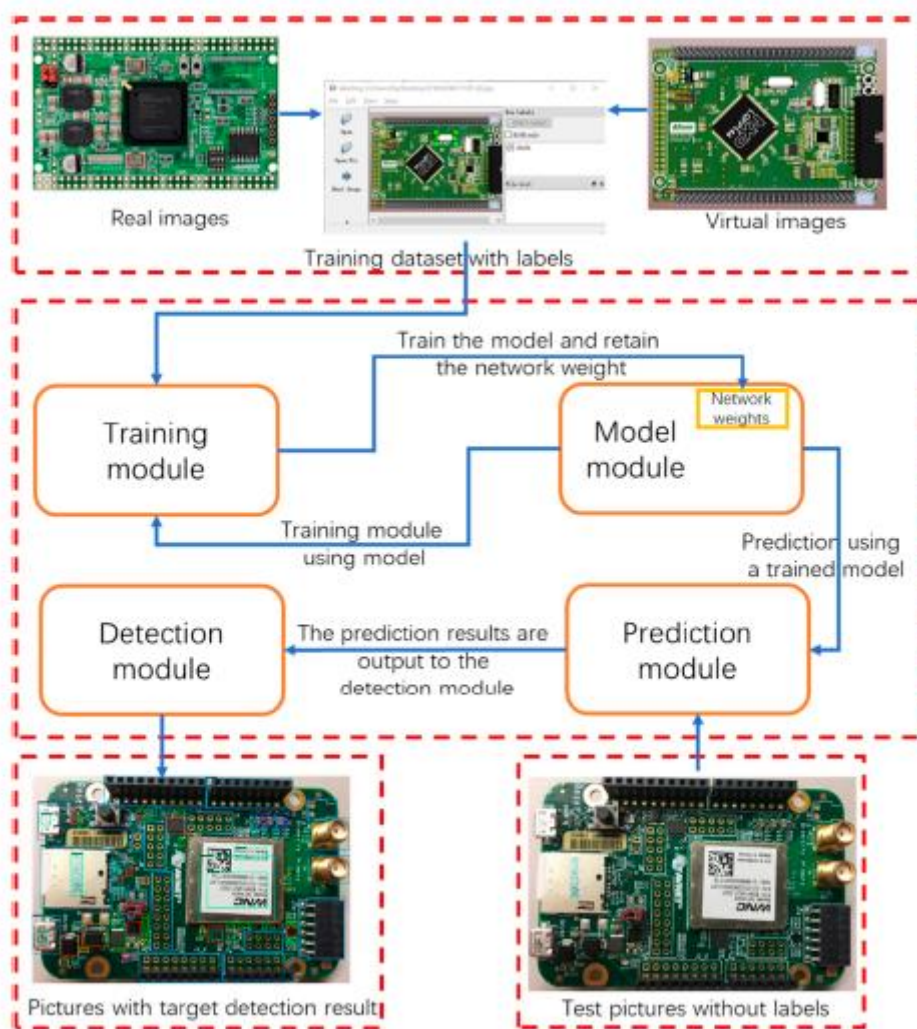


Figure 2.10: General Flowchart of PCB detection (Li et al., 2019).

The backbone network of YOLOv3 was Darknet-53 (D53). The anchor box size ratio of YOLO v3 to detect the object was predicted using K-means clustering. Moreover, the bounding box prediction was completed by the anchor box to define the object in terms of height and width. Before training, the input pictures were resized to 416×416 using YOLOv3. Next, the performance of this PCB component detection was evaluated using different network structures as displayed in Table 2.5. The mean average precision (mAP) and the speed of object detection were recorded for the performance. The mAP of category F was greater than others as shown in Table 2.5 which was 93.07 %.

Table 2.5: mAP and speed of detection for PCB component detection using Different methods (Li et al., 2019).

Category	A	B	C	D	E	F
Method	Faster RCNN	SSD (single-shot multi-box detectors)	YOLO V3 + COCO Dataset 9 Anchors	YOLO V3 + PCB Dataset 9 Anchors	YOLO V3 + PCB Dataset 9 Anchors + Bbox800	YOLO V3 + 4 Outputs + PCB Dataset 12 Anchors + Bbox800
mAP (%)	80.25	83.16	77.14	79.88	76.43	93.07
Run time (sec/img)	0.55	0.32	0.39	0.385	0.395	0.41

In the paper titled “Object Detection and Tracking using YOLO v3 Framework for Increased Resolution Video”, the pre-trained model YOLO v3 was applied for vehicle detection and tracking purpose. Besides, a powerful GPU system was required to support YOLO v3 in increasing the accuracy and speed of object detection.

Furthermore, the implementation of the system was planned to work in the online and offline process using the YOLO v3 pre-trained model as shown in Figure 2.11. After training, a YOLO v3 weights file was generated.

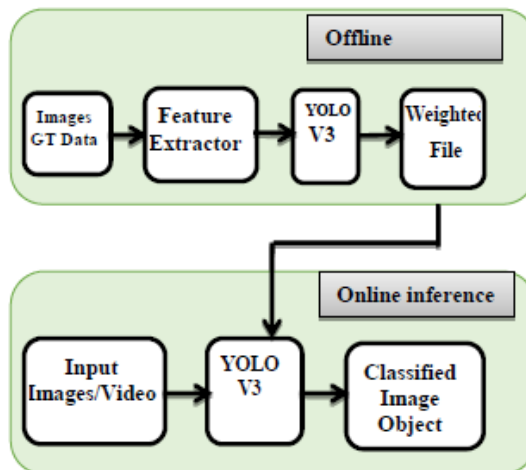


Figure 2.11: Implementation of System using YOLO v3 pre-trained model (Shakil, Rajjak and Kureshi, 2020).

In YOLO v3, the network architecture used was D53 and the libraries used were OpenCV and TensorFlow. The results of the experiment show that the object can be identified as a car and tracked as indicated in Figure 2.12.

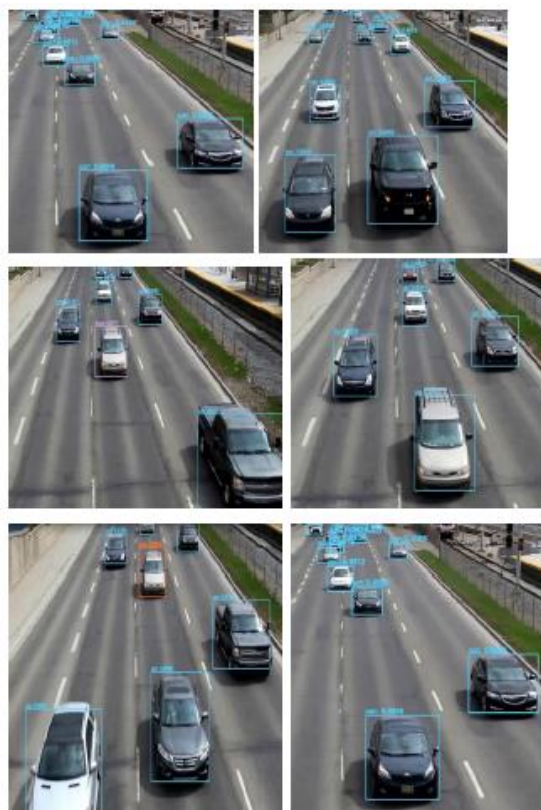


Figure 2.12: Object Detection and Tracking as cars (Shakil, Rajjak and Kureshi, 2020)

Next, the results for mAP, execution time, and frame per second (FPS) were compared using different methods as shown in Table 2.6. Although the YOLO v3 model was not the highest in each part result, it obtained good results in general. At the same time, the accuracy of the system was measured in different methods. According to Table 2.7, YOLO v3 had a high accuracy which was 80 %.

Table 2.6: Comparison of the Performance of the System using Different Methods (Shakil, Rajjak and Kureshi, 2020).

Methods	mAP (%)	Execution Time Sec./Frame	Rate (fps)
SS+FRCN	66.9	1.72	0.6
SS+HYPERNET*	76.3	0.20	5
Faster R-cnn (vgg 16)	73.2	0.11	9.1
Faster R-cnn (Resnet 101)	83.8	2.24	0.4
YOLO	63.4	0.02	45
SSD 300	74.3	0.02	46
YOLO v2 (544x544)	78.6	0.03	40
YOLO v3(416x416) our work	76.7	0.018	55

Table 2.7: Comparison of Accuracy of System in Different Methods (Shakil, Rajjak and Kureshi, 2020).

Methods	Faster R-CNN	R-FCN	SSD @512	YOLO @554	R-FCN **	Our method YOLO v3 @416
Accuracy	70.4	77.6	74.9	73.4	80.5	80

In summary, the object detection can be improved using YOLO v3 model because the speed of detection and accuracy were performed in good results. The mAP for PCB detection and vehicle detection were 93.07 % and 76.7 % respectively.

2.5 Summary

To summarise the reviews, there are many applications using CNN nowadays. However, the classification methods using CNN can be further improved to rise the accuracy of the model or system. There are also different CNN architectures that are used for classification. Moreover, the searching or classifying process can be enhanced into the 3D search method instead of 2D search method in the future as shown in Section 2.3. Furthermore, the YOLO v3 model is applied in object detection due to the high accuracy and speed of detection.

CHAPTER 3

METHODOLOGY AND WORK PLAN

3.1 Introduction

In this chapter, the software and hardware used in this project are listed out clearly. The procedures of this project are shown step by step.

3.2 Design Requirements

The design requirements are split into two groups that are hardware and software requirements. Integration of hardware and software is to work well in the system.

3.2.1 Hardware Requirements

The hardware required for this project are Raspberry Pi 3 Model B+ (RP3), Intel Movidius Neural Compute Stick 2 (NCS2), Camera Board for RP3 with 5MP, and micro SD Card with 128GB and 100MB/s

Raspberry Pi 3 Model B+ or RP3 is a single board computer with low cost, small in size that connects into a computer monitor or TV, a mouse, and a keyboard. The laptop has a built-in keyboard and can connect with a mouse through the USB port. Therefore, the work for RP3 can be observed and controlled through the laptop. RP3 provides USB ports, WIFI support, and Bluetooth connection. It is ideal to be implemented in this project because it consumes low power. Besides, it has small in size and affordable in price. Figure 3.1 shows a RP3.

In addition, NCS2 is a plug and play hardware with a USB-thumb-drive-sized. The NCS2 is a visual processing unit (VPU) on a USB stick. It consumes low power but has high performance on real-time interference and user-friendly. Any single-board computers can be paired with the NCS2. RP3 is paired with the NCS2 in this project. Figure 3.2 shows an Intel Movidius NCS2.

The camera board for RP3 with 5MP is chosen for this project. Although the size of the camera is small, the performance of the camera is satisfactory and

it is reasonable in price. The camera is used to capture images or record moving images. Figure 3.3 shows a 5MP camera board.

A micro SD card with 128GB and 100MB/s is chosen due to its large storage and high read speed. A micro SD Card with 128GB and 100MB/s is shown in Figure 3.4.



Figure 3.1: Raspberry Pi 3 Model B+ (RP3)



Figure 3.2: Intel Movidius Neural Compute Stick 2 (NCS2)

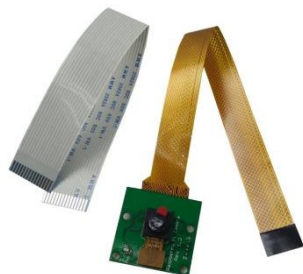


Figure 3.3: Camera Board for RP3 with 5MP



Figure 3.4: Micro SD Card with 128GB and 100MB/s

3.2.2 Software Requirements

The software required for this project are VMware Workstation Player (Virtual Machine), Ubuntu 18.04 (Operating System), Raspbian Stretch Lite (RaspSL), Win32 Disk Imager (Win32), PuTTY, Open Source Computer Vision (OpenCV), Neural Compute Software Development Kit 2 (NCSDK2), Open Visual Inference and Neural Network Optimization (OpenVINO), VNC viewer and TensorFlow (TF).

A virtual machine that can run on a Windows or Linux PC is called VMware Workstation Player (VMware). VMware allows users to run multiple machines synchronize on a single host computer. Ubuntu 16.04 is installed in the VMware Workstation Player.

An open-source Linux distribution based on Debian is called Ubuntu 18.04. Ubuntu 18.04 is chosen in this project because it can support NCSDK.

Raspbian Stretch Lite or RaspSL is selected among the Raspbian operating systems. The Raspbian Lite is a minimal version of Raspbian and does not contain a graphic user interface (GUI). However, it has a command-line interface (CLI) which is shown automatically when it is connected to a monitor or TV. The Raspbian Lite version executes faster than other Raspbian operating systems due to it does not consist of GUI.

A simple open-source application is used in this project which is Win32 Disk Imager (Win32). The image file is written from CD to an SD card using Win32, to generate a virtual disk drive.

PuTTY is an implementation of SSH and Telnet for the Windows platform. PuTTY is used to connect RP3 to the laptop with SSH.

A library of programming functions generally targeted at real-time computer vision is called OpenCV. In short, OpenCV is a library used for image processing.

Intel Movidius Neural Compute software development kit 2, known as NCSDK2 is the legacy soft development kit, SDK provided for users of the Intel Movidius NCS2. NCS2 is supported by NCSDK2.

OpenVINO is a toolkit to facilitate the optimization of DL models and enable the DL models using an inference engine (IE) in Intel hardware. The Intel hardware includes CPU, GPU, NCS2, and FPGA.

VNC viewer stands for Virtual Network Computing (VNC). VNC viewer facilitates remote desktop sharing over a network connection. RP3 can be controlled with a laptop when sharing the same network connection.

TF is an open-source library for the computation of numerical and large scale ML. Moreover, DNNs can be trained and run by TF. Python language is used in TF due to the ease of learning and implementation processes.

3.3 Setup of Operating System (OS), Dependencies and Software into a RP3

RaspSL will be loaded to the RP3 through a microSD card. Lite version is chosen because it does not have a graphic user interface (GUI). After the installation of OS, an SSH server uses the secure shell protocol to connect the remote computer which is RP3. To enable the SSH server on RaspSL, an empty file called “ssh” is created in the micro SD card. The SSH server enables remote configuration of the RaspSL OS running the RP3. Next, the Wi-Fi chip on RP3 connects to a 2.4 GHz network. A file named “wpa_supplicant.conf” is created and allocated into the micro SD card.

After that, NCSDK2 is installed for the deployment of NCS2. OpenVINO supports a lot of DL libraries, for example, TF, Caffe, and mxnet. Therefore, OpenVINO and OpenCV dependencies are installed into RP3. A virtual environment of OpenVINO is formed on RP3 to support NCS2. The installation of TF into RP3 is completed. VNC viewer is installed into RP3 to smoothen the process of testing and collecting results. The detections are easy to be seen

through the VNC viewer. After installing all the software dependencies, RP3 is ready for the project.

3.4 Model Selection for Training Dataset

The model selection for training the dataset is the You Only Look Once Version 3, or YOLO v3. YOLO is an algorithm that utilizes CNNs for object detection. YOLO is considered one of the top object detection algorithm among other algorithms. Although the accuracy of object detection in YOLO v3 is not the highest, it is a great selection for real-time detection due to its loss of accuracy is low.

The architecture of the feature extractor used in YOLO v3 is Darknet-53 (D53). D53 consists of 53 convolutional layers. D53 is slower than Darknet-19 due to the number of convolutional layers in D53 is more than Darknet-19. Figure 3.5 shows a model of Darknet-53. Besides, the performance of D53 is higher and 1.5 times faster than ResNet-101. The performance of D53 and ResNet-152 are similar but D53 is two times faster (Redmon and Farhadi, 2018). Figure 3.6 displays the comparison of the backbone.

	Type	Filters	Size	Output
	Convolutional	32	3×3	256×256
	Convolutional	64	$3 \times 3 / 2$	128×128
1x	Convolutional	32	1×1	128×128
	Convolutional	64	3×3	
	Residual			
	Convolutional	128	$3 \times 3 / 2$	64×64
2x	Convolutional	64	1×1	64×64
	Convolutional	128	3×3	
	Residual			
	Convolutional	256	$3 \times 3 / 2$	32×32
8x	Convolutional	128	1×1	32×32
	Convolutional	256	3×3	
	Residual			
	Convolutional	512	$3 \times 3 / 2$	16×16
8x	Convolutional	256	1×1	16×16
	Convolutional	512	3×3	
	Residual			
	Convolutional	1024	$3 \times 3 / 2$	8×8
4x	Convolutional	512	1×1	8×8
	Convolutional	1024	3×3	
	Residual			
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 3.5: Model of Darknet-53 (Redmon and Farhadi, 2018).

Backbone	Top-1	Top-5	Bn Ops	BFLOP/s	FPS
Darknet-19 [15]	74.1	91.8	7.29	1246	171
ResNet-101[5]	77.1	93.7	19.7	1039	53
ResNet-152 [5]	77.6	93.8	29.4	1090	37
Darknet-53	77.2	93.8	18.7	1457	78

where

Bn Ops = Billions of Operations

BFLOP/s = Billion Floating Point Operation per Second

FPS = Frame per Second

Figure 3.6: Comparison of Backbone (Redmon and Farhadi, 2018).

A predefined bounding boxes with specific width and height for multiple object detection in a one grid cell, known as anchor boxes. YOLO v3 has 9 anchor boxes while three of them are for each scale. Nine anchors are generated using K-Means clustering. The number of anchor boxes for YOLO v2 is less than YOLO v3, so YOLO v3 has higher accuracy as compared to YOLO v2.

In YOLO v3, the average precision (AP) for small objects is greater than YOLO v2 and Fast RCNN. The mAP is increased when the localization errors are reduced. The ratios of the width to the height of an image and different scales of predictions for the identical object are enhanced by adding the feature pyramid-like method. The overall performance with mAP at 50 % threshold and inference time for YOLO v3 is better than other methods as shown in Figure 3.7. YOLO v3 has faster inference time than other models. For example, FPN FRCN has the highest mAP but longer inference time as compared to YOLO v3.

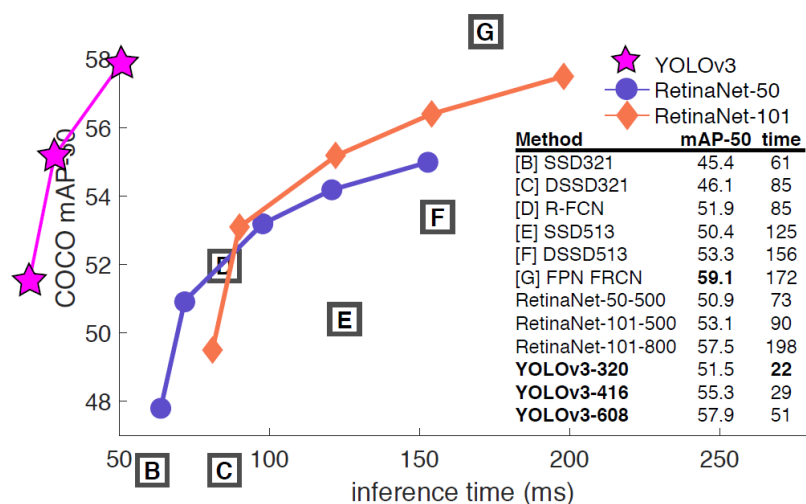


Figure 3.7: COCO Dataset Testing with mAP@0.5 (Redmon and Farhadi, 2018)

In short, YOLO v3 and its feature extraction, D53 are chosen for dataset training due to its overall performance.

3.5 Procedures for Training and Testing YOLO v3 Model

Google Colab (GC) is used for training the YOLO v3 model because it has built-in free 12GB-RAM GPU and easy to work. Besides, GC is a free cloud service and works with Google Drive. The advantages of GC used in this project are an environment for deep learning is provided by GC and the data can be saved in Drive. However, the disadvantage of GC is the runtime is volatile, the runtime must be reconfigured to start training dataset after every twelve hours.

A Google account is signed up and then sign in to GC. Next, all the software dependencies for training are installed in GC. After that, the Darknet environment is set up in GC because the YOLO v3 model is trained using Darknet.

After setting up GC, a dataset for three classes is collected which are gun, helmet, and knife. Data augmentation is a technique that increases the size of the dataset. The images in a dataset can be modified by adjusting rotation, brightness, flip and other methods. For example, the original dataset of the knife only has around three hundred images and then the dataset is expanded to more than one thousand images through data augmentation for training. Next, the process of labelling the dataset in text format is known as data annotation. All the images are labelled using Labelling and saved in text format as shown in

Figure 3.8. After that, all the images and annotations files in .jpg and .txt format are saved in the same folder and then upload to Google Drive. Figure 3.9 shows the annotations in the text file format. The object-class represents the class id of the labelled data. Additionally, x_center and y_center represent the centre of the bounding box in coordination while width and height represent the image dimensions.

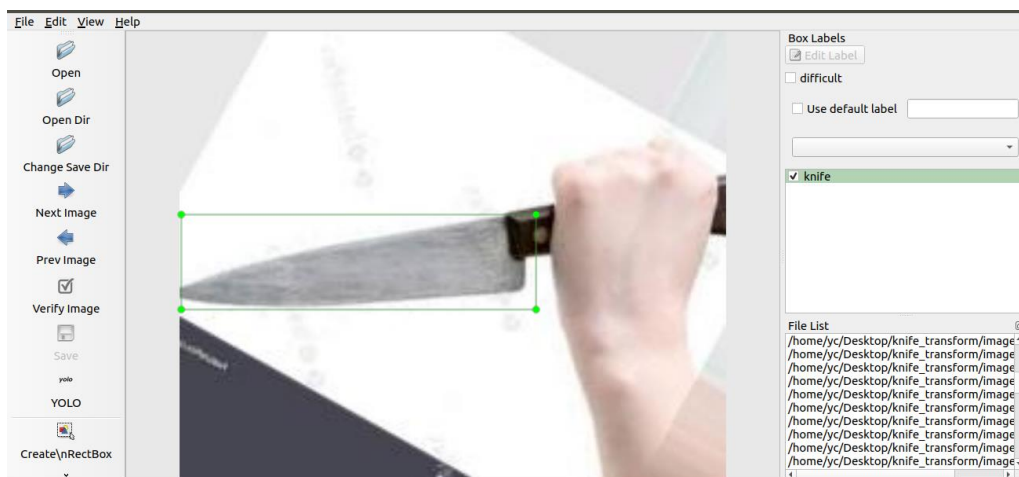


Figure 3.8: Example of Labelling Image using LabelImg.

```
<object-class> <x_center> <y_center> <width> <height>
```

Figure 3.9: Annotation in Text File Format.

Next, the dataset is separated into the training set and testing set by executing a Python script. After running the Python script, two text files are created which are train.txt and test.txt. Furthermore, the *weapon.names* file and the data file, *weapon.data* are created. Figure 3.10 displays the format of the data file, *weapon.data*.

```
classes= 3
train = /content/gdrive/My\ Drive/darknet/darknet/train.txt
valid = /content/gdrive/My\ Drive/darknet/darknet/test.txt
names = /content/gdrive/My\ Drive/darknet/darknet/weapon.names
backup = /content/gdrive/My\ Drive/darknet/darknet/backup/
```

Figure 3.10: Data file, *weapon.data*.

Next, configuration file, *yolov3.cfg* is created for the training process. The training process is started after done preparation of the data file, configuration file, and Darknet. The command is run as shown in Figure 3.11. The training process takes around two days to meet a certain accuracy.

```
!./darknet detector train "/content/gdrive/My Drive/darknet/darknet/weapon.data"  
"/content/gdrive/My Drive/darknet/darknet/yolov3.cfg"  
"/content/gdrive/My Drive/darknet/darknet/darknet53.conv.74" -map -dont_show
```

Figure 3.11: Command for training the dataset.

Moreover, the testing set only has ten per cent of images from the dataset. The testing process depends on the mAP at 0.5 of IoU threshold and tests whether the object is detected or not. When both requirements are achieved, the pre-trained model of YOLO v3, *yolov3.weights* can be downloaded. Figure 3.12 illustrates the flow diagram for the training and testing processes of the YOLO v3 pre-trained model.

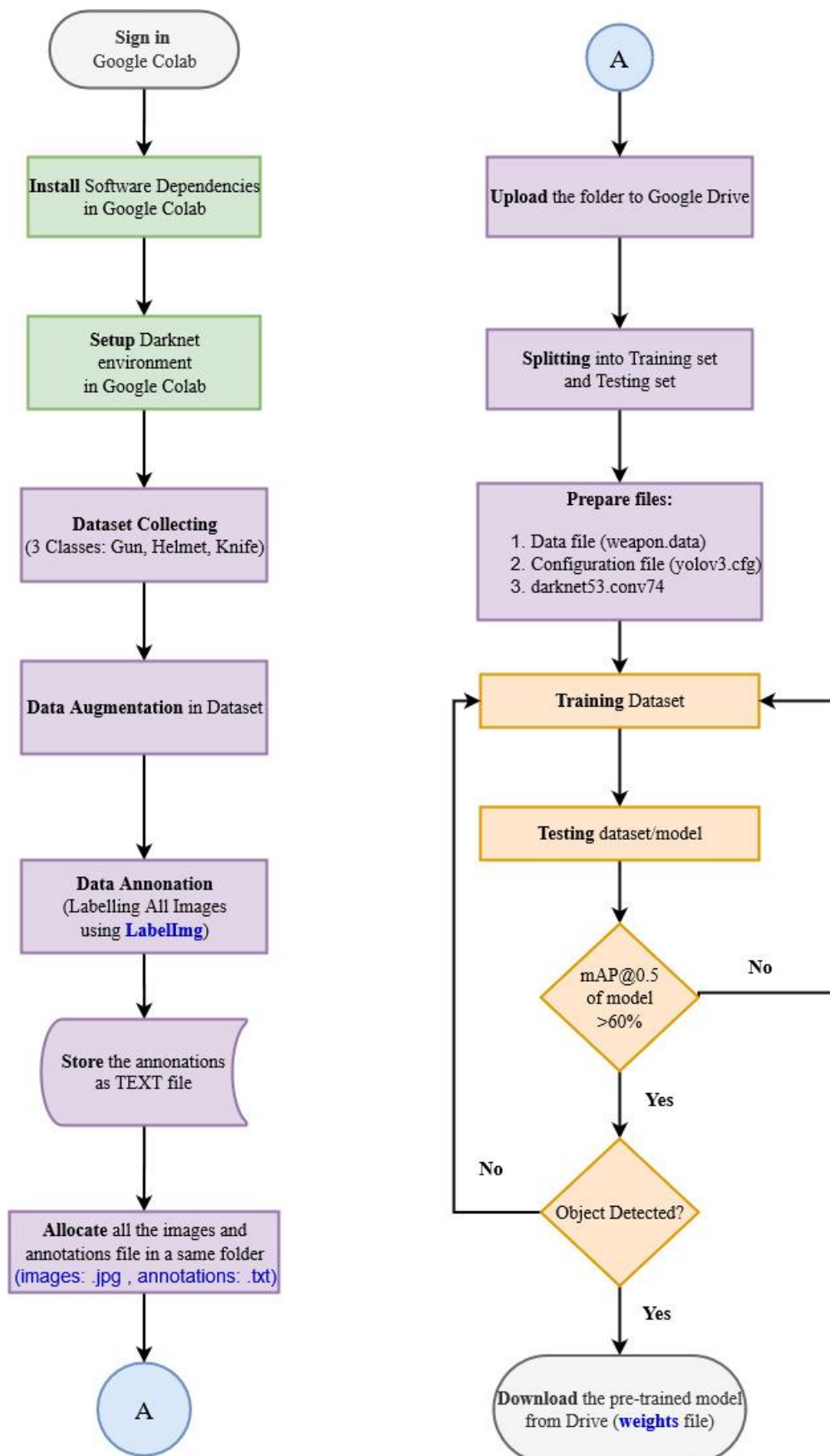


Figure 3.12: Process Flow Diagram of Training and Testing YOLO v3 model.

3.6 Conversion of YOLO v3 model into Intermediate Representation (IR) to load in Inference Engine (IE)

A Python-based command-line tool that converts DL frameworks into an intermediate representation (IR) is known as Model Optimiser (MO). The DL frameworks include Caffe, TF, MXNet, and ONNX (Intel, 2020). Besides, the two main objectives of MO are producing a valid IR and generating an optimised IR. The IR is the model with a pair of files that defines the network topology and consists of weights and biases binary data which are .xml and .bin files respectively.

After converting DL frameworks into IR, the IR files are loaded into the Inference Engine (IE). IE has a common Application Programming Interface (API) that uses to infer input data and perform inference on IR files. The API in IE includes C, C++, and Python to execute IR files and perform an optimisation on inference. Moreover, IE supports targeted hardware devices based on Intel architecture including CPU, GPU, FPGA, and VPU. An overview of the structure of OpenVINO in DL is shown in Figure 3.13.

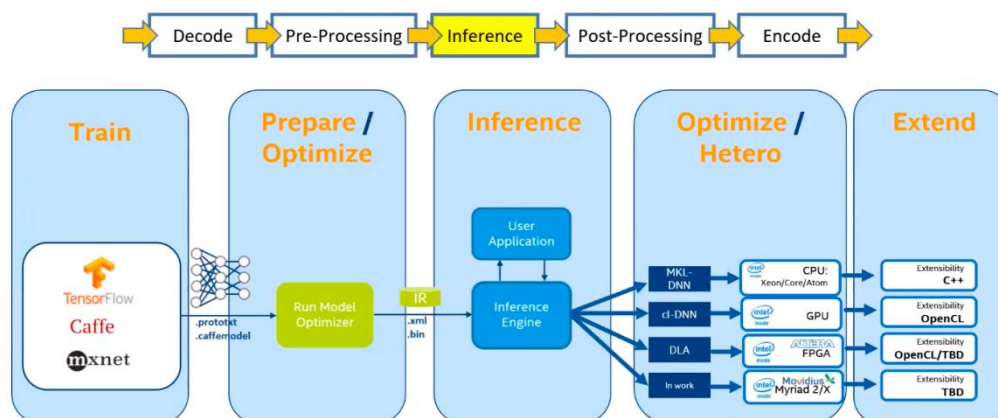


Figure 3.13: Overview of the structure of OpenVINO in DL (Intel, 2019).

First, the YOLO v3 model with configuration and weights files is converted into TF. Next, the TF file (.pb) is converted into IR using MO. IR includes network topology (.xml) and binary data files (.bin) are generated to load into IE. After that, IE optimises the generated IR files and then executes with different devices which are CPU and NCS2. The devices are used to test

the FPS in real-time object detection. Figure 3.14 shows the summary of the conversion into IR and then loads into IE.

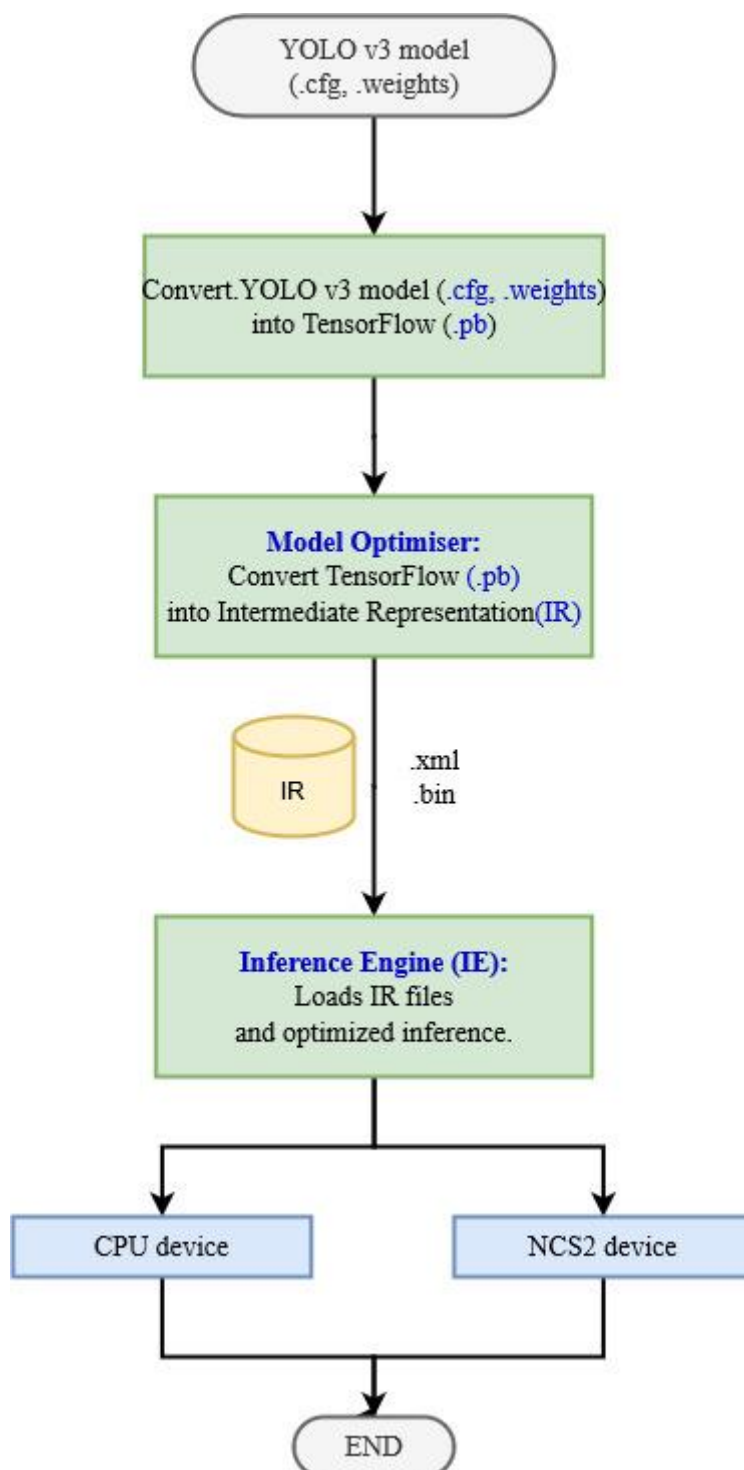


Figure 3.14: Conversion of YOLO v3 into IR, then loads into IE.

3.7 Flowchart of System Operation

The flowchart of overall system operation in RP3 is shown in Figure 3.15. If the camera is not activated, a video file is required to perform the system. NCS2 is required for RP3 to support IR files. Next, the frame is read and processed. If the classes in the model are detected, the bounding box is drawn and labelled relevant classes. If the 'q' is pressed, it will exit and shut down the system. The system is stopped.

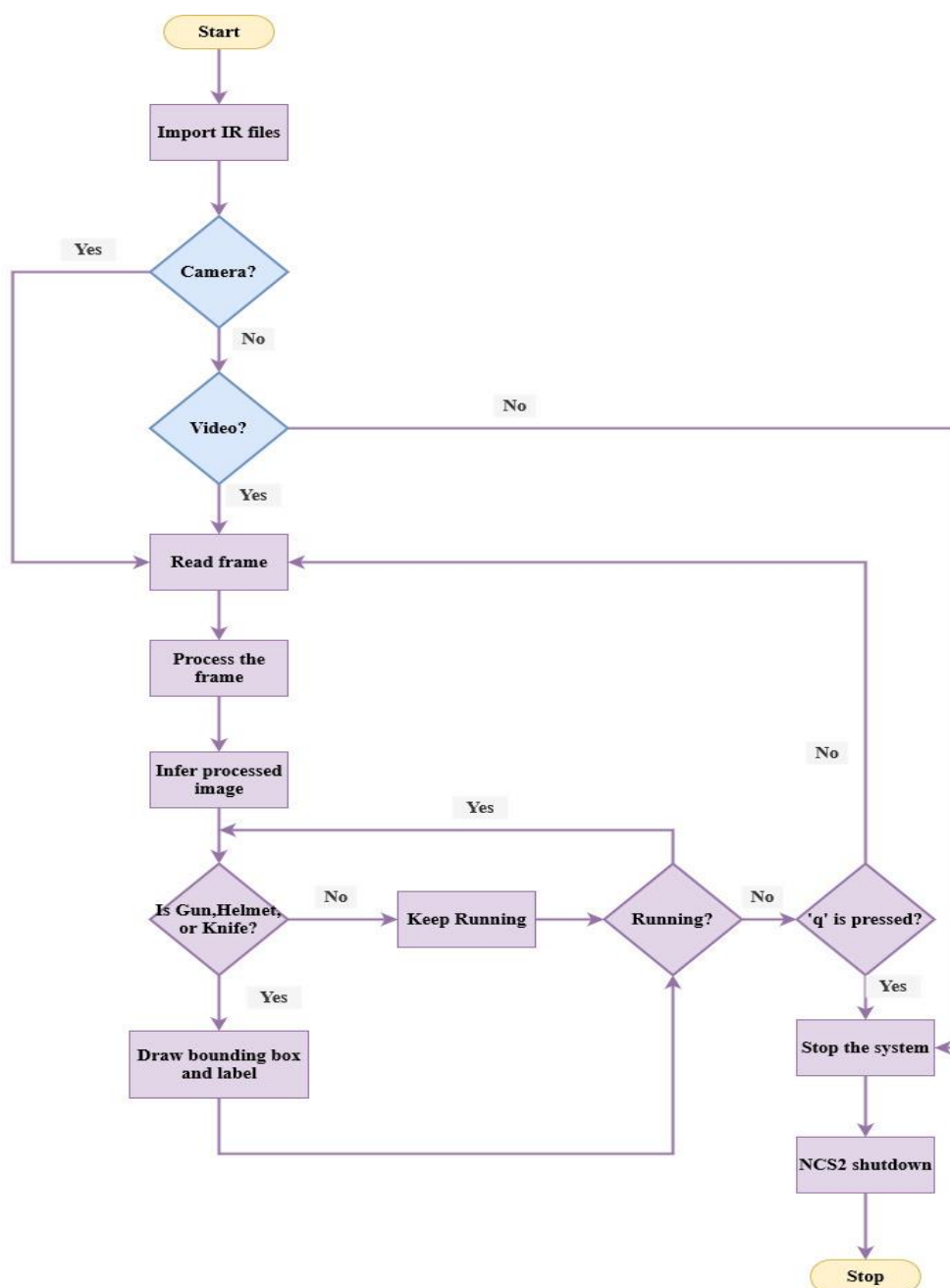


Figure 3.15: Flowchart of System Operation.

3.8 Evaluation Performance of the Model

The performance of the model is divided into four parts, which are precision (P) and recall (R), mean average precision (mAP), crime detection, and FPS of the system.

3.8.1 Measurement of Precision, Recall and Mean Average Precision

P, R and mAP are recorded in this model. P and R are recorded with different confidence scores while mAP is recorded using different intersection over union (IoU) thresholds. P is defined as the percentage of accuracy level of prediction while R is defined as the percentage of all possible positive results that can be found. P and R depend on TP, FP, and FN values. The formula to calculate P and R are shown in equations 3.1 and 3.2 respectively.

$$P = \frac{TP}{TP + FP} \quad (3.1)$$

$$R = \frac{TP}{TP + FN} \quad (3.2)$$

where

P = Precision

R = Recall

TP = True Positive

FP = False Positive

FN = False Negative

Average Precision (AP) is the average precision at a group of eleven recall values from 0 to 1 and the formula is shown in Equation 3.3. The step size of the recall values is 0.1.

$$AP = \frac{1}{11} \sum_{r \in \{0.0, \dots, 1.0\}} P_r \quad (3.3)$$

where

AP = *Average Precision*

r = *Recall value*

P = *Precision at certain Recall Values*

Next, mAP is the average of AP with the given formula as shown in Equation 3.4.

$$mAP = \frac{1}{N} \sum_{i=1}^N AP_i \quad (3.4)$$

where

mAP = *Mean Average Precision*

AP = *Average Precision*

i = *Number of Average Precision*

N = *Total number of classes in the model*

3.8.2 Measurement of Object Detection and FPS

The sample images of the object detection and the frame per second (FPS) of the system with different devices are collected.

Sample images are collected in three classes:

- Gun
- Helmet
- Knife

Testing FPS of the system:

- CPU
- CPU + NCS2
- RP3 + NCS2

3.9 Summary

In short, the YOLO v3 model is trained in GC with Darknet feature extraction and then converted into IR. The performance of the system is evaluated in the terms of P, R, mAP, crime detection and FPS of the system.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

In this chapter, results are recorded and discussed.

4.2 System Overview

In this project, an adapter is required to provide power supply to the RP3. Next, the Pi camera and NCS2 are set up and ready to run the detection. SSH connects RP3 and laptop to allow detections can be viewed on the screen of a laptop. Next, NCS2 is a plugin to accelerate the speed of the detections. The overall embedded system is presented in Figure 4.1. Moreover, Figure 4.2 displays the prototype with NCS2.

When the IR files are loaded, there are two input options to be selected which are camera or video. After that, the video frame is read. The frame should be resized to (416, 416) and provide the shape to a suitable size. Then, the NCS2 run the inference to process the inferred images. Next, if the relevant object is detected, the bounding box is drawn and then label it. The video frame keeps running until 'q' is pressed to exit the window. Lastly, the NCS2 and the whole system are shut down.

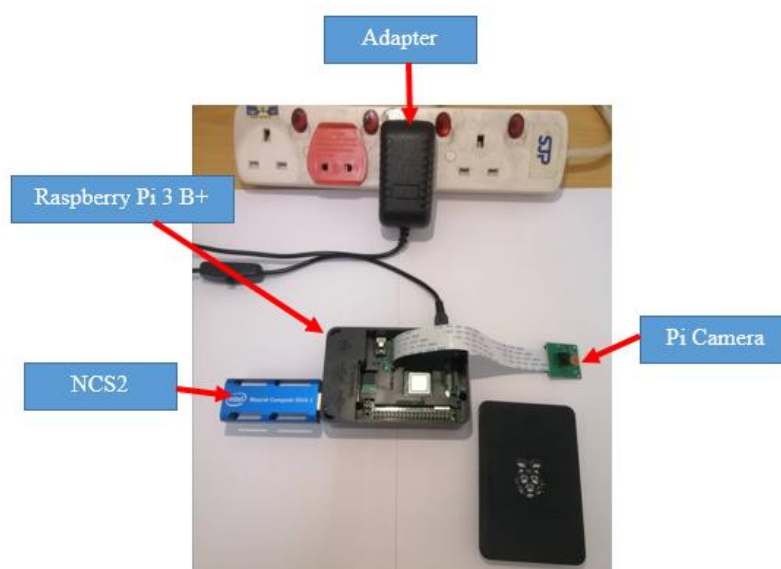


Figure 4.1: Overall System



Figure 4.2: Prototype of RP3 with NCS2 and Pi Camera

4.3 Evaluation Precision and Recall of the Model

In this section, P and R are evaluated from the testing dataset with different confidence scores. There are 400 images from the dataset for the testing process. Besides, the data of P and R for all classes, gun detection, helmet detection, and knife detection are recorded in Table 4.1 and Table 4.2. Precision vs Recall Curve for all classes refers to Figure 4.3. Furthermore, Figure 4.4 displays Precision vs Recall Curve for gun detection. Next, the Precision vs Recall Curve for helmet detection is shown in Figure 4.5. The Precision vs Recall Curve for knife detection is displayed in Figure 4.6. Meanwhile, Figure 4.7 displays the overall Precision vs Recall Curve in this trained model.

Table 4.1: Precision and Recall for All Classes and Gun Detection in Different Confidence Scores.

Class	All Classes		Gun	
	Precision (P)	Recall (R)	Precision (P)	Recall (R)
0.1	0.732	0.668	0.734	0.692
0.2	0.772	0.658	0.759	0.677
0.3	0.791	0.617	0.776	0.621
0.4	0.803	0.602	0.777	0.590
0.5	0.817	0.590	0.789	0.574
0.6	0.820	0.574	0.788	0.554
0.7	0.830	0.542	0.813	0.513
0.8	0.839	0.510	0.835	0.467
0.9	0.859	0.458	0.857	0.431

Table 4.2: Precision and Recall for Helmet and Knife Detection in Different Confidence Score.

Class	Helmet		Knife	
	Precision (P)	Recall (R)	Precision (P)	Recall (R)
0.1	0.641	0.528	0.835	0.795
0.2	0.695	0.506	0.861	0.791
0.3	0.738	0.449	0.86	0.782
0.4	0.765	0.443	0.867	0.773
0.5	0.768	0.432	0.894	0.764
0.6	0.768	0.415	0.902	0.755
0.7	0.780	0.403	0.897	0.709
0.8	0.788	0.381	0.893	0.682
0.9	0.838	0.324	0.883	0.618

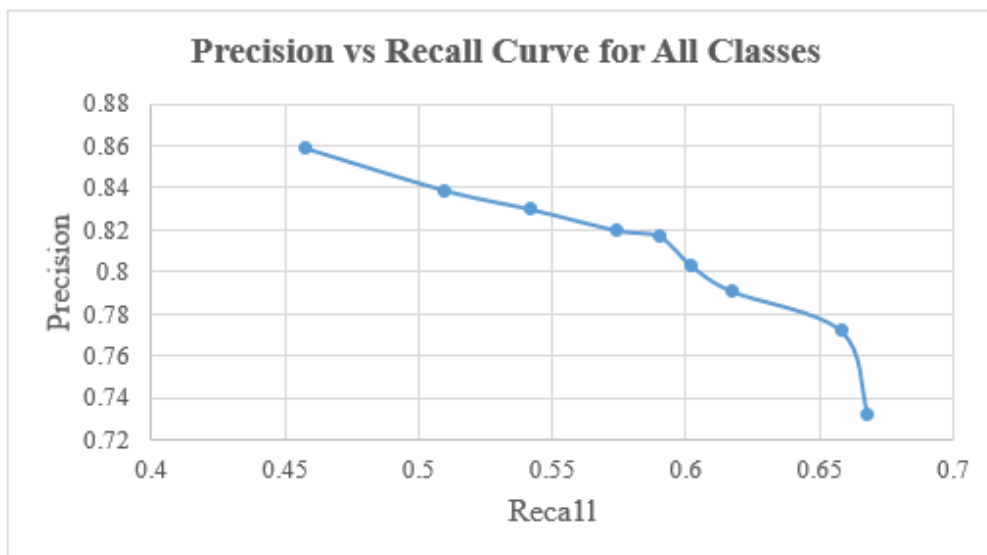


Figure 4.3: Precision vs Recall Curve for All Classes.

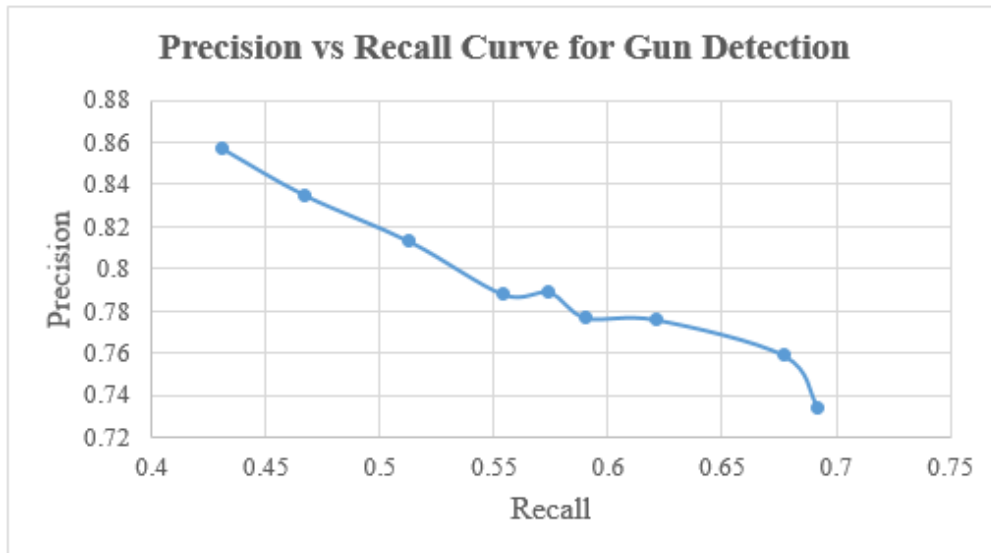


Figure 4.4: Precision vs Recall Curve for Gun Detection.

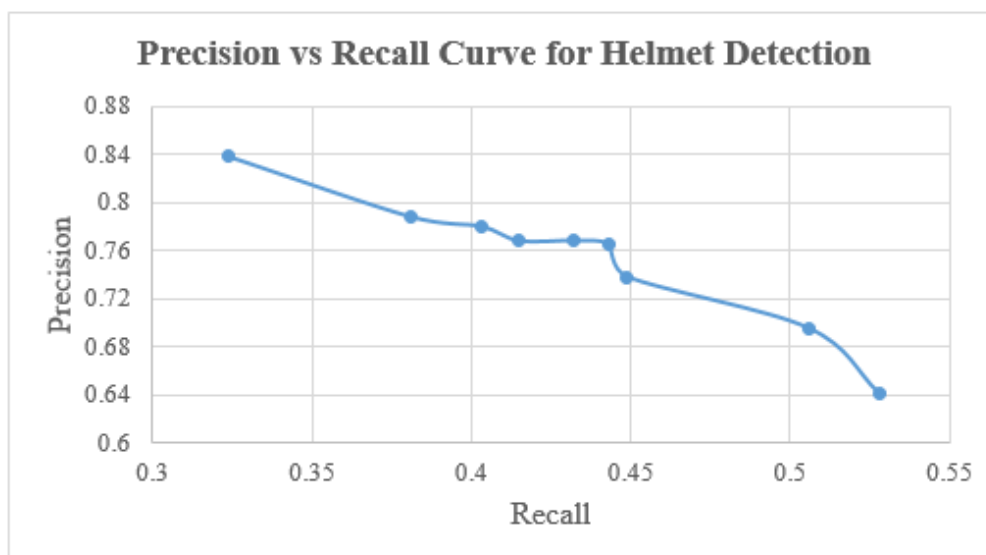


Figure 4.5: Precision vs Recall Curve for Helmet Detection.

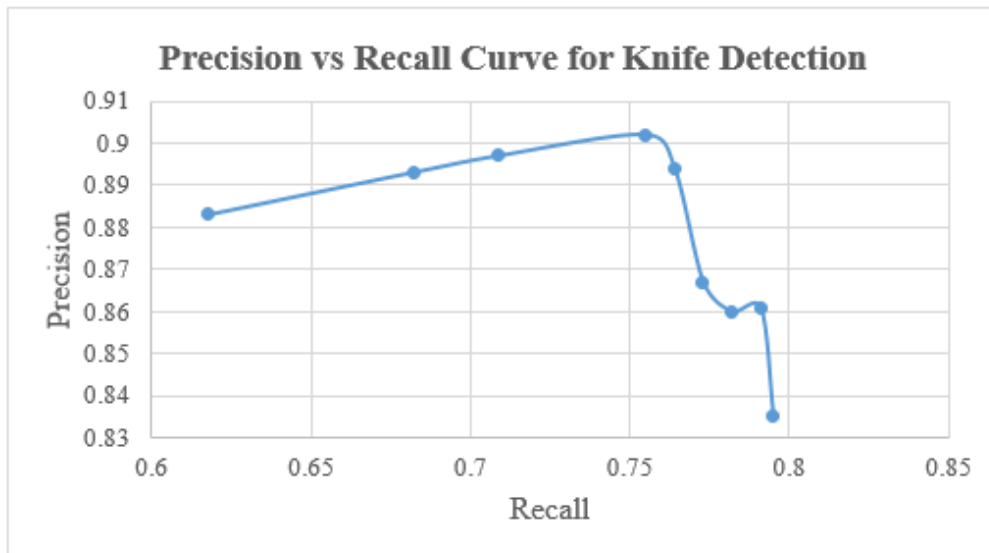


Figure 4.6: Precision vs Recall Curve for Knife Detection.

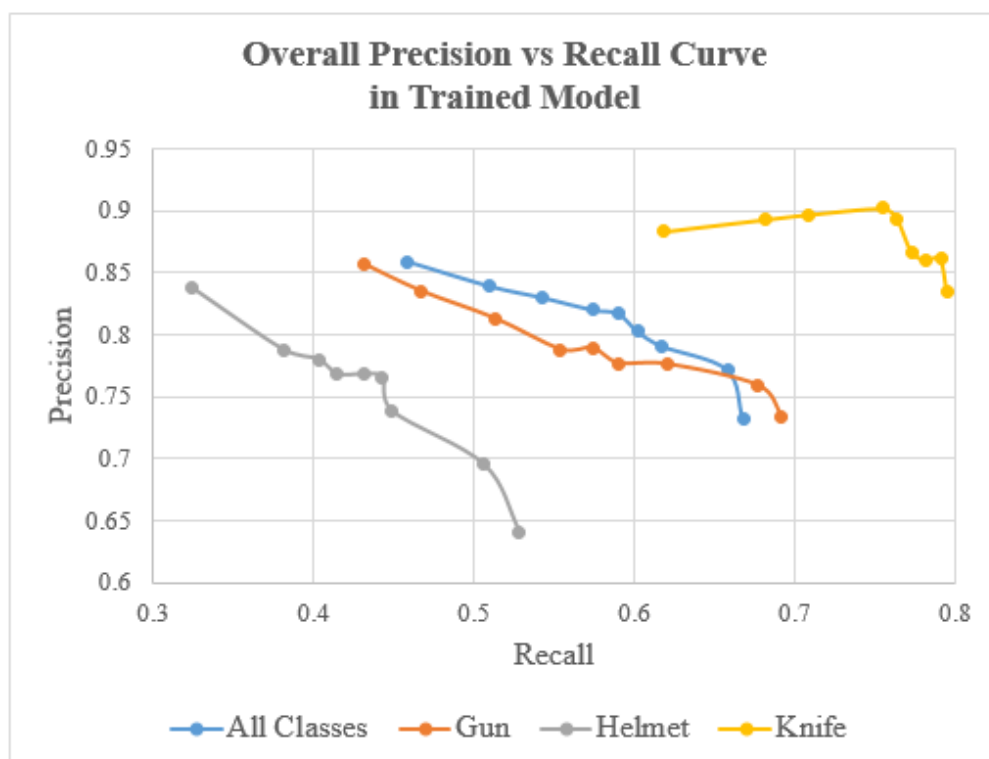


Figure 4.7: Overall Precision vs Recall Curve in Trained Model.

P and R are plotted using different confidence thresholds from 0.1 to 0.9 with the step size 0.1. From these figures, it can be observed that P is inversely proportional to R. The pattern of graphs is similar for each graph. Besides, P and R represents the accuracy of the model. Each bounding box contains its confidence level to give a rank of the output. According to Figure 4.7, knife detection has the highest P and R while helmet detection has the lowest P and R. This shows that accuracy in knife detection is higher than helmet detection. In short, the highest P and R obtained in this model are 0.859 and 0.668 respectively.

4.4 Evaluation Average Precision (AP), Mean Average Precision (mAP) and Intersection over Union Threshold (IoU) of the Model

In this section, the relationship between mAP or AP and IoU threshold is evaluated from the testing dataset and shown in Table 4.3. The mAP against IoU for all classes is shown in Figure 4.8. Meanwhile, Figure 4.9 displays AP against IoU for gun detection. According to Figure 4.10, AP against IoU for helmet detection is illustrated. Additionally, AP against IoU for knife detection is displayed in Figure 4.11. Additionally, overall mAP or AP against IoU in this trained model is shown in Figure 4.12.

Table 4.3: mAP or AP for Overall System, Gun, Helmet and Knife Detection in Different IoU thresholds.

Class	ALL	Gun	Helmet	Knife
IoU	mAP	AP	AP	AP
0.50	0.669	0.681	0.517	0.808
0.55	0.672	0.692	0.516	0.808
0.60	0.672	0.694	0.513	0.810
0.65	0.673	0.690	0.510	0.817
0.70	0.668	0.683	0.504	0.817
0.75	0.655	0.672	0.493	0.799
0.80	0.623	0.634	0.467	0.767
0.85	0.557	0.563	0.421	0.687
0.90	0.455	0.462	0.354	0.548
0.95	0.362	0.356	0.276	0.453

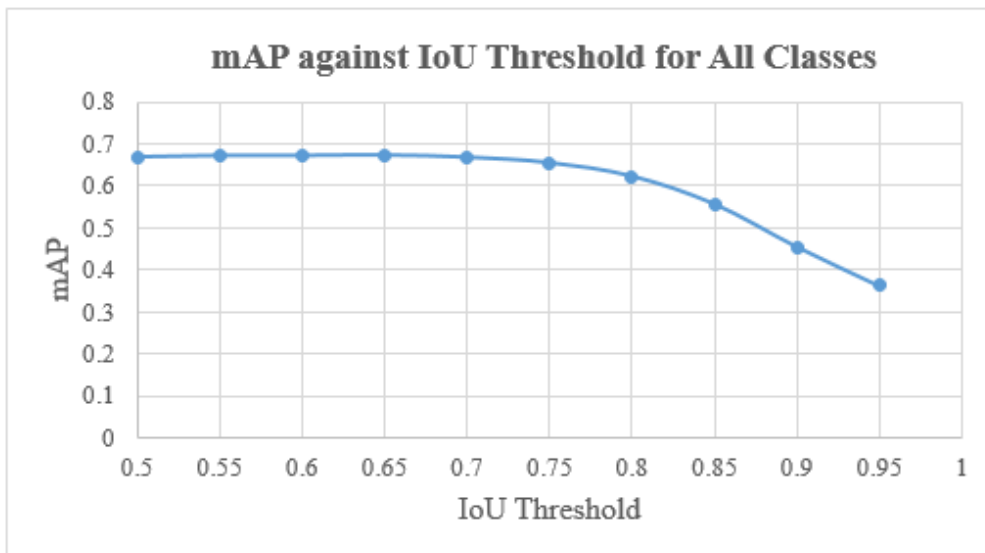


Figure 4.8: mAP against IoU Threshold for All Classes.

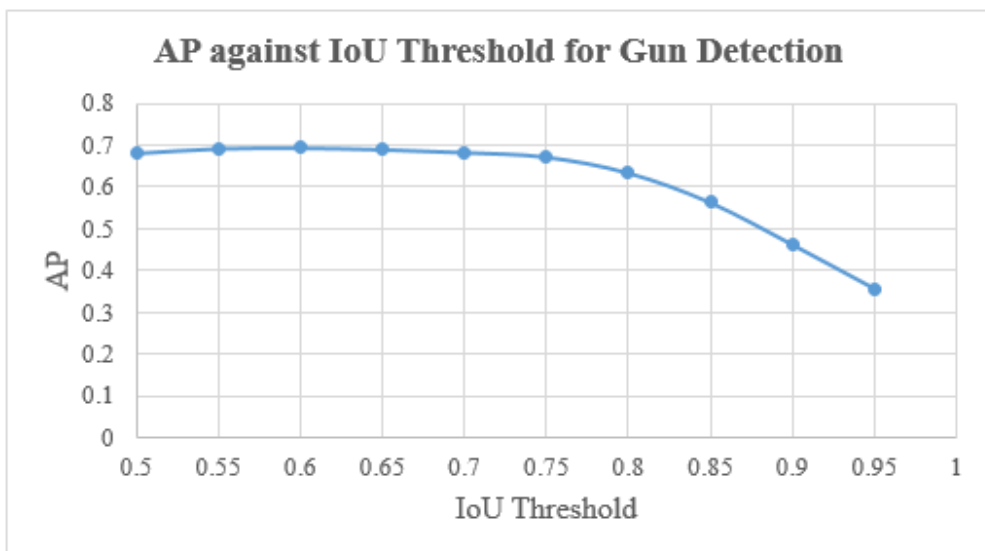


Figure 4.9: AP against IoU Threshold for Gun Detection.

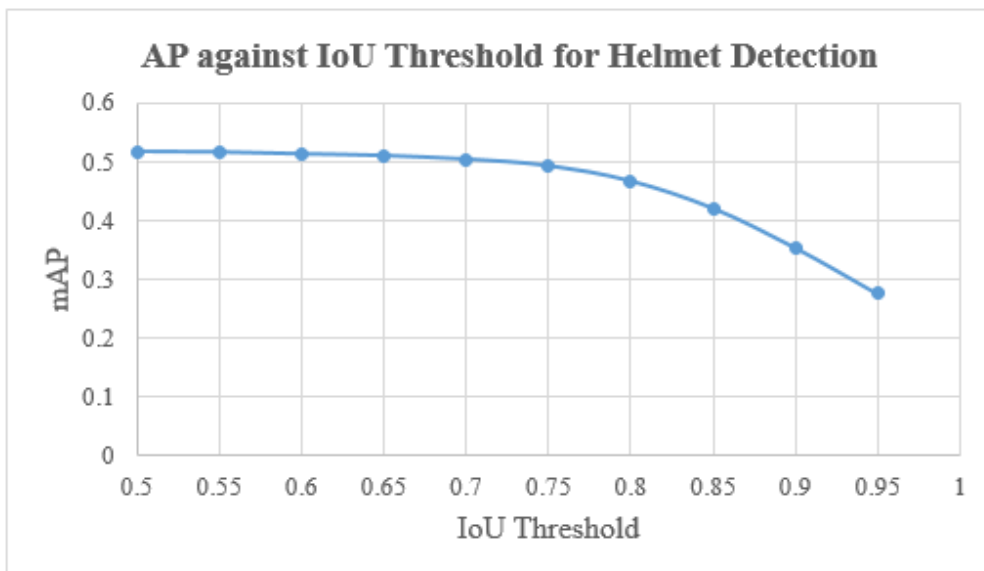


Figure 4.10: AP against IoU Threshold for Helmet Detection.

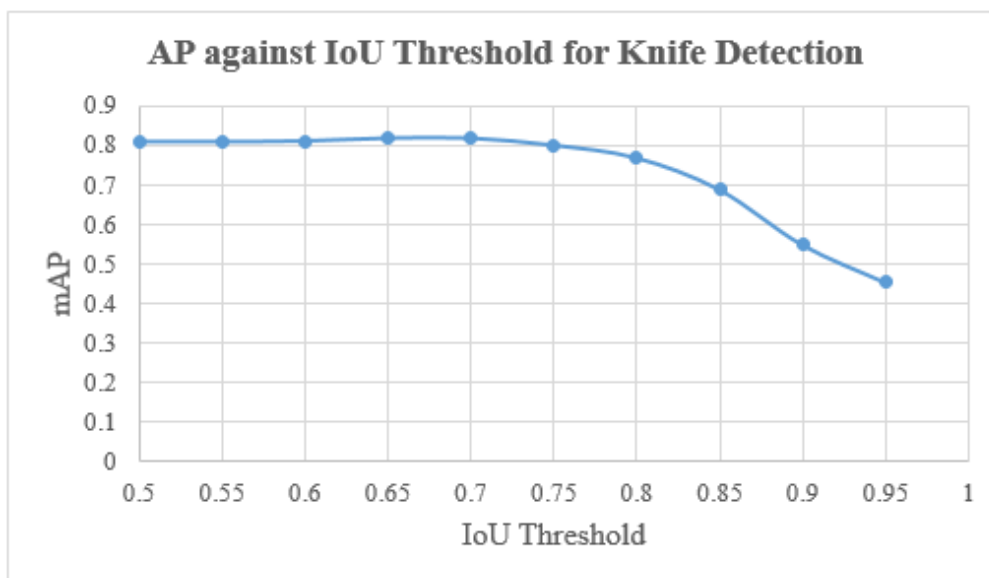


Figure 4.11: AP against IoU Threshold for Knife Detection.

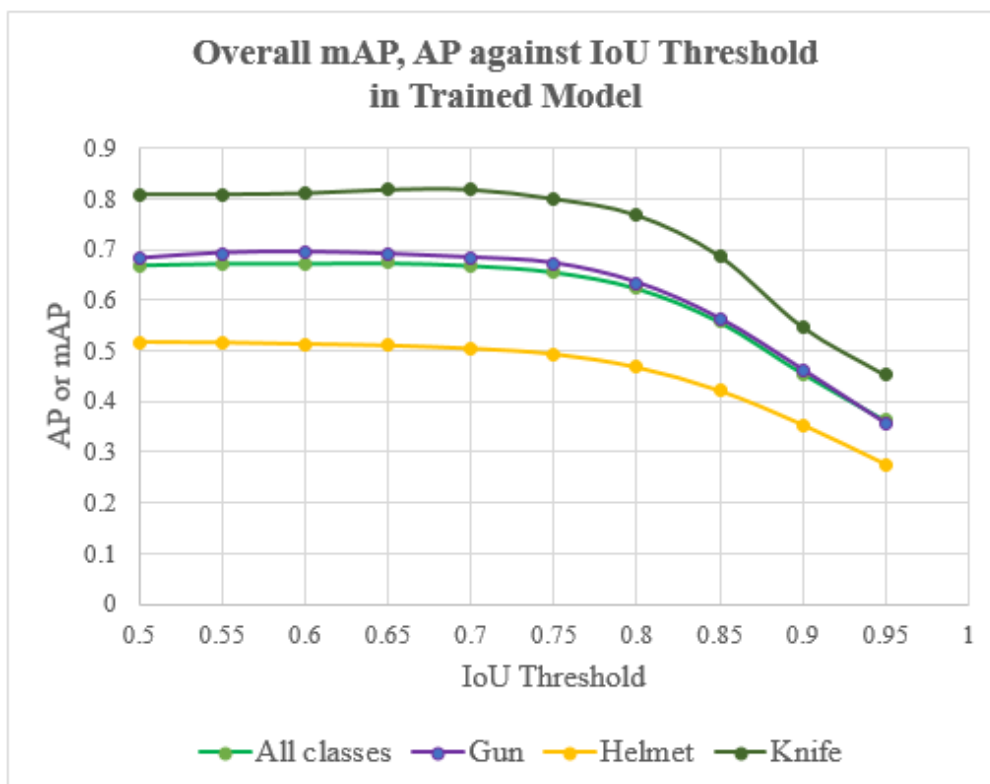


Figure 4.12: Overall mAP or AP against IoU Threshold in Trained Model.

IoU evaluates the accuracy of a detected object in a dataset. The concept and formula of IoU can be shown in Figure 4.13.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

Figure 4.13: IoU calculation (Rosebrock, 2016).

In short, AP is for each class while the mAP is average of AP. Next, AP and mAP are plotted with different IoU threshold from 0.5 to 0.95 by increasing 0.05. Based on these figures, AP and IoU threshold are inversely proportional.

The pattern of graphs is similar to each other. Moreover, Figure 4.12 indicates the AP of knife detection is highest while helmet detection is the lowest AP resulting in the accuracy of knife detection is high. The optimum IoU threshold in this model is 0.65 which the mAP of 67.3 %.

4.5 Crime Detections

The best sample crime detections for the gun, helmet, and knife are collected. Besides, Figure 4.14 shows a person is holding a gun and the gun is detected. Moreover, Figure 4.15 indicates a person with a helmet and the helmet is detected. Based on Figure 4.16, a knife is detected successfully.

The confidence score or accuracy of the gun, helmet, and knife detections 84.4 %, 99.97 %, and 95.32 % respectively. The detections prove that the model can detect the gun, helmet, and knife successfully.



Figure 4.14: Gun Detection



Figure 4.15: Helmet Detection

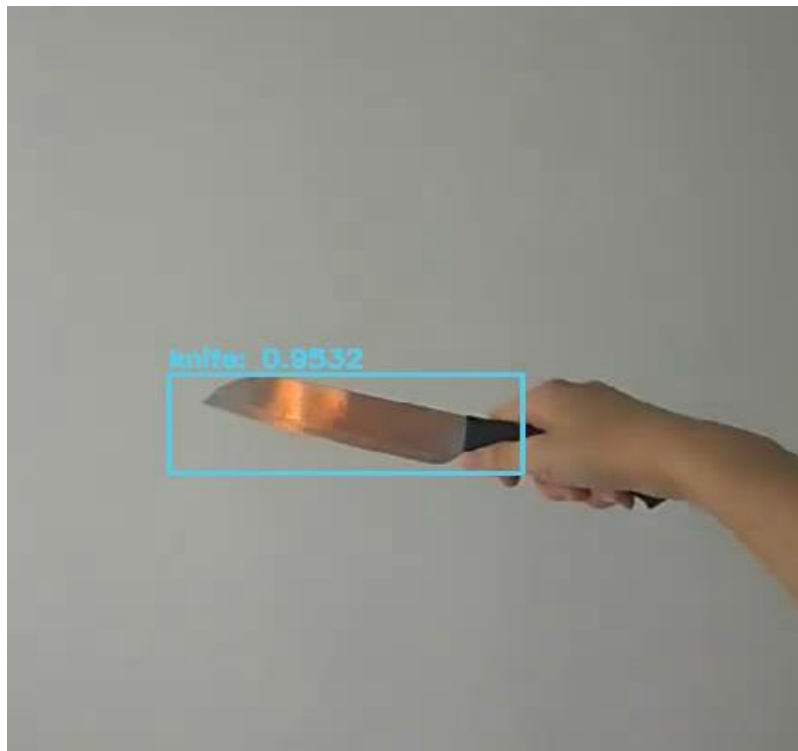


Figure 4.16: Knife Detection

4.6 Real-time and Input Video Performance

The real-time and input video performance for the testing frame per second (FPS) with different devices is recorded in Table 4.4. FPS also refers to the speed of detection.

The CPU used is Intel Core-i5. Besides, the FPS of the model is relatively low. Moreover, the FPS for the integration of CPU and NCS2 is higher as compared to the testing of CPU and a combination of RP3 with NCS2. However, RP3 is chosen because it is lightweight and affordable in price. Hence, the model requires further improvement for the speed of detection or FPS.

Table 4.4: Testing FPS with Different Devices in Input Video and Real-time.

Devices	CPU	CPU + NCS2	RP3 + NCS2
FPS	2.48	3.40	2.30

4.7 Summary

In a nutshell, the performance of the model is evaluated. The Precision vs Recall curve and graph of mAP against IoU threshold are plotted and analysed. Besides, P is inversely proportional to R. The highest P and R obtained in this model are 0.859 and 0.668 respectively. Moreover, mAP and IoU threshold are inversely proportional. The optimum IoU threshold in this model is 0.65 which the mAP for the overall system is 67.3 %. Furthermore, the gun, helmet, and knife are detected successfully. However, the FPS of the model is relatively low when testing on different devices in real-time or input video.

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

In short, the model is trained in Darknet architecture and then converted into IR files to support NCS2. The time for training and testing processes is needed in this project. Besides, the crime detection model is developed and integrated into an embedded security camera system. The system can be performed in real-time or input video.

The confidence score affects the results of P and R. P is inversely proportional to R. Moreover, the figures of mAP against IoU threshold show that the mAP and IoU threshold are inversely proportional. The mAP of the system is 67.3% with the 0.65 optimum IoU threshold. The gun, helmet, and knife are detected successfully. Although the FPS of the model in real-time performance is low, the accuracy of the model is satisfied. The system is unstable to perform in real-time.

5.2 Future Work

In order to process the video, the system should work well in real-time. According to the FPS results of the system, the FPS is quite low and less efficient in real-time performance. The FPS of the model is needed to be enhanced. Besides, the accuracy of the system is considered to be improved. The source of the dataset is limited because the current training dataset only has 3300 images. Therefore, the size of the training dataset can be increased to as much as possible. The collected images are increased for the training process to rise the accuracy for crime detection of the model. Future work is more concentrated to increase the detection speed and accuracy of the system so that it can work well in real-time performance.

The other suggestion is to integrate the system to the Internet of Things (IoT) so that it can send notifications for alerting owners. The system can be modified into the CCTV system in the future.

REFERENCES

Amato, G., Carrara, F., Falchi, F., Gennaro, C. and Vairo, C., 2016. Car parking occupancy detection using smart camera networks and Deep Learning. *Proceedings - IEEE Symposium on Computers and Communications*, 2016-Augus(D1), pp.1212–1217.

Dash, S. and Subudhi, B., 2016. *Handbook of research on computational intelligence applications in bioinformatics. Handbook of Research on Computational Intelligence Applications in Bioinformatics*. IGI Global.

Department of Statistics Malaysia, 2019. *Crime Statistics, 2019*. [online] Available at: <https://www.dosm.gov.my/v1/index.php/index.php?r=column/ctHEMEByCat&cat=455&bul_id=MEs4QzNxWkNZZDEyM08yM0Jsd05vQT09&menu_id=U3VPMldoYUxzVzFaYmNkWXZteGduZz09> [Accessed 23 Apr. 2020].

Eidinger, E., Enbar, R. and Hassner, T., 2014. Age and gender estimation of unfiltered faces. *IEEE Transactions on Information Forensics and Security*, 9(12), pp.2170–2179.

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep learning*. [online] *Healthcare Informatics Research*, Available at: <<https://books.google.com/books?hl=en&lr=&id=omivDQAAQBAJ&oi=fnd&pg=PR5&dq=Ian+Goodfellow+and+Yoshua+Bengio+and+Aaron+Courville&ots=MLU59rozNU&sig=YHYf6iAwhmFAQkBFuLnHAgkMts>> [Accessed 17 Aug. 2019].

Guresen, E. and Kayakutlu, G., 2011. Definition of Artificial Neural Networks with comparison to other networks. In: *Procedia Computer Science*. pp.426–433.

Haykin, S.S., 1999. *Neural networks : a comprehensive foundation*. Prentice Hall.

Intel, 2019. *Model Optimizer Concept | OpenVINO™ toolkit | Ep.08 | Intel Software - YouTube*. [online] Available at: <https://www.youtube.com/watch?v=Kl1ptVb7aI8&list=PLDKCjIU5YH6jMzcTV5_cxX9aPHsborbXQ&index=8> [Accessed 24 Apr. 2020].

Intel, 2020. *Deep Learning Inference | Intel® Distribution of OpenVINO™ Toolkit | Intel® Software*. [online] Available at: <<https://software.intel.com/en-us/openvino-toolkit/deep-learning-inference>> [Accessed 20 Apr. 2020].

Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2017. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, [online] 60(6), pp.84–90. Available at: <<http://dl.acm.org/citation.cfm?doid=3098997.3065386>> [Accessed 18 Aug. 2019].

Lecun, Y., Bengio, Y. and Hinton, G., 2015. Deep learning. *Nature*, 521(7553), pp.436–444.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. and Jackel, L.D., 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation*, 1(4), pp.541–551.

Li, J., Gu, J., Huang, Z. and Wen, J., 2019. Application research of improved YOLO V3 algorithm in PCB electronic component detection. *Applied Sciences (Switzerland)*, 9(18).

Li, Y., Wu, B. and Nevatia, R., 2008. Human detection by searching in 3d space using camera and scene knowledge. *Proceedings - International Conference on Pattern Recognition*.

Lovelace, A.K., 1842. *Sketch of the analytical engine invented by Charles Babbage*.

McCarthy, J., 2007. What Is Artificial Intelligence? *American Scientist*, 73(3), p.258.

Norouzzadeh, M.S., Nguyen, A., Kosmala, M., Swanson, A., Palmer, M.S., Packer, C. and Clune, J., 2018. Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning. *Proceedings of the National Academy of Sciences of the United States of America*, 115(25), pp.E5716–E5725.

R. Hirschmann, 2020. *Singapore: Crime Rate 2019*. [online] Statista. Available at: <<https://www.statista.com/statistics/628339/crime-rates-in-singapore/>> [Accessed 23 Apr. 2020].

Redmon, J. and Farhadi, A., 2018. YOLOv3: An Incremental Improvement. [online] Available at: <<http://arxiv.org/abs/1804.02767>>.

Rosebrock, A., 2016. *Intersection over Union (IoU) for object detection - PyImageSearch*. [online] www.pyimagesearch.com. Available at: <<https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/>> [Accessed 24 Apr. 2020].

Shakil, S., Rajjak, A. and Kureshi, A.K., 2020. Object Detection and Tracking using YOLO v3 Framework for Increased Resolution Video. (6), pp.118–125.

Wani, M.A., Bhat, F.A., Afzal, S. and Khan, A.I., 2020. *Advances in Deep Learning*. [online] Available at: <<http://link.springer.com/10.1007/978-981-13-6794-6>> [Accessed 18 Aug. 2019].

Willi, M., Pitman, R.T., Cardoso, A.W., Locke, C., Swanson, A., Boyer, A., Veldhuis, M. and Fortson, L., 2019. Identifying animal species in camera trap images using deep learning and citizen science. *Methods in Ecology and Evolution*, 10(1), pp.80–91.

APPENDICES

APPENDIX A: Python Script

```

# *****
# Copyright(c) 2018 Intel Corporation.
# License: MIT See LICENSE file in root directory.
# *****

# import necessary packages
from imutils.video import VideoStream
from imutils.video import FPS
import sys, os, cv2, time
import numpy as np, math
from argparse import ArgumentParser

# for Raspbian OS
try:
    from armv7l.opencv_inference_engine import IENetwork, IEPlugin
except:
    from opencv_inference_engine import IENetwork, IEPlugin

# input size of model
m_input_size = 416

# network of YOLO layer
yolo_scale_13 = 13
yolo_scale_26 = 26
yolo_scale_52 = 52

# initialization
classes = 3 # 3 classes in this model
coords = 4
num = 3
anchors = [10,13,16,30,33,23,30,61,62,45,59,119,116,90,156,198,373,326]

# initialize the list of class
LABELS = ("gun", "helmet", "knife")

# bounding box and label color
label_text_color = (255, 255, 255)
label_background_color = (125, 175, 75)
box_color = (255, 128, 0)
box_thickness = 1

# construct the argument parse and parse the arguments
# load plugin for Inference Engine
def build_argparser():
    parser = ArgumentParser()
    parser.add_argument("-d", "--device", help="Specify the target device to infer on; CPU, GPU, FPGA or MYRIAD is acceptable. \
        Sample will look for a suitable plugin for device specified (CPU by default)", default="CPU", type=str)
    return parser

def EntryIndex(side, lcoords, lclasses, location, entry):
    n = int(location / (side * side))
    loc = location % (side * side)
    return int(n * side * side * (lcoords + lclasses + 1) + entry * side * side + loc)

```

```

# Confidence
class DetectionObject():
    xmin = 0
    ymin = 0
    xmax = 0
    ymax = 0
    class_id = 0
    confidence = 0.0

    def __init__(self, x, y, h, w, class_id, confidence, h_scale, w_scale):
        self.xmin = int((x - w / 2) * w_scale)
        self.ymin = int((y - h / 2) * h_scale)
        self.xmax = int(self.xmin + w * w_scale)
        self.ymax = int(self.ymin + h * h_scale)
        self.class_id = class_id
        self.confidence = confidence

#Load IoU threshold function
def IntersectionOverUnion(box_1, box_2):
    width_of_overlap_area = min(box_1.xmax, box_2.xmax) - max(box_1.xmin,
box_2.xmin)
    height_of_overlap_area = min(box_1.ymax, box_2.ymax) - max(box_1.ymin,
box_2.ymin)
    area_of_overlap = 0.0
    if (width_of_overlap_area < 0.0 or height_of_overlap_area < 0.0):
        area_of_overlap = 0.0
    else:
        area_of_overlap = width_of_overlap_area * height_of_overlap_area
    box_1_area = (box_1.ymax - box_1.ymin) * (box_1.xmax - box_1.xmin)
    box_2_area = (box_2.ymax - box_2.ymin) * (box_2.xmax - box_2.xmin)
    area_of_union = box_1_area + box_2_area - area_of_overlap
    retval = 0.0
    if area_of_union <= 0.0:
        retval = 0.0
    else:
        retval = (area_of_overlap / area_of_union)
    return retval

#----Parsing for YOLO v3 Output
def ParseYOLOV3Output(blob, resized_im_h, resized_im_w, original_im_h,
original_im_w, threshold, objects):

    out_blob_h = blob.shape[2]
    out_blob_w = blob.shape[3]

    side = out_blob_h
    anchor_offset = 0

#----Extracting layer parameters
if len(anchors) == 18: ## YoloV3
    if side == yolo_scale_13:
        anchor_offset = 2 * 6
    elif side == yolo_scale_26:
        anchor_offset = 2 * 3
    elif side == yolo_scale_52:
        anchor_offset = 2 * 0

```



```

else:
    if side == yolo_scale_13:
        anchor_offset = 2 * 6
    elif side == yolo_scale_26:
        anchor_offset = 2 * 3
    elif side == yolo_scale_52:
        anchor_offset = 2 * 0

side_square = side * side
output_blob = blob.flatten()

#-----Parsing YOLO Region Output-----#

for i in range(side_square):
    row = int(i / side)
    col = int(i % side)
    for n in range(num):
        obj_index = EntryIndex(side, coords, classes, n * side * side + i, coords)
        box_index = EntryIndex(side, coords, classes, n * side * side + i, 0)
        scale = output_blob[obj_index]
        if (scale < threshold):
            continue
        x = (col + output_blob[box_index + 0 * side_square]) / side * resized_im_w
        y = (row + output_blob[box_index + 1 * side_square]) / side * resized_im_h
        height = math.exp(output_blob[box_index + 3 * side_square]) *
anchors[anchor_offset + 2 * n + 1]
        width = math.exp(output_blob[box_index + 2 * side_square]) *
anchors[anchor_offset + 2 * n]
        for j in range(classes):
            class_index = EntryIndex(side, coords, classes, n * side_square + i,
coords + 1 + j)
            prob = scale * output_blob[class_index]
            if prob < threshold:
                continue
            obj = DetectionObject(x, y, height, width, j, prob, (original_im_h /
resized_im_h), (original_im_w / resized_im_w))
            objects.append(obj)
    return objects

# Inference Engine main program
def main_IE_infer():
#initiate the camera setting
    camera_width = 320
    camera_height = 240
    fps = ""
    framepos = 0
    frame_count = 0
    vidfps = 0
    skip_frame = 0
    elapsedTime = 0
    new_w = int(camera_width * m_input_size/camera_width)
    new_h = int(camera_height * m_input_size/camera_height)

```

```

#----load IR files, .xml and .bin file
args = build_argparser().parse_args()
model_xml = "/home/pi/Desktop/FYP2-v3/obj_detection.xml"
model_bin = os.path.splitext(model_xml)[0] + ".bin"

#----this is for input camera (real-time)
# initialize the video stream, allow the camera sensor to warmup,
# and initialize the FPS counter
cap = cv2.VideoCapture(0)
cap.set(cv2.CAP_PROP_FPS, 30)
cap.set(cv2.CAP_PROP_FRAME_WIDTH, camera_width)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, camera_height)

#----this is for input video
#cap = cv2.VideoCapture("data/input/testvideo.mp4")
#camera_width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
#camera_height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
#frame_count = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
#vidfps = int(cap.get(cv2.CAP_PROP_FPS))
#print("videosFrameCount =", str(frame_count))
#print("videosFPS =", str(vidfps))

time.sleep(1)

#----load plugin to Inference Engine
plugin = IEPlugin(device=args.device)
if "CPU" in args.device:
    plugin.add_cpu_extension("lib/libcpu_extension.so")
net = IENetwork(model=model_xml, weights=model_bin)
input_blob = next(iter(net.inputs))
exec_net = plugin.load(network=net)

while cap.isOpened():
    t1 = time.time()

    ## Uncomment only when playing video files
    #cap.set(cv2.CAP_PROP_POS_FRAMES, framepos)

    #read frame
    ret, image = cap.read()
    if not ret:
        break

#----Configuring input and output
#resize frame
resized_image = cv2.resize(image, (new_w, new_h), interpolation =
cv2.INTER_CUBIC)
canvas = np.full((m_input_size, m_input_size, 3), 128)
canvas[(m_input_size-new_h)//2:(m_input_size-new_h)//2 +
new_h,(m_input_size-new_w)//2:(m_input_size-new_w)//2 + new_w, :] =
resized_image
prepimg = canvas
prepimg = prepimg[np.newaxis, :, :, :] # Batch size axis add
prepimg = prepimg.transpose((0, 3, 1, 2)) # NHWC to NCHW
outputs = exec_net.infer(inputs={input_blob: prepimg})

objects = []

```

```

for output in outputs.values():
    objects = ParseYOLOV3Output(output, new_h, new_w, camera_height,
camera_width, 0.7, objects)

# Filtering overlapping boxes
objlen = len(objects)
for i in range(objlen):
    if (objects[i].confidence == 0.0):
        continue
    for j in range(i + 1, objlen):
        if (IntersectionOverUnion(objects[i], objects[j]) >= 0.4):
            objects[j].confidence = 0

# Drawing bounding boxes for object detection
for obj in objects:
    if obj.confidence < 0.2:
        continue
    label = obj.class_id
    confidence = obj.confidence
    if confidence > 0.2:
        label_text = LABELS[label] + " (" + "{:.1f}".format(confidence * 100) +
"%)"
        cv2.rectangle(image, (obj.xmin, obj.ymin), (obj.xmax, obj.ymax),
box_color, box_thickness)
        cv2.putText(image, label_text, (obj.xmin, obj.ymin - 5),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, label_text_color, 1)

        cv2.putText(image, fps, (camera_width - 170, 15),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (38, 0, 255), 1, cv2.LINE_AA)
        #show output
        cv2.imshow("Result", image)

#---shutdown program
# press 'q' to exit
if cv2.waitKey(1)&0xFF == ord('q'):
    break
elapsedTime = time.time() - t1
#show FPS value
fps = "(Playback) {:.1f} FPS".format(1/elapsedTime)

## frame skip, video file only
#skip_frame = int((vidfps - int(1/elapsedTime)) / int(1/elapsedTime))
#framepos += skip_frame

# ---Close window
cv2.destroyAllWindows()
del net
del exec_net
del plugin

if __name__ == '__main__':
    sys.exit(main_IE_infer() or 0)

```