Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino

BY

Chiew Jing Cheng

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

JAN 2023

UNIVERSITI TUNKU ABDUL RAHMAN

REPORT STATUS DECLARATION FORM

Title: **Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino**

Academic Session: JAN2023

I CHIEW JING CHENG

(CAPITAL LETTER)

declare that I allow this Final Year Project Report to be kept in

Universiti Tunku Abdul Rahman Library subject to the regulations as follows:

1. The dissertation is a property of the Library.
2. The Library is allowed to make copies of this dissertation for academic purposes.

Verified by,

[illegible]

(Author's signature)

[illegible]

(Supervisor's signature)

Address:

87 Jalan Taman Melati 6,

Taman Melati, Setapak, 53100,

Kuala Lumpur

Ts. Wong Chee Siang

Supervisor's name

Date: 23 APRIL 2023

Date: 25 APRIL 2023

Universiti Tunku Abdul Rahman			
Form Title : **Sample of Submission Sheet for FYP/Dissertation/Thesis**			
Form Number: **FM-IAD-004**	Rev No.: **0**	Effective Date: **21 JUNE 2011**	Page No.: **1 of 1**

FACULTY/INSTITUTE* OF _INFORMATION AND COMMUNICATION TECHNOLOGY_________

UNIVERSITI TUNKU ABDUL RAHMAN

Date: ___24/4/2023___

SUBMISSION OF FINAL YEAR PROJECT /DISSERTATION/THESIS

It is hereby certified that ______***Chiew Jing Cheng***__________ (ID No: _***20ACB02331***)has completed this final year project/ dissertation/ thesis* entitled " Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino " under the supervision of __Ts_Wong Chee Siang_ (Supervisor) from the Department of ___Computer Science_____, Faculty/Institute* of _Information and Communication Technology.

I understand that University will upload softcopy of my final year project / dissertation/ thesis* in pdf format into UTAR Institutional Repository, which may be made accessible to UTAR community and public.

Yours truly,

[illegible]

____

(Chiew Jing Cheng)

DECLARATION OF ORIGINALITY

I declare that this report entitled "**Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino**" is my own work except as cited in the references. The report has not been accepted for any degree and is not being submitted concurrently in candidature for any degree or other award.

Signature	:	[illegible]
Name	:	Chiew Jing Cheng
Date	:	23 April 2023

ACKNOWLEDGEMENTS

I want to sincerely thank and appreciate my supervisor, Ts Wong Chee Siang, for providing me with this inspiring opportunity to engage on a deep learning project with OpenVino Technology. Many thanks in advance. I want to express my gratitude to my parents and my entire family for their support, love, and never-ending encouragement during the journey.

ABSTRACT

Deep learning technologies are becoming increasingly popular in recent years. Numerous industries, including healthcare, entertainment, automation systems, natural language processing, and others, are impacted by it. It advances global technology to a new level. Deep learning techniques are now widely employed as a result, especially on edge devices that perform IoT tasks. It is because we no longer need people to assist us in our work, we instead choose to deploy an edge device with a deep learning model. To run the code effectively without being bothered by the slow processing times, those deep learning approaches demand for a lot of processing power, which requires strong computer hardware. This project interprets and demonstrates how OpenVino (Open Visual Inference and Neural) toolkits assist in improving performance and enable us to run a demanding deep learning model on an Intel's computer system that most regular people have. The OpenVino's inference engine is designed to speed up the inference of deep learning models under IR format that is provided by OpenVino. This project will explain whether the OpenVino toolkit does indeed offer a shorter inference time and eventually how much performance can be delivered. Before the project ends, a traffic violation detection application that combined with several deep learning pre-trained models will be configured and deployed in an intel-powered edge device (Intel UP board). It aims to determine whether running an OpenVino-optimized deep learning framework application on an edge device with a low-power processor can surprisingly produce a respectable result.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

OpenVino	Open Visual Inference and Neural Network Optimization
IR	Intermediate Representation
AI	Artificial Intelligence
DL	Deep Learning
DNN	Deep Neural Network
SSD	Single Shot Detector
FPS	Frames Per Second
TF	TensorFlow
IE	Inference Engine
OCR	Optical Character Recognition
AVX	Advanced Vector Extensions

CHAPTER 1 Introduction

1.1 Project Background

Deep learning is widely used by businesses on an edge device nowadays. Individuals have expressed concern over about the amount of computing power needed to any DNNs model predicts output. In this project, we will talk about the OpenVino toolkit, it is a toolkit that Intel has made available to speed up conversion of the original deep learning framework (TensorFlow, pytorch, ONNX, …) to the OpenVino (IR) format. (Intermediate Representation). The development process can be accelerated by using OpenVino Toolkit to simplify deep learning models by converting them to conform to IR format as well as optimising DNN models to reduce the time it takes for inference.

Correspondingly, there are traffic offenses, particularly in one-way lanes, in residential, academic, and urban areas. Many people complain that the number of incidents occurring in those areas keeps rising. Even when there isn't an accident, people still strongly dislike and criticise of the actions of other drivers because they force them to drive carefully and with all of their attention every time. Hence, this project will then configure a traffic violation detection system with paired with several pre-trained deep learning models on an Intel edge device. In order to enhance the DNNs' performance on the edge device, this project will then deploy system along with OpenVino technology. The edge device will be installed at the side of the road to detect and save those records of drivers who violate traffic laws in one-way lanes,

As shown in [1], The OpenVino project workflow can be separated to 4 part which is Train, Model optimizer, Inference Engine and Deployment. First, we have to train a model with code, then we use OpenVino toolkit's model optimizer to optimize the model and generate an Intermediate Representation (.xml + .bin files) of the model which can be inferred with OpenVino Runtime. Furthermore, the inference engine's job is to check for model compatibility based on the framework used to train the model as well as the hardware used (Intel hardware). Lastly, the application is deployed to edge devices. Figure 1.1 shows the OpenVino toolkit workflow.

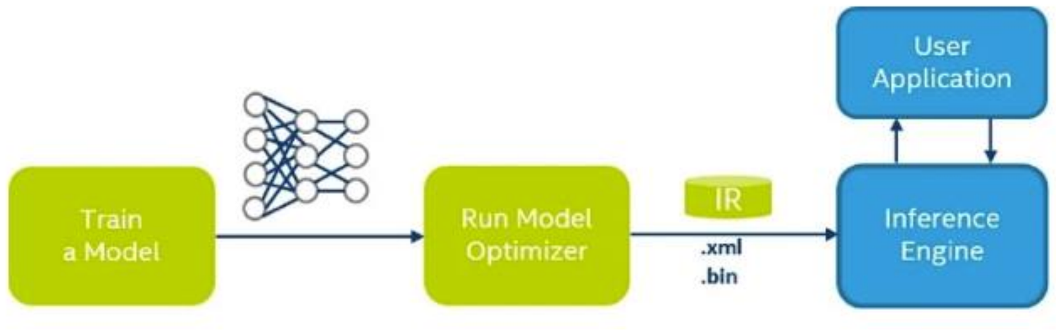


Figure 1.1 OpenVino Implementation workflow.

1.2 Problem Statement

Recently, Deep Neural Networks (DNNs) have made significant advancements in a variety of fields. These powerful deep learning models are capable of imitating human behaviour and autonomous decision-making. Therefore, it could displace the majority of common human jobs. To deploy and produce such good performance and usable deep learning models on an edge device, indeed, requires significant hardware as well as DNN models and frameworks. It might require powerful hardware as well as pricey processing resources.

Additionally, particularly in urban and residential areas, serious traffic offences like driving the wrong way are frequent. Many people use a one-way lane to travel in the opposite direction in an effort to get to their destination faster, but this frequently leads to accidents. Even though there are no accidents, people still detest having to keep their attention focused all the time on those annoying drivers.

1.3 Motivation

The project's goal was to implement a deep learning inference system and a traffic violation detection system using the OpenVino toolkit. In this paper, we will examine how much the OpenVino toolkit can optimise or speed up the system's performance. And furthermore, a traffic violation detection application will be developed at the project's ending using deep learning and OpenVino technology. The application will then be installed on an edge device with an Intel processor with low power (Intel UP board). This last is to see whether an OpenVino-optimized deep learning framework application running on a low-power edge device can deliver an acceptable result.

1.4 Project Scope

The scope of the project is to evaluate whether the OpenVino Toolkit significantly increases DNN performance before deploying traffic violation detection system on an edge device. In this project, we firstly compare some pre-trained deep learning models from various frameworks along with OpenVino toolkits (by using supported Intel hardware) to determine if it really brings out performance enhancement (lowering the inference time needed). Then, we configure traffic violation detection system with the combination of several deep learning models that are supported by OpenVino. At last, the system is then deployed on an Intel edge device after OpenVino has been implemented to improve system performance.

1.5 Project Objectives

The main objective of the project is to implement a traffic violation detection system on an edge device using a set of deep learning models as well as the OpenVino Toolkit. On the edge device, the OpenVino Toolkit is used to enhance performance by decreasing the DNNs' inference time. The main objective can be divided into the following two sub-objectives:

1. To measure the performance enhancement introduced by OpenVino and select the most effective DL framework.
2. Configure Deploy the traffic violation detection on edge device along with OpenVino Toolkits.

After the project is finished, a traffic violation detection system will be successfully deployed on the edge device and can be used readily in a one-way road to detect if any vehicles are moving in the wrong direction, thereby reducing the number of accidents. This project can also draw conclusions about whether OpenVino actually improves performance and whether doing so on an Intel edge device for a deep learning project is worthwhile.

1.6 Impact, Significance, Contribution

A complex computer vision deep learning application, especially that forms with multiple DNNs models can be thrilling to a CPU. CPU will reach a bottleneck If it unable to manage to handle those hard cores of DNNs inference tasks. Additionally, people are more willing to buy affordable edge devices that powered by only low-power CPU to perform the computer vision task for saving cost. Thus, the solution of the project can conclude that if OpenVino really help us for saving cost through accelerate DNNs performance/inference time.

Besides that, traffic offences like driving the wrong direction in one-way lane are a concern today, particularly in urban areas like residential neighbourhoods and school campuses. The deployed traffic violation system application can assist in identifying and capturing any vehicles that are moving against traffic laws (driving in the opposite direction). As such result, management is able to punish rulebreakers. In the end, this will lower the accident rate in urban areas.

CHAPTER 2 Literature Review

2.1 Previous Works on assessing OpenVino Toolkit

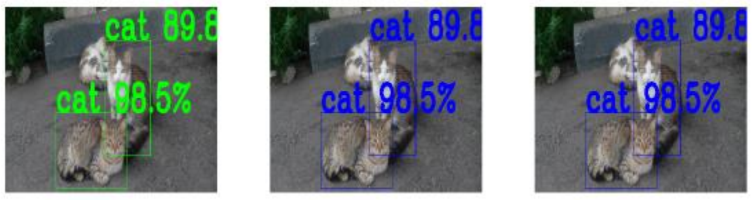
2.1.1 Analysis of the Acceleration of Neural Networks Inference on Intel Processors Based on OpenVino Toolkit

In [2] research, the SSD MobileNet V2 COCO model-based TensorFlow and OpenVino Inference Engine are used to infer neural network models for the tasks of object detection in pictures. It is demonstrated that a neural network's performance may be greatly increased by reconfiguring the network for use on Intel processors using the OpenVino Inference Engine. TensorFlow only is used in the comparison as the network implementation.

The COCO 2017 dataset and 90 categories of objects with 200 images were used to train the object detection model, which was then used to evaluate from[2]. Mobilev2 SSD Deep Neural Networks are one that are utilised in this study. The Intel i5-4460 processor, which is only supported by Intel hardware, was used.

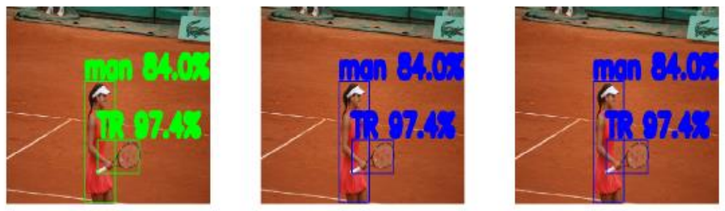
This literature review [2] has drawn out two conclusions. First, using the OpenVino inference engine has proven that the output of a neural network using OpenVino is equivalent to the output of a traditional TensorFlow network implemented without OpenVino. Second, the network that applied with OpenVino toolkit improves performance by hundreds of frames per second compared to traditional network implementations using TensorFlow. The example result is shown in Figure 2.1 – 2.3.

In table 2.1, we also can see that the OpenVino Inference engine gain average performance of 126.449 times compared to traditional TensorFlow neural network.



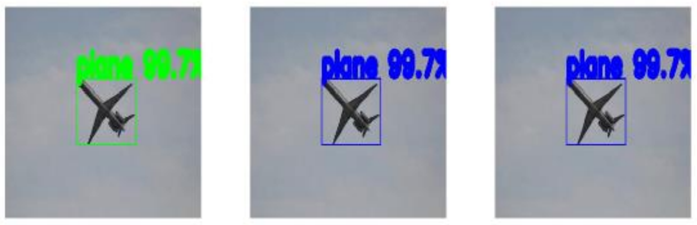
(a) TensorFlow 0.147(FPS) (b) Inference Engine 17.735(FPS) (c) TensorFlow and Inference Engine predictions are identical

Figure 2.1 detection of object labelled as “cat”



(a) TensorFlow 0.151(FPS) (b) Inference Engine 17.495(FPS) (c) TensorFlow and Inference Engine predictions are identical

Figure 2.2 Detection of objects labelled as “Man” (person) and “TR” (tennis racket)



(a) TensorFlow 0.147(FPS) (b) Inference Engine 15.005(FPS) (c) TensorFlow and Inference Engine predictions are identical

Figure. 2.3 Detection of object labelled as “Plane”

Table 2.1 Statistical characteristic of performance (PF)

	Average Processed Frame per Second (FPS)
TensorFlow	0.149 fps
OpenVino	18.841 fps
OpenVino Gain	126.449x faster

2.1.2 Philips Healthcare Uses the OpenVino Toolkit and the intel DevCloud for the edge to accelerate compressed Sensing Image reconstruction algorithm for MRI.

In research [3], In order to shorten scan times, Philips Healthcare included compressed sensing to their magnetic resonance imaging (MRI) scanners. Philip uses deep learning techniques to recreate the MRI image. According to research [3], Philips Healthcare was able to use the OpenVino toolset to accelerate their deep learning inference.

The relative speedups of the OpenVino toolkit over TensorFlow that is not optimised are illustrated in Figure 2.4. W-net inference and Adaptive-CS-Net Inference are two deep neural network models that were employed in this project from the TensorFlow framework. The testing on three separate Intel processors—Intel Core i7-8665UE, Intel Xeon E3 1268L v5, and Intel Xeon gold 6138—resulted in these results.

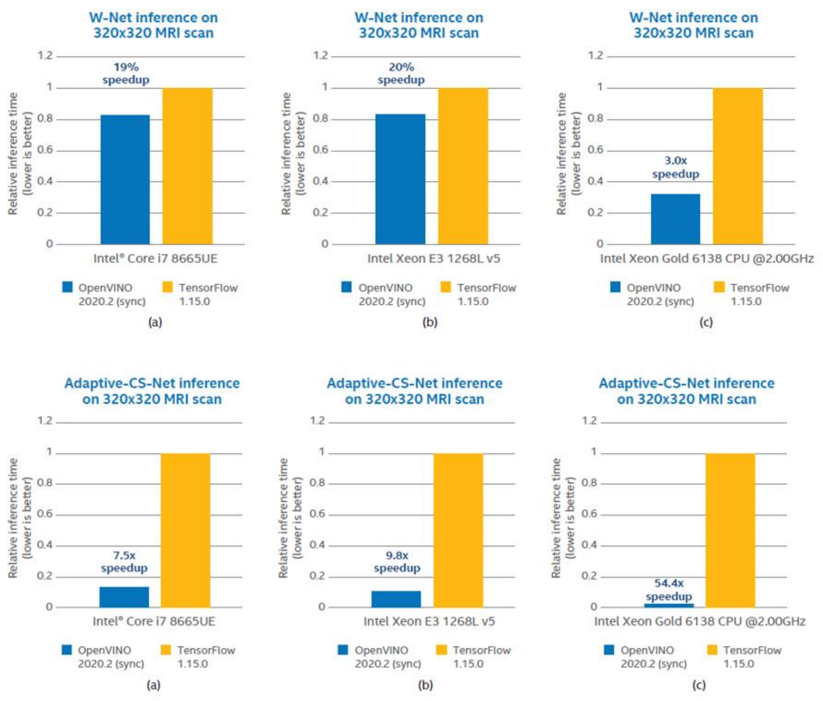


Figure 2.4 comparing unoptimized TensorFlow, versus OpenVino inference network

As we can see from [3], compared to unoptimized TensorFlow, OpenVino toolkits were able to accelerate the compressed sensing workloads for Philip Health care by maximum up to 54x and minimum on Adapticve-CS-Net and minimum with19%

improvement on W-Net inferenet. We can also conclude that W-Net seems to have the better optimization in OpenVino toolkit.

2.1.3 Lung Nodule Detection from low dose CT scan using Optimization on Intel Xeon and Core processors with Intel Distribution of OpenVino Toolkit

They aim to develop a system that can detect lung nodules using deep learning in paper [4]. They ran inference performance tests on systems with Intel processors using an optimised OpenVino Deep Learning model against an unoptimized BVLC Caffe framework, and they got excellent outcomes, which are detailed in the Result section.

Figures 2.5, It was observed that the average inference time for each image using the Intel Distribution of OpenVino optimised model was only 0.2304 seconds on an Intel Core i7 machine as opposed to 7.5 seconds when using the BVLC Caffe DetectNet model, achieving a 33x performance improvement over the baseline model. Additionally, it certainly improves by roughly 31 times in the i5 processor, which produces an average inference time of 0.3455 seconds as opposed to 11 seconds. As a result, we may draw the conclusion that using the OpenVino toolset will enable us to enhance the performance in the BVLC Caffe model by 30 times.

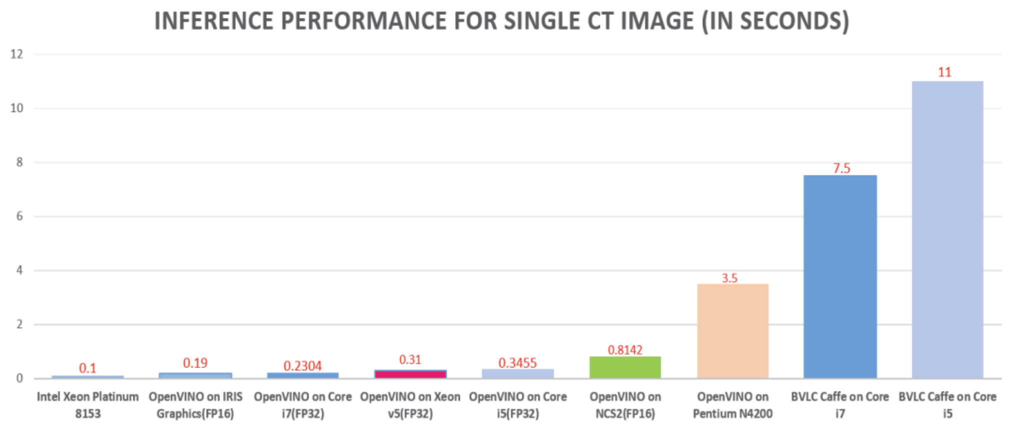


Figure 2.5 performance chart for inference time comparison

2.1.4 Face Detection and Face Re-identification System Using Deep Learning and OpenVino

The system proposed in [5] was built with Face detection and face re-identification to identify and track people from the video footages.

The Convolutional Neural Network (CNN) model used by the system is based on the Mobilenet V2 face re-identification model, which is a TensorFlow source framework. The suggested design is put into practise utilising OpenVino (Open Visual Inference and Neural Network Optimization) on a budget-friendly Intel UP board.

Figure 2.6 shows the findings of the average processing times from several assessment platforms. When the model is optimised utilising OpenVino implementation rather than only the Intel i7 CPU, the processing time is decreased by 73%, from an average processing time of 52 ms to 14 ms. According to [5], this project utilises an Intel UP board that is less costly, optimised for OpenVino, and equipped with a quad-core CPU, and an integrated Intel GPU. The suggested technique is ideal for low-cost real-time edge face recognition applications because to the 60ms inference speed and satisfactory results.

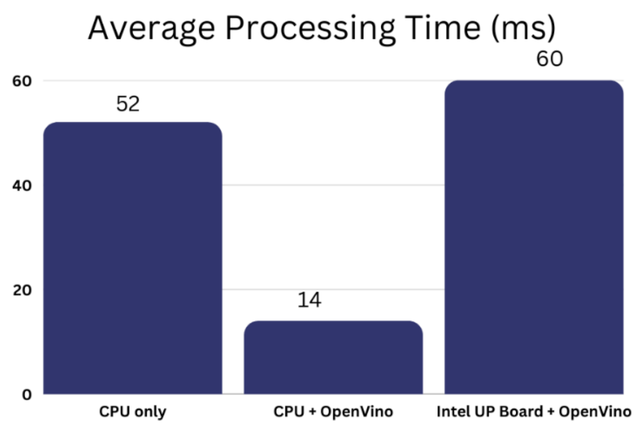


Figure 2.6 comparison of average processing time

2.1.5 Intel OpenVino Toolkit for Computer Vision: Object Detection and Semantic Segmentation

The paper[6] provides an overview of the current state of neural network implementation and introduces the Intel® OpenVino Toolkit for operating neural networks on various Intel hardware platforms. The experiment on many hardware platforms is described. It also provides a summary of the effectiveness and costs of using a single CPU or GPU to run specific neural networks in normal mode with the OpenVino Toolkit, automatically dividing neural network inference between CPU and GPU (Heterogeneous Plugin), and launching a network on CPU and GPU simultaneously (Multi-Device Plugin).

The investigation of object identification and semantic segmentation computer vision neural network types was the focus of the paper[6].The type of neural network such as RetinaNet, DUC, SSD, Tiny YOLOv2 and YOLOv2-COCO were used in this project. All neural networks were represented within the ONNX machine learning model framework.

Table 2.2 shows the three different CPU and two different GPU types that were tested on the target platform for neural network performance. In addition, GPU1 and GPU2 are integrated GPUs in the CPU2 and CPU3 CPUs.

Table 2.2 Target device which neural network performed

Target device	Device name
CPU1	Intel® Core ™ i9-9900KF;
CPU2	Intel® Core ™ i7-7700HQ;
CPU3	Intel® Core ™ i5-8250U;
GPU1	Intel® HD Graphics 630;
GPU2	Intel® UHD Graphics 620.

To describe the advantages of using the chosen device, a cost analysis (FPS) was performed (higher is better) as shown in Figure 2.7 – 2.10. The neural network DUC and yolo v2-coco's average performance dropped by 20.9% on the CPU and 16.2% on the GPU when the OpenVino toolkit was utilised. On the CPU and GPU, neural network SSD performance is often increased by roughly 30%. RetinaNet's performance when utilising the OpenVino toolkit is difficult to determine because it might periodically go down and then up by a tiny amount. Since it can increase by as much as 100%, tiny YOLOv2 should have an OpenVino framework that is well optimised (except for CPU1 which was unable to be tested). The biggest gain in performance (up to 141.3%) occurs on GPU2.

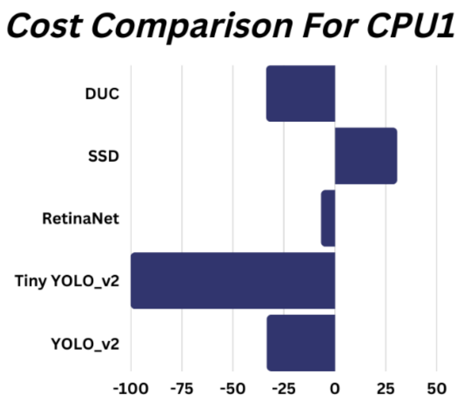


Figure 2.7 Cost comparison for cpu1

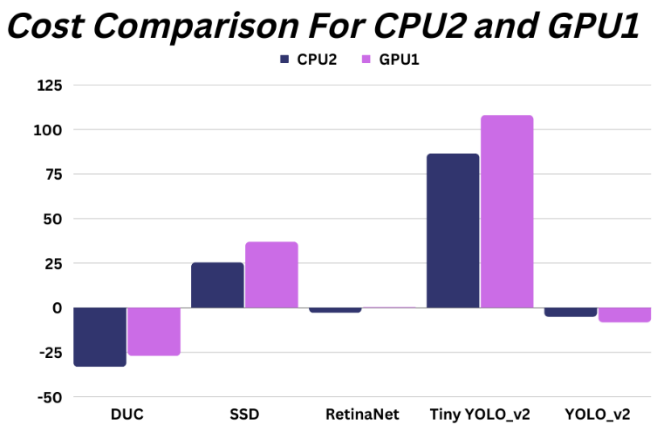


Figure 2.8 cost comparison for cpu2 and gpu1

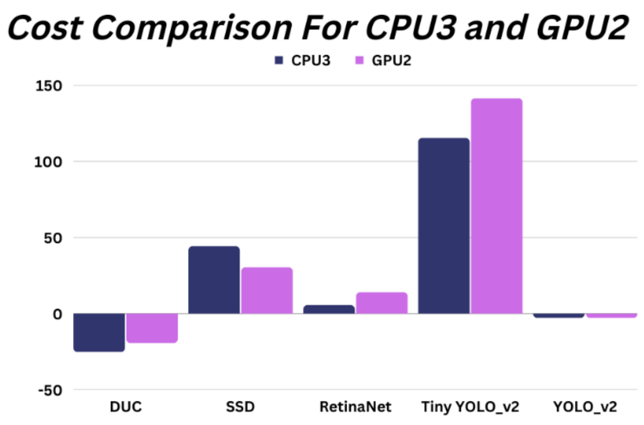


Figure 2.9 cost comparison for cpu3 and gpu2

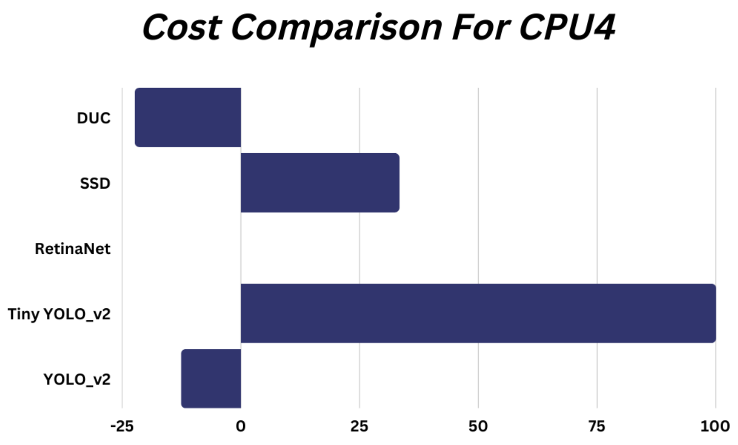


Figure 2.10 cost comparison for cpu4

It was difficult to say whether OpenVino actually improved the performance. It might not be tuned for a particular model, but according to a study by [6], if the model was properly optimised with OpenVino, performance could even go up by 141 percent. However, compared to a single device, using a multi-device plugin and heterogeneous plugin does not actually improve speed (cpu or gpu be executed only).

2.2 Conclusion of Improvement done by OpenVino Toolkit from previous study

These figure five studies have demonstrated how the OpenVino toolkit could well be improved. These studies used the OpenVino-supported TensorFlow, Caffe, and ONNX deep learning framework. And the majority of the research employed more user-friendly, broadly available, and affordable computers (except for 2.1.1 intel Xeon processor which is high end processor). Now, Let us wrap things up by evaluating the performance of the OpenVino toolkit.

In figure 2.7, it was demonstrated that OpenVino could successfully multiply the performance of neural network SSD MoblieNet V2 coco from the TensorFlow framework by up to 126.449 times. This project's processor, an affordable i5-4450 that can be considered a low-end processor, produces surprising results.

In figure 2.8 If the OpenVino toolkit is used in conjunction with the Adaptive-CS-Net from the TensorFlow framework, it can improve by a maximum of 54.4 times. However, it only improves three times in the W-Net neural network. Even less improvement is obtained for the low-cost CPU (i7-8665u) when OpenVino is used, with only a 19% increase in W-Net speed and a 7.5 time increase in Adaptive-CS Net speed.

In figure 2.9 OpenVino was tested using the BVLC caffe DetectNet neural network model of the Caffe deep learning framework. As a result, OpenVino was able to successfully increase the performance speed of i5 and i7 processors by about 30 times (processor model number were not stated).

In figure 2.10, The project uses an i7 processor and the OpenVino toolkit (model number was not stated). The CNN MobileNet V2 neural network model used in this project is based on the TensorFlow source framework. Compared to the not inferred model, it successfully increased system performance by 73%. (without implement OpenVino Inference Engine)

This project examined how OpenVino could be made better with various target devices that have three CPUs and two iGPUs. These processors are reasonably priced and user-friendly (except for CPU1 i9-9900KF which can considered as high cost). This project's neural network model includes DUC, SSD, RetinaNet, Tiny YOLOv2, and YOLO v2-coco. This entire neural network is based on the ONNX framework. As a result, performance for DUC and Yolo v2-Coco decreased while OpenVino toolkit was used. Only the performance of the other three neural network model has improved. This project also came to the conclusion that implementing a single GPU or CPU outperformed heterogeneous plugins and multi-device plugins in terms of improvement (sometimes even get more bad result).

Table 2.3 shows the overall performance of the OpenVino toolkit (only the cost-effective i5/i7 processor is displayed). As we can see, the TensorFlow Framework typically takes many advantages. Caffe framework also sees some good development. ONNX framework appears to be unstable because it occasionally causes the performance to drop. In addition, rather than being a problem with the deep learning framework, it could be that the OpenVino toolkit is not well optimised to a particular neural network model.

Table 2.3 Overview OpenVino toolkit improvement has made

neural network model	Framework	average improvement x	cpu used
SSD_MoblieNet_V2_coco	TensorFlow	26.45	i5-4450
W-Net	TensorFlow	0.19	i7 – 8665u
Adaptive-CS Net	TensorFlow	7.9	i7 – 8665u
BVLC caffe detectNet	Caffe	30	i5/i7 -
CNN MobileNet V2	TensorFlow	0.73	i7 -
DUC	ONNX	-33.02	i7-7700HQ
SSD	ONNX	25.28	i7-7700HQ
RetinaNet	ONNX	-2.46	i7-7700HQ
Tiny YOLOv2	ONNX	86.41	i7-7700HQ
YOLO v2-coco	ONNX	-5	i7-7700HQ
DUC	ONNX	-25.06	i5-8250U
SSD	ONNX	44.08	i5-8250U
RetinaNet	ONNX	5.46	i5-8250U
Tiny YOLOv2	ONNX	115.17	i5-8250U
YOLO v2-coco	ONNX	-2.58	i5-8250U

CHAPTER 3 System Methodology/Approach

3.2 System Development Methodology

For the first objective of measuring the performance enhancement introduced by OpenVino and selecting the most effective DL framework, I utilized the waterfall model to ensure a structured and systematic approach to development.

The planning phase involved a thorough analysis of the project requirements, followed by the design phase where I created a detailed system architecture and specified the necessary hardware and software components. Then, I implemented the system and conducted comprehensive testing to measure performance and select the most effective DL framework.

For the second objective of configuring and deploying the traffic violation detection on edge device along with OpenVino Toolkits, I followed a similar approach using the waterfall model. I started with a planning phase that included analyzing the requirements and designing the system architecture. Next, I implemented the system and tested it thoroughly to ensure proper functionality. Finally, I deployed the system to the edge device, followed by maintenance and support as needed.

Overall, utilizing the waterfall model helped me to develop and deploy my traffic violation detection system in a structured and systematic manner, ensuring that each phase of development was completed before moving on to the next.

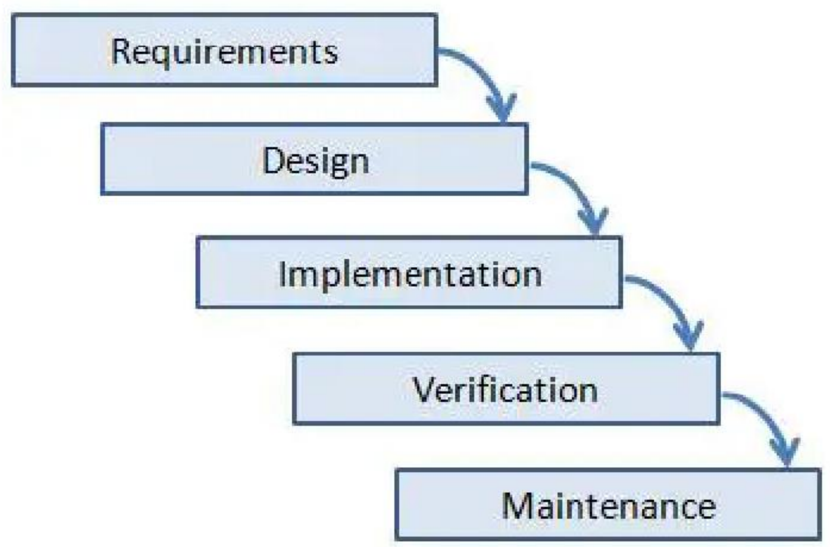


Figure 3.1 Waterfall Model

3.3 System Design Diagram

3.3.1 System Architecture diagram

The traffic violation detection system architecture consists of several components, including the client, internet, IoT edge device, IoT camera, database, and admin panel. The components are connected to one another to facilitate the detection and reporting of traffic violations. The following is a description of each component:

- Client: The client component provides the user interface for the admin to interact with the system. The admin can set up the activation time for the system, view the report of detected violations, and manage the system using the admin panel.

- Internet: The internet component connects the client and IoT edge device, allowing the admin to access and manage the system remotely.

- IoT Edge Device: The IoT edge device component is responsible for capturing real-time images from the road using the IoT camera. The edge device performs real-time image processing to detect illegal traffic vehicles.

- IoT Camera: The IoT camera component captures real-time images of the road, which are transmitted to the edge device for processing.

- Database: The database component stores the car plate numbers of detected violations for reporting purposes. The admin can access the database through the admin panel to view the report of detected violations

- Admin Panel: The admin panel component provides a user-friendly interface for the admin to set up the activation time, view the report of detected violations, and manage the system.

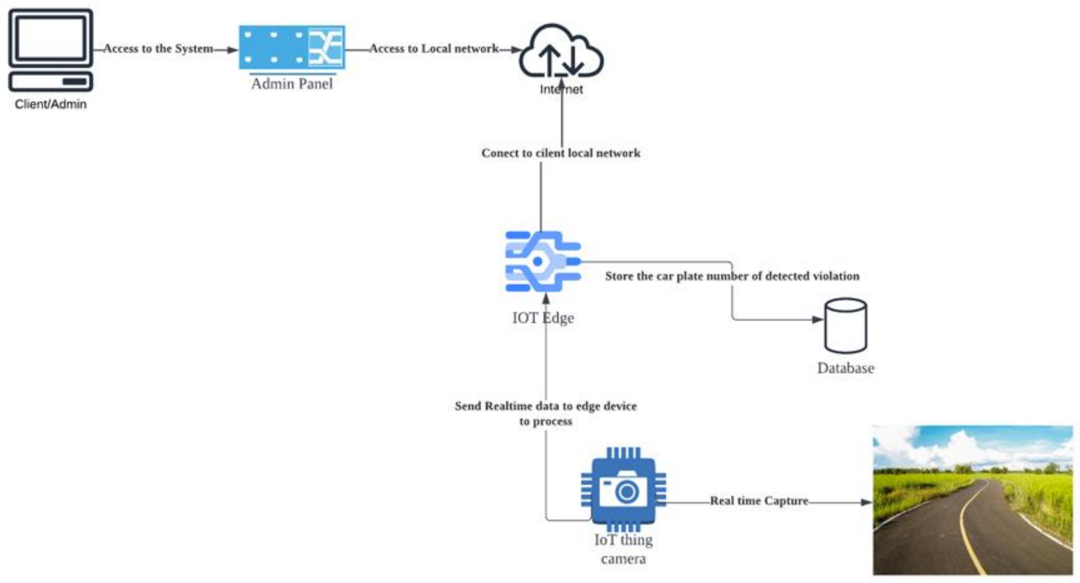


Figure 3.2 System Architecture Diagram

3.3.2 Use case diagram of Traffic Violation Detection System

The use case diagram for the traffic violation detection system includes three actors: admin, edge device (Intel UP board), and illegal traffic vehicle. The admin can set the system activation time and check for corresponding illegal vehicles. The edge device is responsible for detecting illegal traffic vehicles using car plate recognition and saving the recognized carplate. The illegal traffic vehicle triggers the detection system when it violates traffic rules, such as driving in the wrong direction in a one-way lane.

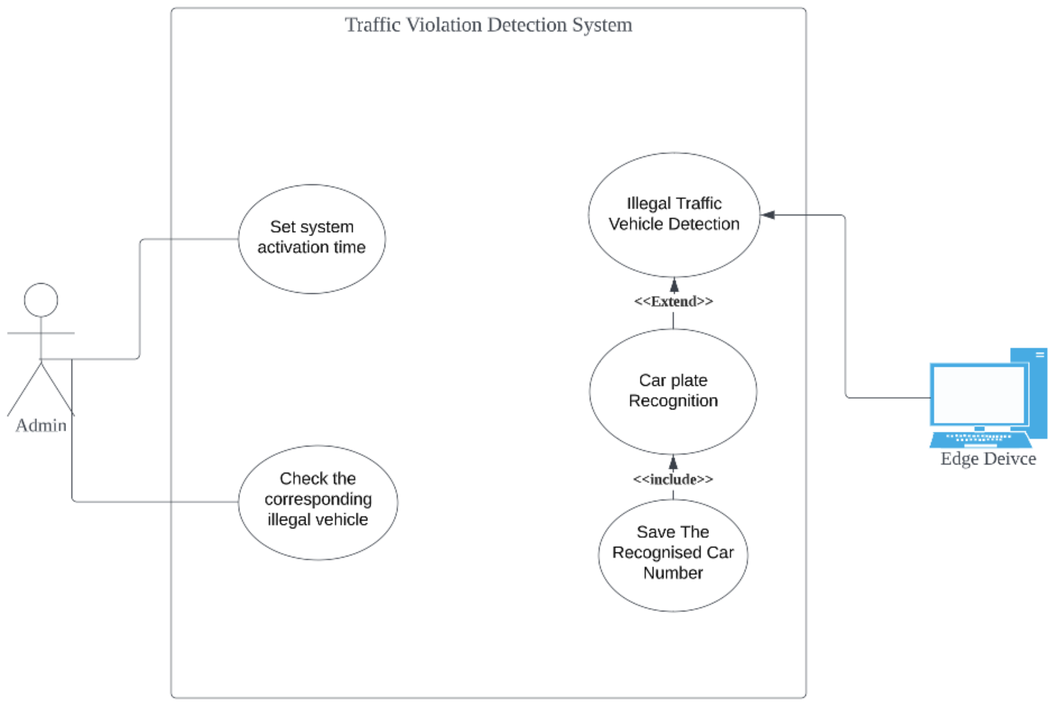

Traffic Violation Detection System
Set system activation time
Check the corresponding illegal vehicle
Admin
Illegal Traffic Vehicle Detection
<<Extend>>
Car plate Recognition
<<include>>
Save The Recognised Car Number
Edge Deivce

Figure 3.3 Use Case Diagram of Traffic Violation Detection System

3.3.3 Activity Diagram of Traffic Violation Detection System

The activity diagram for the traffic violation detection system shows the steps involved in detecting illegal traffic vehicles and reporting their car plate numbers to the admin. The process starts with the admin setting up the activation time for the system. The system then captures real-time images from the camera, which are analyzed for any illegal traffic vehicles.

If no illegal traffic vehicles are detected, the system loops back to real-time capture, continuing to monitor the road until the end of the activation time. However, if an illegal traffic vehicle is detected, the system performs car plate number recognition using OCR technology. The recognized car plate number is then saved to the system's database for reporting purposes.

After saving the car plate number, the system checks if the end of the activation time has been reached. If not, the system loops back to real-time capture to continue monitoring for additional violations. However, if the end of the activation time has been reached, the system allows the admin to view the report containing the car plate numbers of all the detected violations. Finally, the system ends.

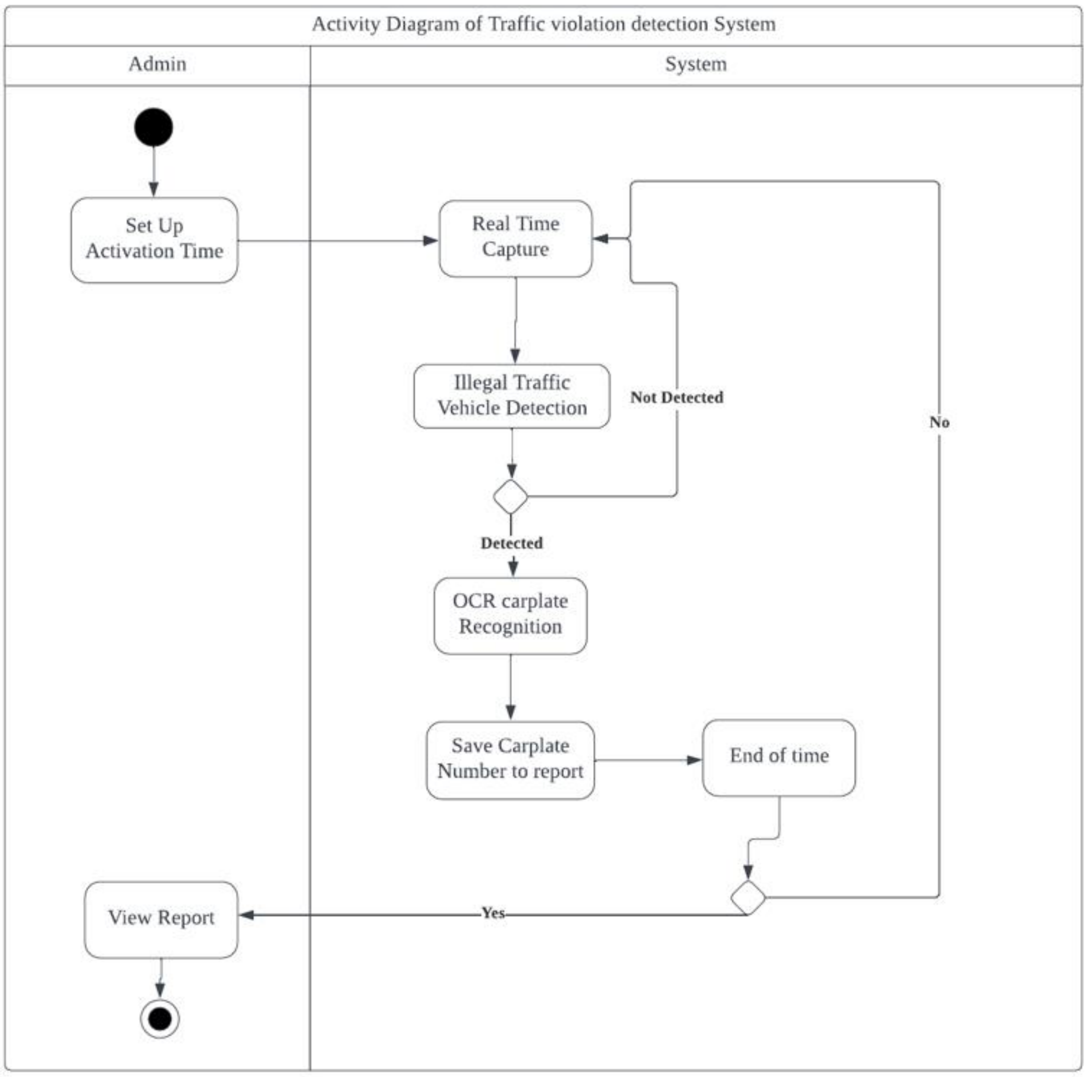
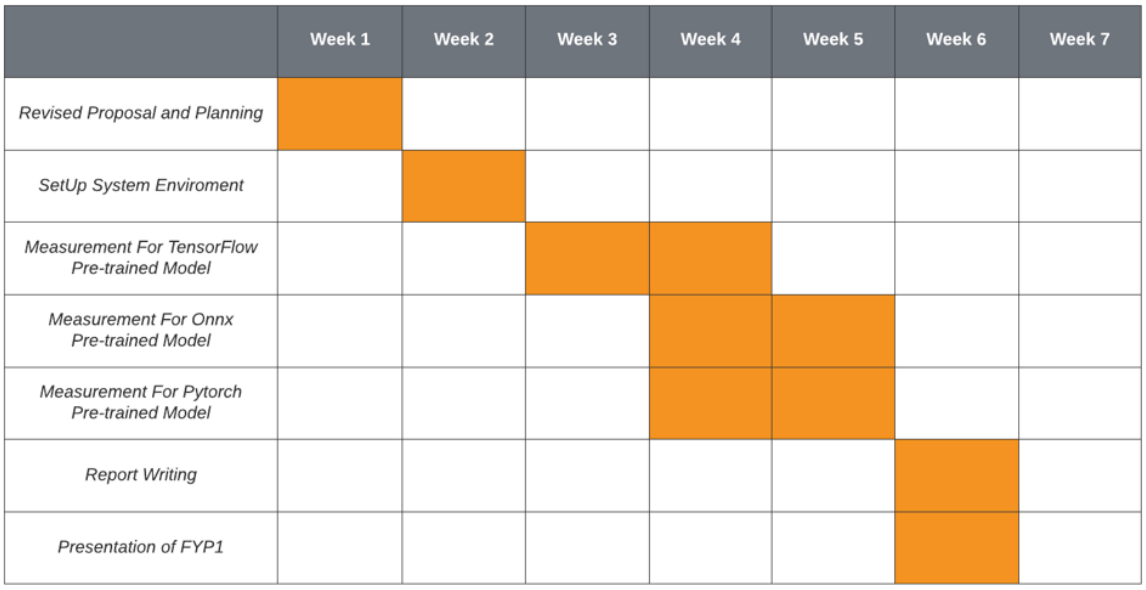


Figure 3.4 Activity Diagram of Traffic Violation Detection System

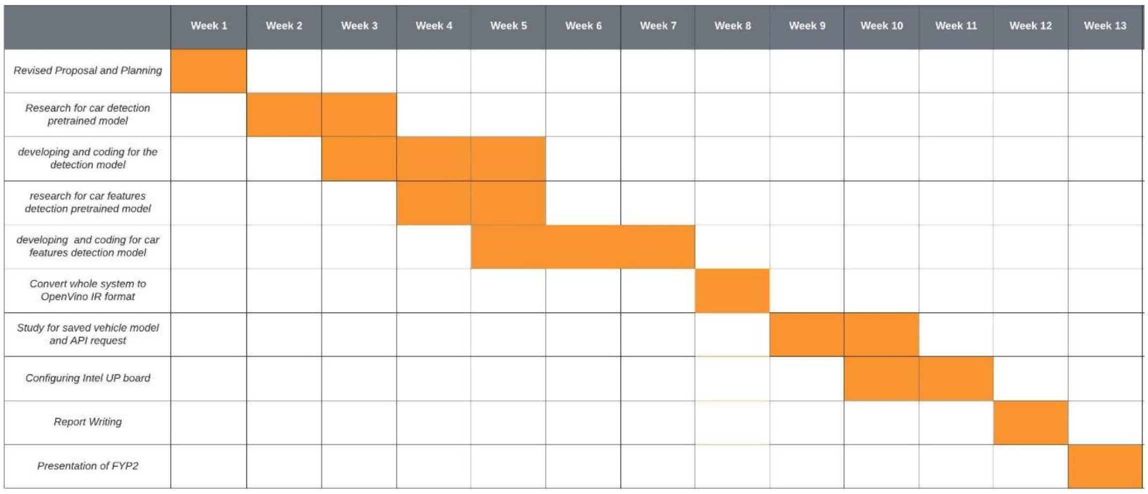
3.4 Timeline

The FYP1 and FYP2 project timeline (Gantt chart) has shown below:



	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7
Revised Proposal and Planning	🟧						
SetUp System Enviroment		🟧					
Measurement For TensorFlow Pre-trained Model			🟧	🟧			
Measurement For Onnx Pre-trained Model				🟧	🟧		
Measurement For Pytorch Pre-trained Model				🟧	🟧		
Report Writing						🟧	
Presentation of FYP1						🟧	

Figure 3.5 Timeline of FYP1



	Week 1	Week 2	Week 3	Week 4	Week 5	Week 6	Week 7	Week 8	Week 9	Week 10	Week 11	Week 12	Week 13
Revised Proposal and Planning	🟧												
Research for car detection pretrained model		🟧	🟧										
developing and coding for the detection model			🟧	🟧	🟧								
research for car features detection pretrained model				🟧	🟧								
developing and coding for car features detection model					🟧	🟧	🟧						
Convert whole system to OpenVino IR format								🟧					
Study for saved vehicle model and API request									🟧	🟧			
Configuring Intel UP board										🟧	🟧		
Report Writing												🟧	
Presentation of FYP2													🟧

Figure 3.6 Timeline of FYP2

CHAPTER 4 System Design

The entire project workflow is depicted in Figure 4.1. The project first will setup system environment to implement OpenVino Toolkit. Then, we will compare the performance improvements made by OpenVino to a variety of deep learning frameworks (such as ONNX, Pytorch, and TensorFlow) to ensure that OpenVino actually contributes to improving performance and in order to select the most effective deep learning framework to use in associated with OpenVino. Next, we will configure traffic violation detection system using the most effective deep learning frameworks along with OpenVino Toolkit. At last, the system will then be installed along the roadside after being configured on edge device (Intel Up Board).

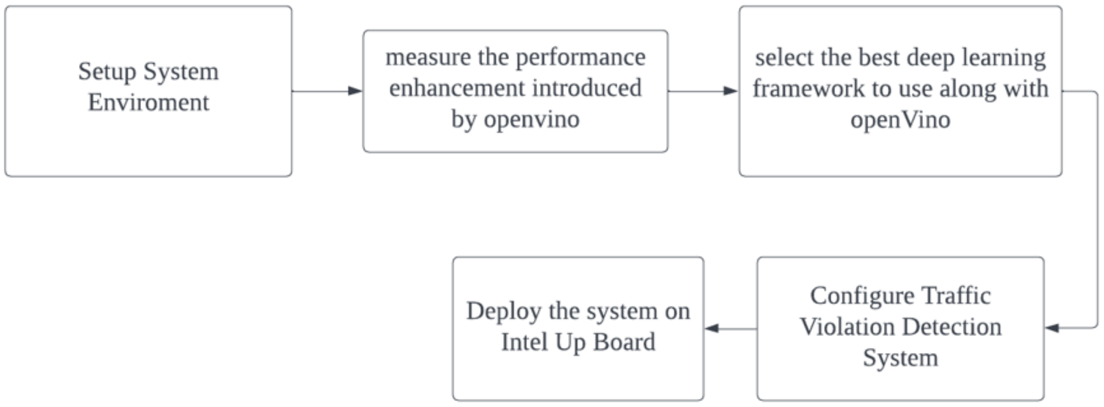

Setup System Enviroment
measure the performance enhancement introduced by openvino
select the best deep learning framework to use along with openVino
Configure Traffic Violation Detection System
Deploy the system on Intel Up Board

Figure 4.1 Whole Project workflow

4.1 Performance Enhancement with OpenVino and DL Framework Selection

The aim is to evaluate the impact of OpenVino on the performance of the system and to identify the most effective deep learning framework.

4.2.1 Flow Diagram of Performance Measurement introduced by OpenVino

In this section, it will iterate the process from figures 4.2 and 4.3 three times. We will first pick a pre-trained model for object detection that uses the TensorFlow, Pytorch, and ONNX deep learning frameworks. With the flow diagram from figure 4.2 we will first evaluate the performance of the initial framework. The same pre-trained model will then pass through OpenVino's model optimizer to produce IR format, which will be used to implement the inference engine. The OpenVino's optimized model (IR format) will then be measured as shown in the flow diagram in figure 4.3. Then, we will record the processed frames per second (fps) and sketch out how much percentage of improvement OpenVino has made. By following the improvement percentages from the three deep learning frameworks. The best effective deep learning framework will then be chosen and configured for integration with the traffic violation detection system in this project.

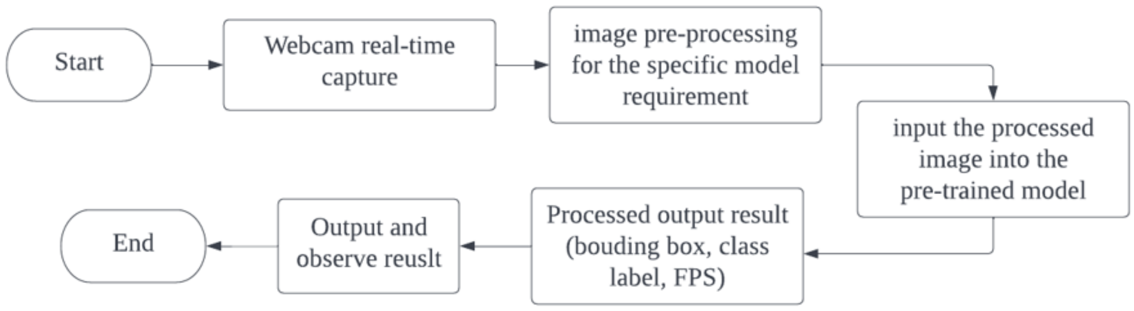

Start
Webcam real-time capture
image pre-processing for the specific model requirement
input the processed image into the pre-trained model
Processed output result (bouding box, class label, FPS)
Output and observe reuslt
End

Figure 4.2 Pre-trained Object Detection model system flow diagram

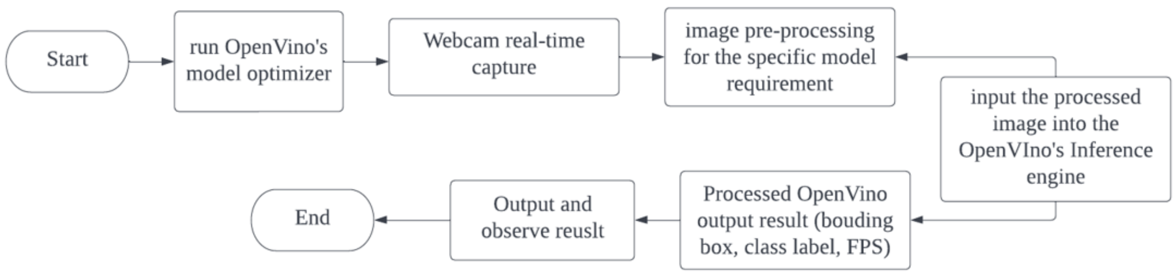

Start
run OpenVino's model optimizer
Webcam real-time capture
image pre-processing for the specific model requirement
input the processed image into the OpenVIno's Inference engine
Processed OpenVino output result (bouding box, class label, FPS)
Output and observe reuslt
End

Figure 4.3 OpenVino Object Detection system flow diagram

4.2 Traffic Violation Detection System Deployment with OpenVino

After completing the first objective as per Chapter 4.2, the most effective DL framework will be identified. Subsequently, the traffic violation detection system will be developed, and the system's high-level overview will be presented below.

4.2.1 Traffic Violation Detection System System Block Diagram

The system block diagram consists of several components, including the client, IoT edge device, IoT camera, and the admin panel. The client is responsible for accessing the system and triggering the detection process. The IoT edge device is equipped with the necessary hardware components, such as an Intel UP board and a power bank, to support the detection and recognition processes. The IoT camera captures the live video feed of the traffic, which is then processed by the deep learning models built with Tensorflow and OpenVINO. Upon detection of a traffic violation, the system records the car plate number and the date and time of the incident, which is stored in a text file for future reference. Finally, the admin panel provides the user with the ability to interact with the system and view the report generated, containing the details of the traffic violations detected by the system.

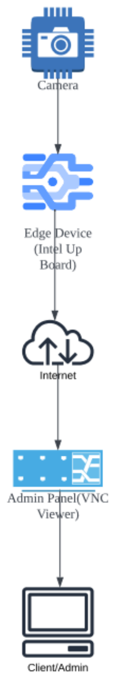


Figure 4.4 System Block Diagram of Traffic Violation Detection System

4.2.2 Flow Diagram of Traffic Violation System Development

Before the traffic violation system begins, the pre-trained deep learning models will be processed by OpenVino's model optimizer to create the IR format necessary to run with OpenVino's inference engine. Figure 4.5 demonstrates the workflow of the Traffic Violation System. The system will first capture the real-time snapshot of the traffic situation. Next, Pre-processing will be applied to the captured image, followed by vehicle detection (fixed position of vehicle) to detect the offending vehicle. Next, The detected vehicle will be perfectly cropped out and insert into the others DL model (EasyOCR) for detecting vehicle features. At last, the result will be processed and recorded in a txt file in order for the administrator to check which vehicle has broken traffic laws.

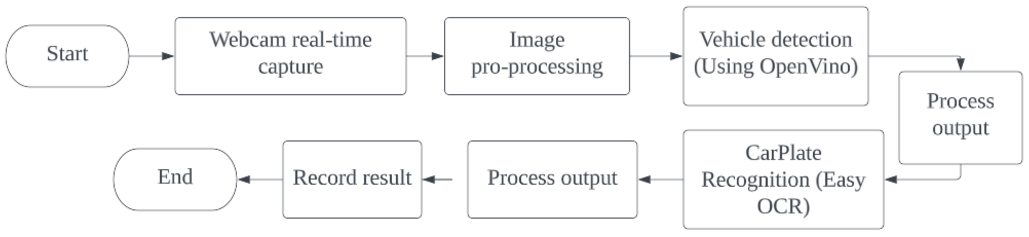


Figure 4.5 Traffic Violation system Flow Diagram

4.3 System Components Specifications

4.3.1 Hardware Components

1. **Workstation**

In this project, we use the OpenVino toolkit to benefits our system deep learning workloads and boosts the inference time. Only Intel hardware is officially supported by the OpenVino toolkit. It requires a computer system with Intel Core Processors ranging from the 6th to the 12th generation. The laptop we set up for this project study is shown in Table 4.1.

Table 4.1 Specifications of laptop

Description	**Specifications**
Model	Microsoft Surface Pro 7
Processor	Quad-core 10th Gen Intel® Core™ i7-1065G7 Processor
Operating System	Windows 10 Home
Graphic	Intel® Iris™ Plus Graphics
Memory	16GB DDR4x RAM
Storage	256GB SSD

2. Intel Up Board

Furthermore, an edge device will be used to deploy the deep learning applications for traffic violations. However, the edge device we select must also be compatible with the OpenVino Toolkit. Hence, Intel UP Board is selected as the edge device as shown in Figure 3.0. Besides, table 3.2 displays the edge device's specification.

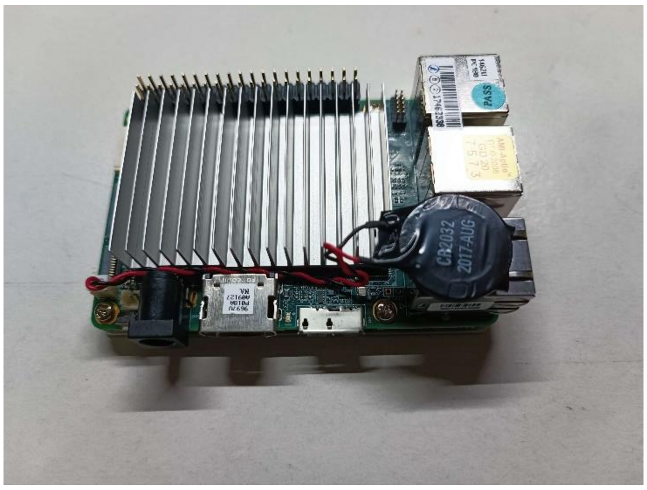


Figure 4.6 Intel UP Board for testing in edge device

Table 4.2 Specifications of Edge Device

Description	Specifications
Model	Intel UP Board, UP-CHT01-A10-0432
Processor	Intel® Atom™ x5 Z8350
Operating System	Ubuntu 20.04 LTS
Graphic	Intel® HD400
Memory	4GB DDR3L RAM
Storage	32GB eMMC

3. Camera

For this project, a Logitech C615 camera will be used to perform the deep learning task. While a more professional camera could potentially be used with the edge device, the Logitech C615 has been selected as a cost-effective option that meets the project's requirements.

Figure 4.7 Camera (Logitech C615)

4. Power Bank

To simulate the power supply of the edge device, a power bank has been utilized for this project. This allows the edge device to be powered wherever it goes and wherever it is installed, without the need for a fixed power source. By leveraging a portable power source, the edge device can be easily deployed in various settings, making it more versatile and flexible. The specific power bank model used for this project has been selected based on its capacity, output voltage, and compatibility with the edge device, in order to ensure reliable and stable performance during operation.

Figure 4.8 Power Bank

5. **Wifi USB adapter**

   In this project, a Dlink DWA-123 Wifi USB adapter was selected to provide wireless connectivity to the edge device. This adapter is capable of providing up to 150Mbps of wireless speed, and is designed to be plug and play, making it easy to use. The adapter was connected to the edge device, allowing it to connect to a local wifi network and subsequently to the admin PC through VNC viewer. The selection of this adapter was based on its affordability and ease of use, making it a practical option for the project requirements.

Figure 4.9 Wifi Adapter (Dlink DWA-123)

4.3.2 Software Components

1. **Python 3.8**

   Python 3.8 was used as the primary programming language for this project due to its popularity, versatility, and extensive libraries and tools available for scientific computing and machine learning. Python has emerged as a popular choice among researchers and practitioners in the field of artificial intelligence and data science due to its ease of use, readability, and flexibility. Additionally, Python's open-source nature and large community of developers have made it a reliable and well-supported platform for developing and deploying machine learning applications.

Figure 4.10 Python

2. Jupyter Notebook

JupyterLab was chosen as the primary IDE for this project due to its user-friendly interface and integration with various data formats and programming languages. Its ability to combine code, visualization, and documentation in a single notebook makes it an ideal platform for data analysis and machine learning experimentation.

Figure 4.11 Jupyter Notebook

3. Deep learning framework

This project involves the implementation and testing of several deep learning models using the open-source software toolkit OpenVINO, which allows for efficient deployment of neural networks on a variety of hardware platforms. The deep learning frameworks used in this project include ONNX, PyTorch, and TensorFlow, which are popular frameworks for developing and training deep learning models. The use of these frameworks allowed for the exploration of different neural network architectures and optimization techniques to achieve optimal performance on the target hardware.

Figure 4.12 Deep Learning Framework (Tensorflwo, ONNX, PyTorch)

4. OpenVino Toolkit 2022

The OpenVINO Toolkit 2022 was used to optimize and deploy deep learning models for traffic violation detection on edge devices in this project. The toolkit's support for popular deep learning frameworks and optimization techniques allowed for efficient execution of models on limited hardware resources.

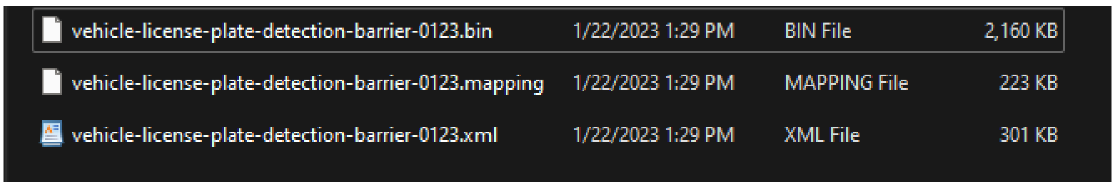
OpenVINO™

Figure 4.13 OpenVino Toolkit

5. Pretrained – Model

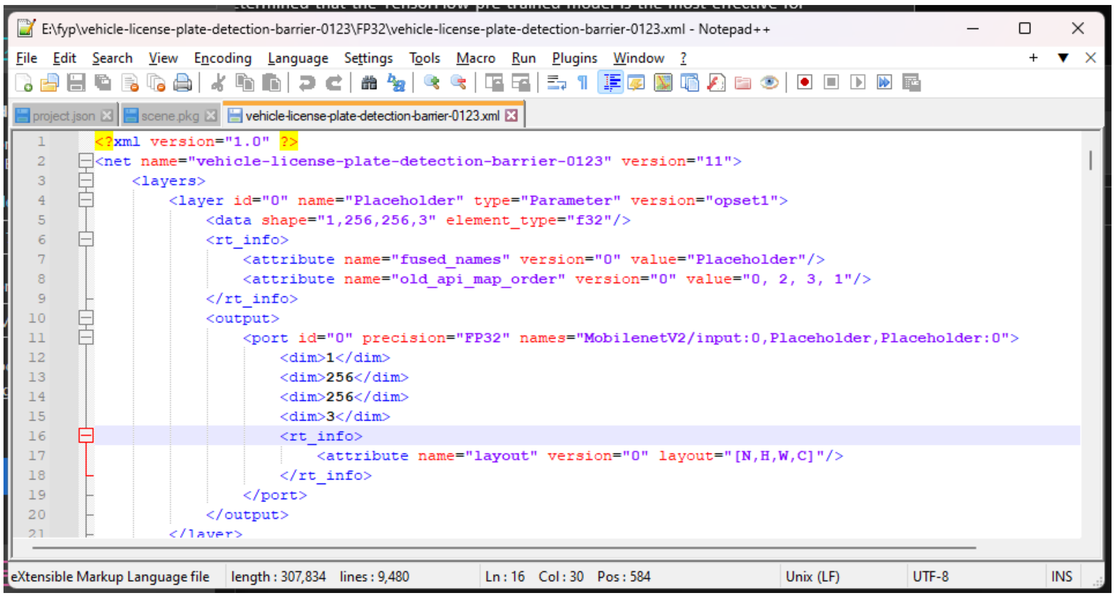
In this project, we will use pre-trained object detection models from TensorFlow, PyTorch, and other frameworks to evaluate their effectiveness for integration with OpenVINO. By testing and comparing the performance of these models, we aim to identify the best one for our project's requirements.

Based on the results of the first objective outlined in Chapter 5, we have determined that the TensorFlow pre-trained model is the most effective for our project. We subsequently selected a pre-trained model for vehicle and license plate detection using TensorFlow. Specifically, we chose the "Vehicle-license-plate-detection-barrier-0123" model from OpenModelZoo on GitHub because it is used to specifically detect front-facing car. While the model framework was originally created using TensorFlow, and it was converted into an IR model that is fully compatible with the OpenVINO toolkit.



vehicle-license-plate-detection-barrier-0123.bin	1/22/2023 1:29 PM	BIN File	2,160 KB
vehicle-license-plate-detection-barrier-0123.mapping	1/22/2023 1:29 PM	MAPPING File	223 KB
vehicle-license-plate-detection-barrier-0123.xml	1/22/2023 1:29 PM	XML File	301 KB

Figure 4.14 Vehicle-license-plate-detection-barrier-0123 model (IR files)



E:\fyp\vehicle-license-plate-detection-barrier-0123\FP32\vehicle-license-plate-detection-barrier-0123.xml - Notepad++

```xml
<?xml version="1.0" ?>
<net name="vehicle-license-plate-detection-barrier-0123" version="11">
    <layers>
        <layer id="0" name="Placeholder" type="Parameter" version="opset1">
            <data shape="1,256,256,3" element_type="f32"/>
            <rt_info>
                <attribute name="fused_names" version="0" value="Placeholder"/>
                <attribute name="old_api_map_order" version="0" value="0, 2, 3, 1"/>
            </rt_info>
            <output>
                <port id="0" precision="FP32" names="MobilenetV2/input:0,Placeholder,Placeholder:0">
                    <dim>1</dim>
                    <dim>256</dim>
                    <dim>256</dim>
                    <dim>3</dim>
                    <rt_info>
                        <attribute name="layout" version="0" layout="[N,H,W,C]"/>
                    </rt_info>
                </port>
            </output>
        </layer>
```

eXtensible Markup Language file | length : 307,834 lines : 9,480 | Ln : 16 Col : 30 Pos : 584 | Unix (LF) | UTF-8 | INS

Figure 4.15 Vehicle-license-plate-detection-barrier-0123 network architecture

6. EasyOCR

EasyOCR was incorporated into the traffic violation detection system of this project to enhance the recognition of text on license plates. This tool uses advanced machine learning algorithms that allow it to accurately identify characters and words from images, even in challenging conditions or enviroments. By using EasyOCR, this project achieved high accuracy and speed in recognizing and extracting the text on car plates. EasyOCR's ease of integration made it a convenient and reliable solution for car plate recognition. We follow the documentation of EasyOCR through [11] on GitHub.

Figure 4.16 EasyOCR

7. **Ubuntu 20.04.6 LTS**

Ubuntu 20.04.6 LTS was chosen as the OS for the Intel Up board edge device due to its stability, security, and compatibility with the hardware. Its LTS release model offers five years of support and security updates. Additionally, Ubuntu is well-supported by the open-source community and provides a range of software packages for machine learning development. It also offers necessary drivers and software libraries required for running the OpenVINO Toolkit and deep learning frameworks on the Intel Up board.

Figure 4.17 Ubuntu 20.04 LTS

8. **VNC Viewer**

VNC viewer was used to remotely access the Intel Up board edge device in this project due to its cross-platform compatibility and ease of use. VNC viewer allows for remote desktop access to the device from any other computer on the same network, enabling remote development and testing of machine learning models. Its cross-platform compatibility ensures that it can be used on a range of devices and operating systems, making it a flexible and convenient option for accessing the edge device. Additionally, VNC viewer's simple user interface makes it easy to set up and use, even for those with limited technical expertise.

Figure 4.18 VNC Viewer

4.4 System Components Interaction Operations

In this project, a webcam was utilized to simulate an IoT camera, a power bank was used as a substitute for a professional power supply, and a Wi-Fi adapter was employed to connect the edge device to a wireless network. The webcam captures the real-time traffic conditions at the roadside, and the edge device processes the results using the selected deep learning framework. The detected car plate numbers and the corresponding date and time information are saved in a text file for record keeping. Users can access the report through an admin panel (VNC viewer) to retrieve the data from the edge device.

It is important to note that the devices used in this project are a simulation of professional-grade equipment. While the results achieved by this simulation are satisfactory, it is expected that a more professional and expensive equipment will provide even better results. The use of professional equipment is recommended in real-world deployments to ensure the highest level of accuracy and reliability in traffic violation detection systems.

CHAPTER 5 System Implementation

5.1 Software Setup (On WorkStation - Window 10)

Before starting to develop this project, there are three software needed to be installed and downloaded in my device which is:

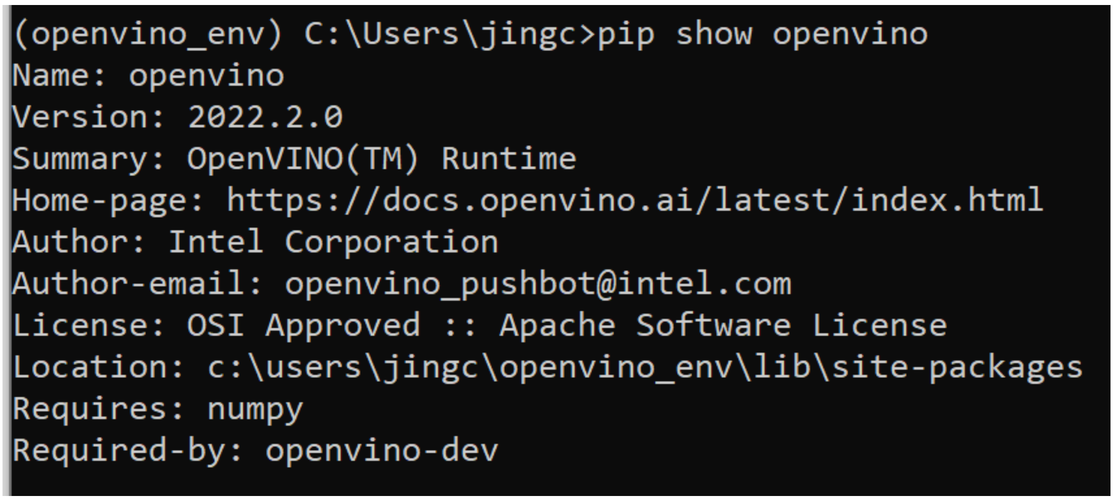
1. Python 3.8.8

2. OpenVino 2022.2.0

3.JupyterLab 3.5.0

4.2 Setting up OpenVino enviroment

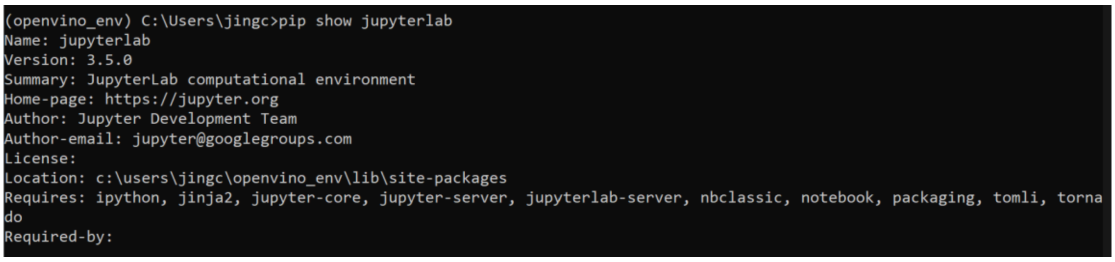
The commands "python -m venv OpenVino env" and "OpenVino envScriptsactivate" can be used to create an environment for the OpenVino project after Python 3.8.8 has been installed on the machine. Next, we install the OpenVino development toolkit in the specific OpenVino environment by using the command "pip install OpenVino-dev[ONNX,TensorFlow2,pytorch]==2022.2.0". We only include "[ONNX, TensorFlow2, Pytorch" in the bracket because we only consider using these deep learning frameworks for our work. Use the command shown in figure 5.1 to verify that if the OpenVino development tools have been properly installed.



```
(openvino_env) C:\Users\jingc>pip show openvino
Name: openvino
Version: 2022.2.0
Summary: OpenVINO(TM) Runtime
Home-page: https://docs.openvino.ai/latest/index.html
Author: Intel Corporation
Author-email: openvino_pushbot@intel.com
License: OSI Approved :: Apache Software License
Location: c:\users\jingc\openvino_env\lib\site-packages
Requires: numpy
Required-by: openvino-dev
```

Figure 5.1 Version of OpenVino development tools

Additionally, Jupyter Notebook serves as our integrated development environment (IDE), allowing us to work on our project efficiently. The command "pip install jupyterlab" can be used to install Jupyter notebooks. Figure 5.2 shows if Jupyterlab has been installed successfully.



```
(openvino_env) C:\Users\jingc>pip show jupyterlab
Name: jupyterlab
Version: 3.5.0
Summary: JupyterLab computational environment
Home-page: https://jupyter.org
Author: Jupyter Development Team
Author-email: jupyter@googlegroups.com
License:
Location: c:\users\jingc\openvino_env\lib\site-packages
Requires: ipython, jinja2, jupyter-core, jupyter-server, jupyterlab-server, nbclassic, notebook, packaging, tomli, torna
do
Required-by:
```

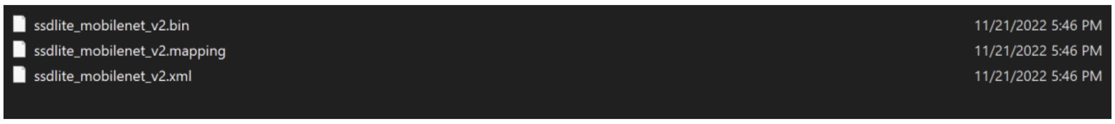
Figure 5.2 Version of Jupyterlab (IDE for python)

5.2 System Implementation for Objective 1: Performance Measurement introduced by OpenVino

We evaluated the performance of three deep learning frameworks in our project in order to determine the most effective deep learning framework optimised for OpenVino and so to deploy the traffic violation detection system in our edge device. TensorFlow, Pytorch, and ONNX were selected for testing because of their established deep learning communities. We choose an object detection pre-trained model that was trained using the COCO 2017 dataset and 80 labeled objects from each framework in order to have a fair comparison process. Besides, all precision formats used are "FP32" and the input batch size is 1.

5.2.1 Measurement for TensorFlow Pre-trained Model (Mobilev2_SSDlite)

The TensorFlow pre-trained model that we used was came from [7]. The pre-trained model is from the open model zoo which is officially published by Intel. Next, we use OpenVino's model optimizer to convert the original TensorFlow pre-trained model format into IR format (OpenVino's optimised model that required to run OpenVino Inference engine). The converted IR files (.xml* + .bin*) is shown in figure 5.3. Figure 5.4 displays the processed FPS result from the original model, while Figure 5.5 displays the processed FPS result for the OpenVino optimized model under IR format. This TensorFlow pre-trained model is provided from intel’s Open Model Zoo, after being optimized by OpenVino, the performance is improved by about 3.04x times (23.5 FPS to 71.5FPS).



```
ssdlite_mobilenet_v2.bin          11/21/2022 5:46 PM
ssdlite_mobilenet_v2.mapping      11/21/2022 5:46 PM
ssdlite_mobilenet_v2.xml          11/21/2022 5:46 PM
```

Figure 5.3 IR files of TensorFlow pre-trained model

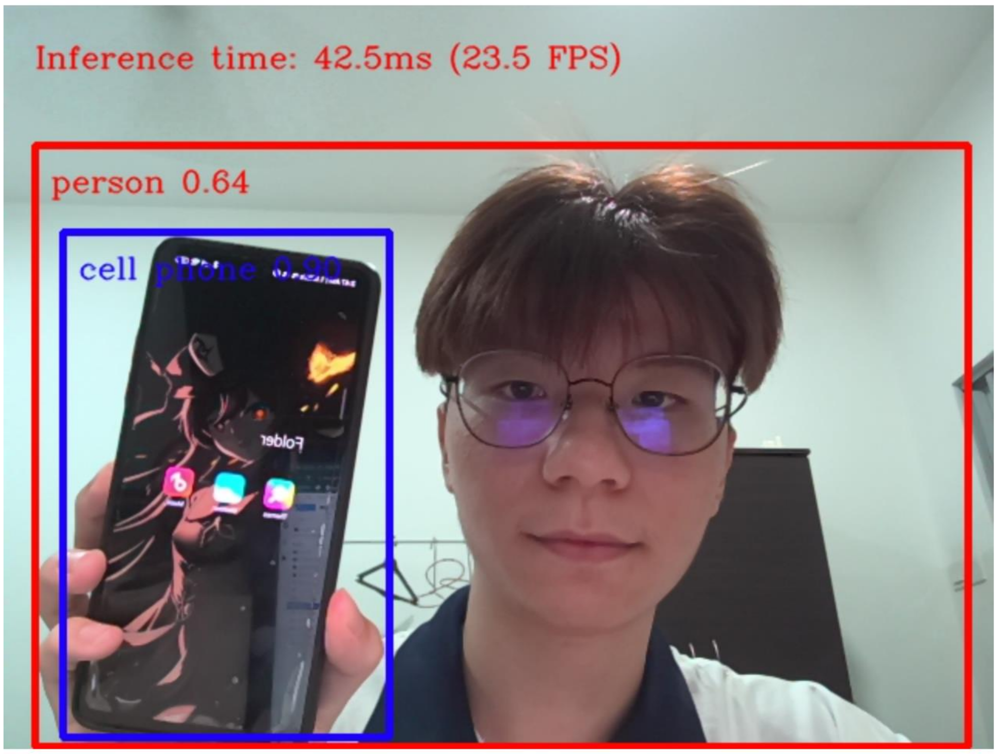


Figure 5.4 Result of original TensorFlow Mobilev2_SSDlite pre-trained model

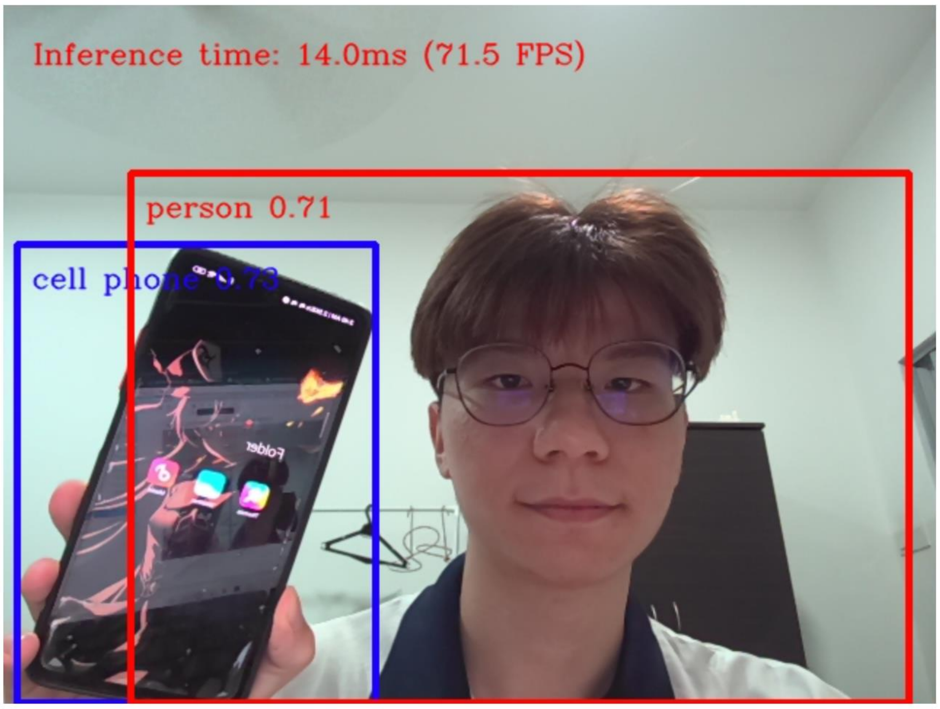
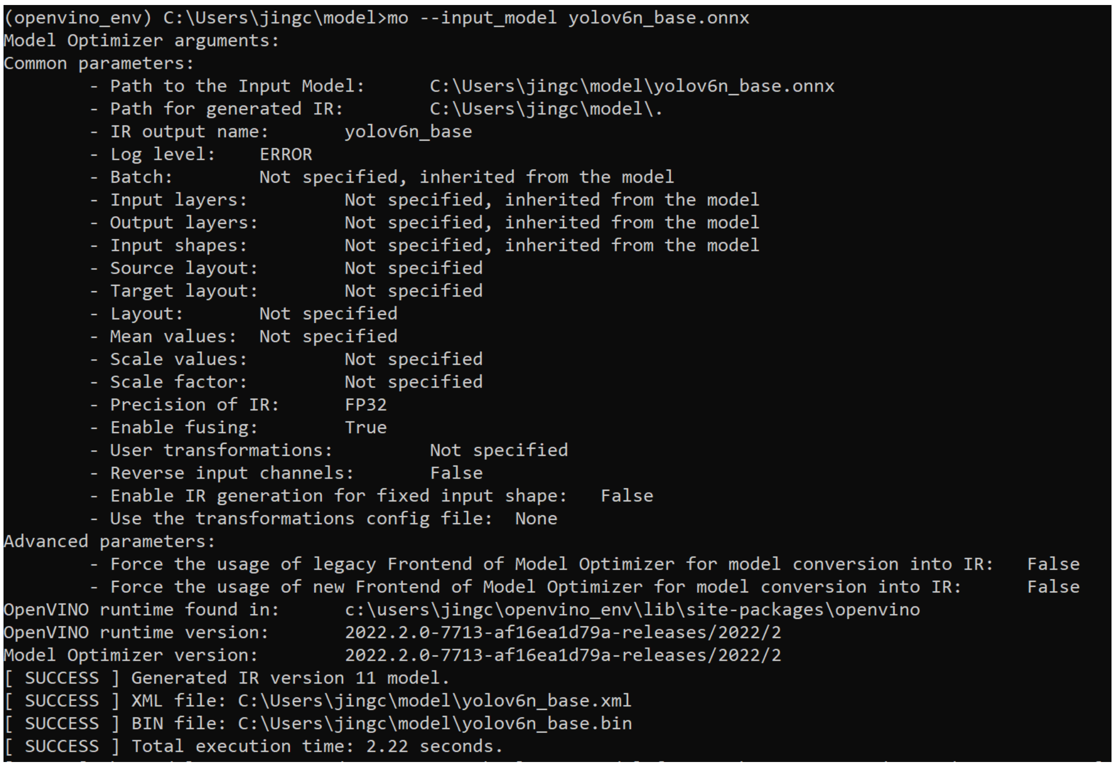


Figure 5.5 Result of OpenVino-Optimized (TensorFlow-to-IR) mobilev2_ssdlite pre-trained model

5.2.2 Measurement for ONNX Pre-trained Model (YOLOv6n)

To test whether OpenVino actually improves the performance of non-Intel-provided models and its general applicability for new DNNs, we selected a model that wasn't made by Open Model Zoo. Instead, we pick for the currently popular new DNNs framwork, YOLOv6, which was released by Meituan from [8] recently in June 2022. The pre-trained ONNX model (yolov6n_base.ONNX) that we used to test the ONNX framework is downloaded from [9]. Then, as shown in figure 5.6, we implement OpenVino's model optimizer to transform the original ONNX YOLOv6 pre-trained model format into IR format (OpenVino-Optimized format). Figure 5.7 displays the result of the original ONNX model, while Figure 4.8 displays the result of the model with OpenVino optimization (IR format). The Yolov6n ONNX pre-training model released by Meituan has improved its performance by about 1.2x(from 14.3FPS to 17.4FPS) times after being optimized by OpenVino.



```
(openvino_env) C:\Users\jingc\model>mo --input_model yolov6n_base.onnx
Model Optimizer arguments:
Common parameters:
        - Path to the Input Model:      C:\Users\jingc\model\yolov6n_base.onnx
        - Path for generated IR:        C:\Users\jingc\model\.
        - IR output name:       yolov6n_base
        - Log level:    ERROR
        - Batch:        Not specified, inherited from the model
        - Input layers:         Not specified, inherited from the model
        - Output layers:        Not specified, inherited from the model
        - Input shapes:         Not specified, inherited from the model
        - Source layout:        Not specified
        - Target layout:        Not specified
        - Layout:       Not specified
        - Mean values:  Not specified
        - Scale values:         Not specified
        - Scale factor:         Not specified
        - Precision of IR:      FP32
        - Enable fusing:        True
        - User transformations:         Not specified
        - Reverse input channels:       False
        - Enable IR generation for fixed input shape:   False
        - Use the transformations config file:  None
Advanced parameters:
        - Force the usage of legacy Frontend of Model Optimizer for model conversion into IR:   False
        - Force the usage of new Frontend of Model Optimizer for model conversion into IR:      False
OpenVINO runtime found in:      c:\users\jingc\openvino_env\lib\site-packages\openvino
OpenVINO runtime version:       2022.2.0-7713-af16ea1d79a-releases/2022/2
Model Optimizer version:        2022.2.0-7713-af16ea1d79a-releases/2022/2
[ SUCCESS ] Generated IR version 11 model.
[ SUCCESS ] XML file: C:\Users\jingc\model\yolov6n_base.xml
[ SUCCESS ] BIN file: C:\Users\jingc\model\yolov6n_base.bin
[ SUCCESS ] Total execution time: 2.22 seconds.
```

Figure 5.6 Converting ONNX format (.ONNX*) to IR format (.xml* + .bin*) using OpenVino's model optimizer

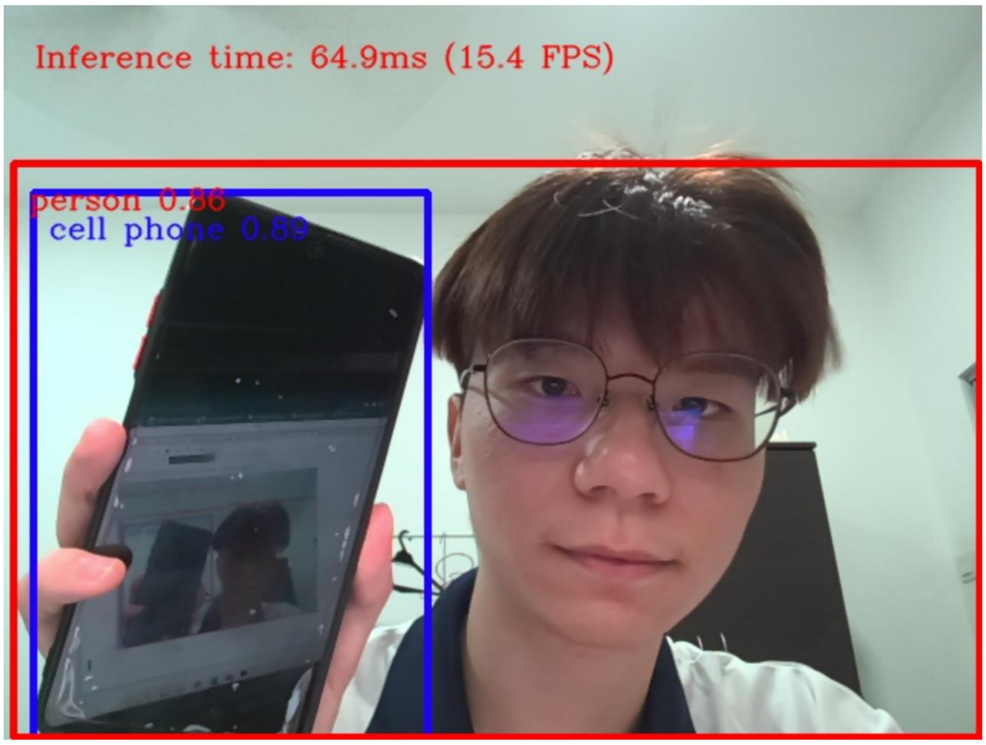


Figure 5.7 Result of original ONNX YOLOv6n pre-trained model

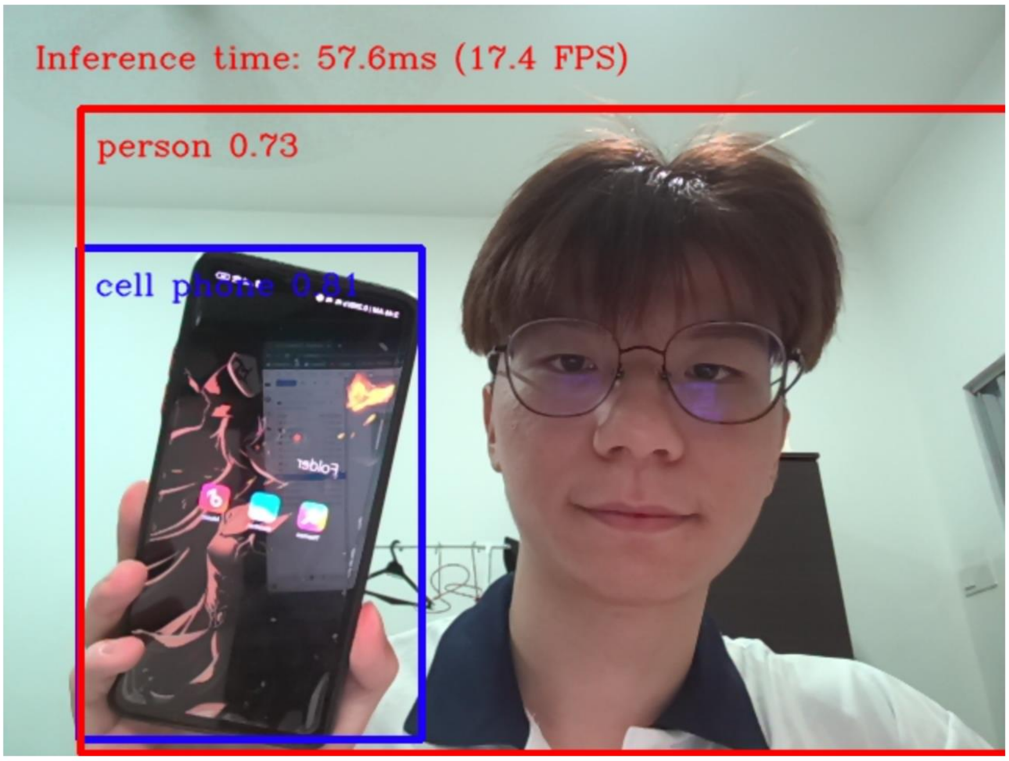


Figure 5.8 Result of OpenVino-Optimized (ONNX-to-IR) YOLOv6n pre-trained model

5.2.3 Measurement for Pytorch Pre-trained Model (YOLOv6n)

We used the same source from [9] of YOLOv6 pretrained model (yolov6n base.pt) for the measurement process of the Pytorch model. We implement the conversion from Pytorch to ONNX format before only converting to IR format because OpenVino Model Optimizer does not support the conversion from Pytorch to IR format. We first export our Pytorch model to an ONNX model using the method provided by [10]. The ONNX file is converted to IR file using the model optimizer in OpenVino. And after the comparison process, we discover that the original Pytorch model result is displayed in figure 5.10 and the OpenVino-optimized result is shown in figure 5.12. The Yolov6n Pytorch pre-trained model released by Meituan has improved its performance by about 1.8x (from 9.7FPS to 17.7FPS) after being optimized by OpenVino. Additionally, we also tested the processed FPS of the converted Pytorch to ONNX model in figure 5.11.

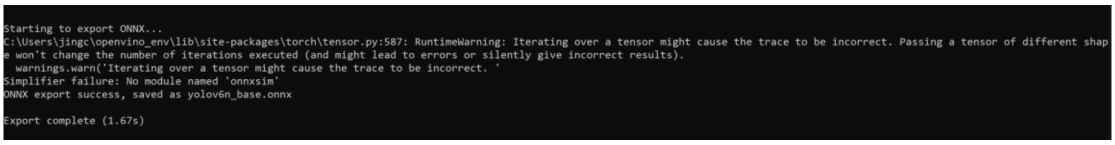

```
Starting to export ONNX...
C:\Users\jingc\openvino_env\lib\site-packages\torch\tensor.py:587: RuntimeWarning: Iterating over a tensor might cause the trace to be incorrect. Passing a tensor of different shap
e won't change the number of iterations executed (and might lead to errors or silently give incorrect results).
  warnings.warn('Iterating over a tensor might cause the trace to be incorrect. '
Simplifier failure: No module named 'onnxsim'
ONNX export success, saved as yolov6n_base.onnx

Export complete (1.67s)
```

Figure 5.9 Exporting Pytorch yolov6 model to ONNX model

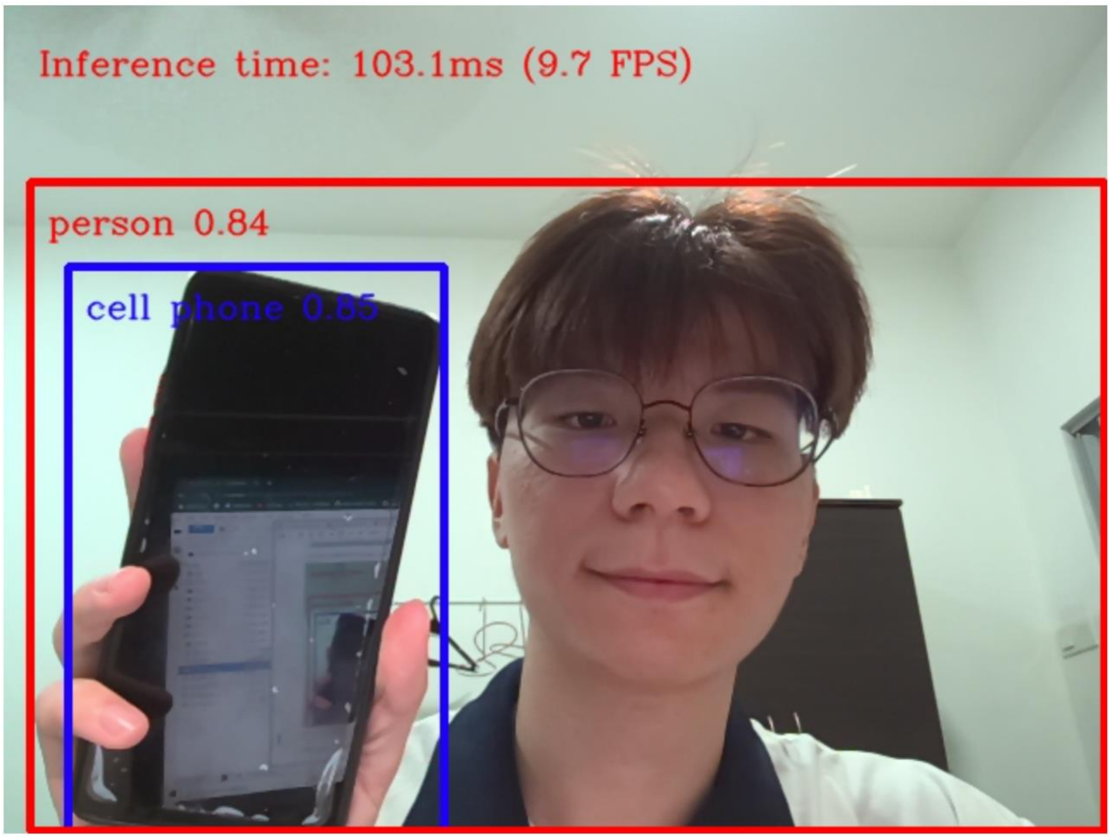


Figure 5.10 Result of original Pytorch YOLOv6n pre-trained model

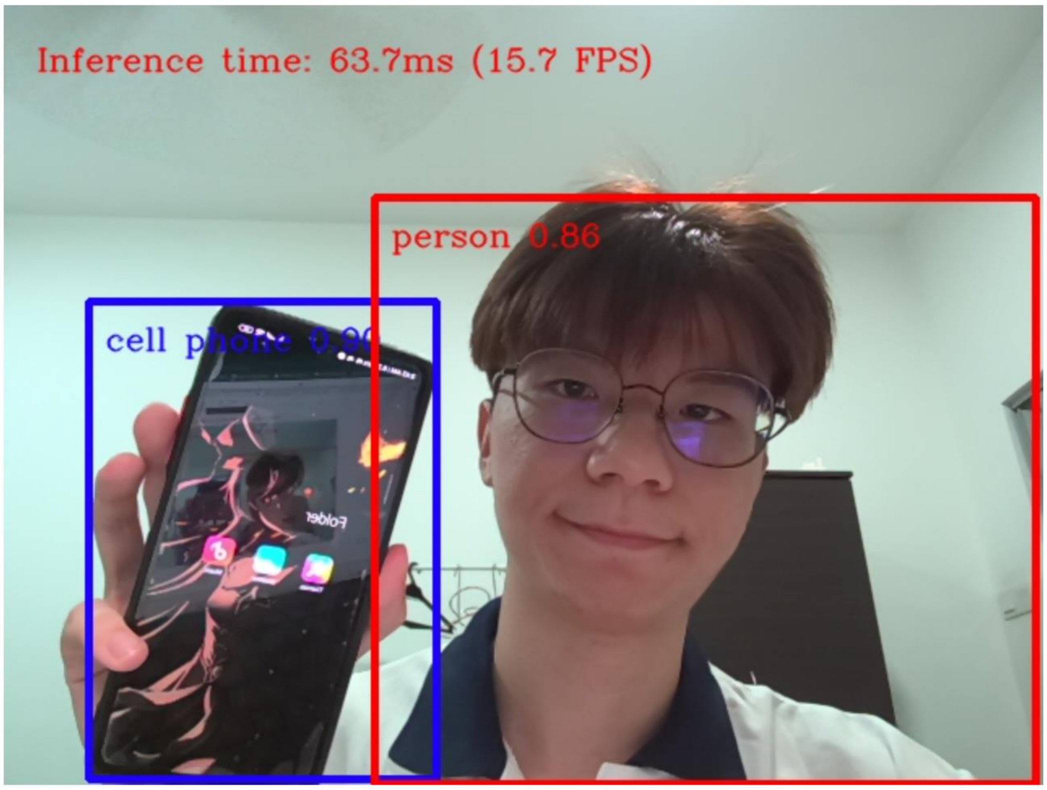


Figure 5.11 Result of (Pytorch-to-ONNX) YOLOv6n pre-trained model

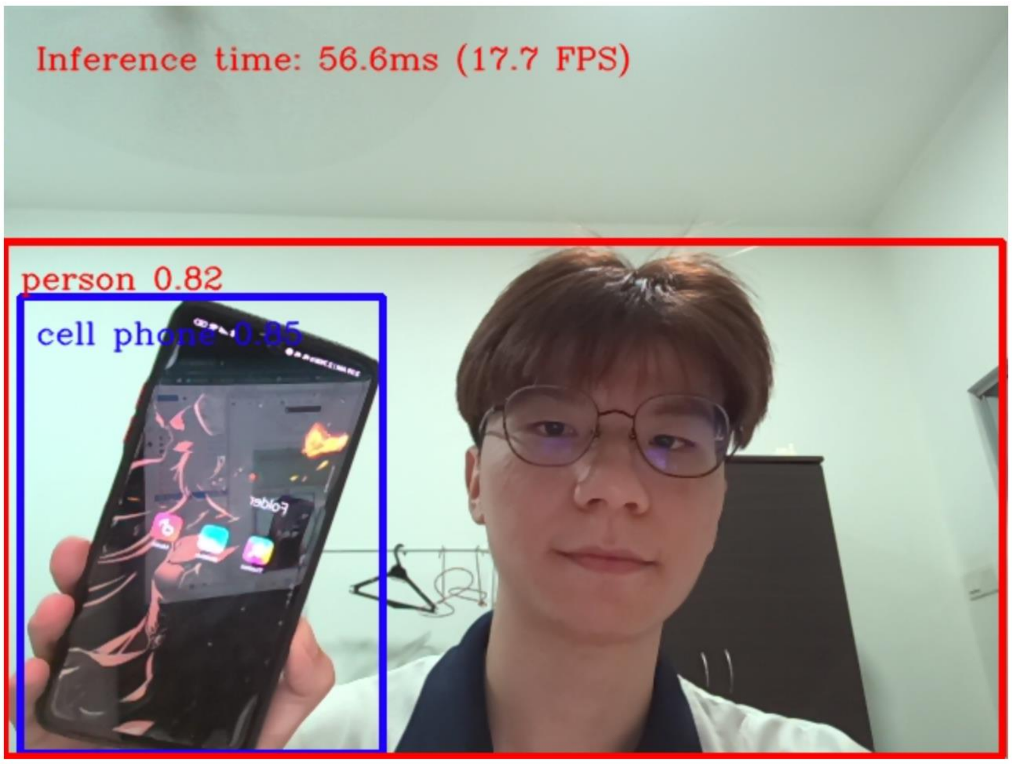


Figure 5.12 Result of OpenVino-Optimized (Pytorch-to-ONNX-to-IR) YOLOv6n pre-trained model

5.2.4 Overview of the Performance measurement

The Overview of performance introduced by OpenVino for the three different deep learning frameworks, TensorFlow, ONNX, and Pytorch, is shown in Table5.1. In this stage, two types of DNN models are implemented. The first DNN model is called Mobilev2 SSDlite, and TensorFlow (improves 204%) is engaged to run this model, it is discovered that it is the best outcome out of all the others. However, since this model is officially offered on the OpenVino website, it's possible that OpenVino has well-optimized it to achieve good results. As a result, we tested YOLOv6, a recently released and well-liked DNNs framework that is not offered on the official OpenVino website. The outcome reveals that ONNX improves 20% while Pytorch improves 80%.

Table 5.1 Overview of the Performance Measurement Result

DL Model and (DNNs)	FPS of Original Model	FPS of OpenVino-Optimised Model	Obtained improvement (x times)
TensorFlow (Mobilev2_SSDlite)	23.5	71.5	3.04x
ONNX (YOLOv6)	15.4	17.4	1.2x
Pytorch (YOLOv6)	9.7 (Pytorch)	-	1.8x
	15.7 (Converted ONNX model)	17.7	

Additionally, the model optimizer for OpenVino is not directly supported by Pytorch. So instead, it must be changed from Pytorch to ONNX format and finally from ONNX to IR format. Therefore, we converted Pytorch to ONNX using the official Yolov6 website converter (https://github.com/meituan/YOLOv6/tree/main/deploy/ONNX). and we put the modified Pytorch to ONNX model to the test. It achieves a similar outcome to the original ONNX model (which around 15 FPS). Besides, the results of converting either the Pytorch format to IR format or the ONNX model to IR format are exactly equivalent (both Pytorch-ONNX-IR and ONNX-IR get around 17 FPS).

5.3 System Implementation for Objective 2: Configuration of Traffic Violation Detection.

Below is the Flow Diagram of Traffic Violation Detection System, where the necessary functions are labeled beside the description in each box. The components involved in each process will be discussed in detail in the subsequent sections of chapter 5.3.

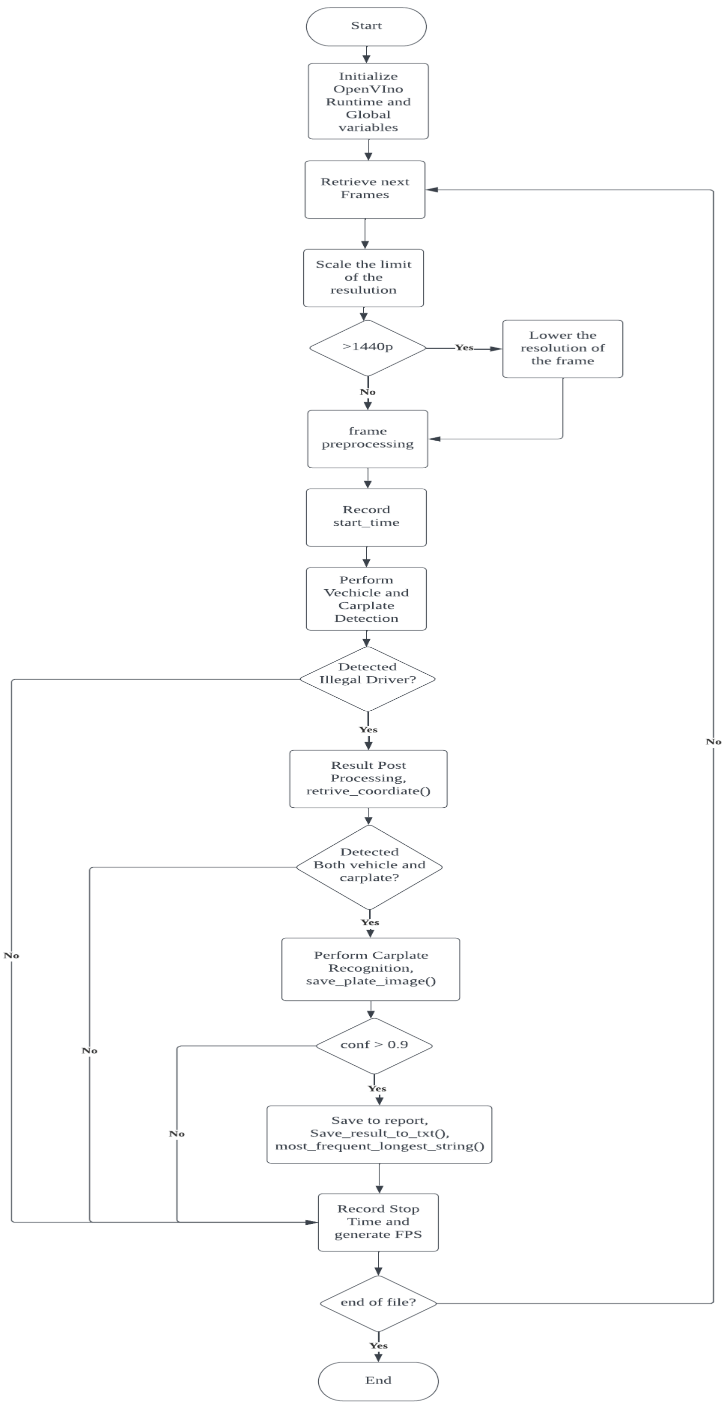
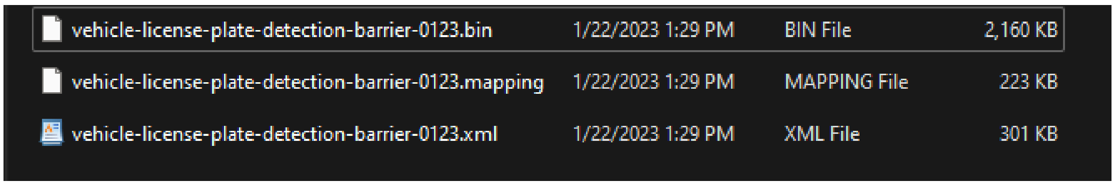

Start
Initialize OpenVIno Runtime and Global variables
Retrieve next Frames
Scale the limit of the resulution
>1440p
Yes
Lower the resolution of the frame
No
frame preprocessing
Record start_time
Perform Vechicle and Carplate Detection
Detected Illegal Driver?
Yes
No
Result Post Processing, retrive_coordiate()
Detected Both vehicle and carplate?
Yes
No
Perform Carplate Recognition, save_plate_image()
conf > 0.9
Yes
No
Save to report, Save_result_to_txt(), most_frequent_longest_string()
Record Stop Time and generate FPS
end of file?
No
Yes
End

Figure 5.13 Flow Diagram of Traffic Violation Detection System

5.3.1 Vehicle-license-plate-detection-barrier-0123

The IR model "Vehicle-license-plate-detection-barrier-0123" was downloaded from the Intel GitHub website Open Model Zoo. The model had already been converted to IR files from the base TensorFlow framework. As we have already determined that the TensorFlow model provides optimal performance, we will be using this model for vehicle and car plate detection in our Traffic Violation Detection System. The primary purpose of this model is to detect **front-facing** cars and their license plates. Once the model was downloaded, it was placed in the project directory for further use

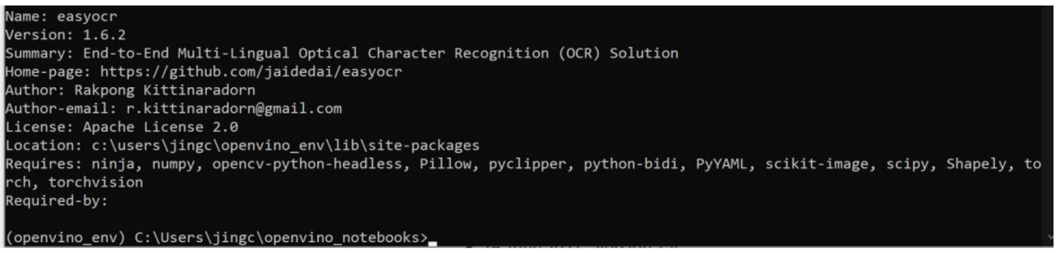


```
vehicle-license-plate-detection-barrier-0123.bin       1/22/2023 1:29 PM   BIN File       2,160 KB
vehicle-license-plate-detection-barrier-0123.mapping   1/22/2023 1:29 PM   MAPPING File     223 KB
vehicle-license-plate-detection-barrier-0123.xml       1/22/2023 1:29 PM   XML File         301 KB
```

Figure 5.14 Vehicle-license-plate-detection-barrier-0123 model (IR files)

5.3.2 Library installation (EasyOCR)

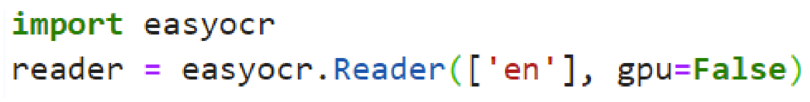
EasyOCR was selected for the car plate recognition in this Traffic Violation Detection System. EasyOCR was chosen for its high accuracy and ease of implementation. The Documentation of Easy OCR is at [11] Below is the version installed:



```
Name: easyocr
Version: 1.6.2
Summary: End-to-End Multi-Lingual Optical Character Recognition (OCR) Solution
Home-page: https://github.com/jaidedai/easyocr
Author: Rakpong Kittinaradorn
Author-email: r.kittinaradorn@gmail.com
License: Apache License 2.0
Location: c:\users\jingc\openvino_env\lib\site-packages
Requires: ninja, numpy, opencv-python-headless, Pillow, pyclipper, python-bidi, PyYAML, scikit-image, scipy, Shapely, to
rch, torchvision
Required-by:

(openvino_env) C:\Users\jingc\openvino_notebooks>_
```

Figure 5.15 Version of EasyOCR

We chose to use an English character-specific model for optimal performance as it allowed for a more streamlined and accurate car plate recognition process, given that car plates typically only contain English letters and numerical characters.



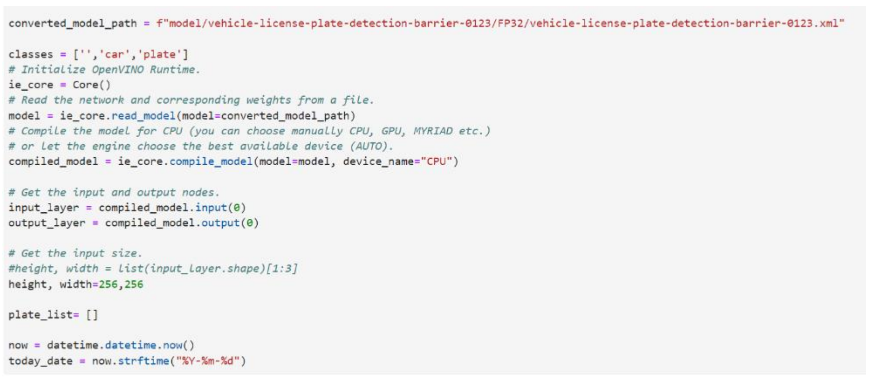
```python
import easyocr
reader = easyocr.Reader(['en'], gpu=False)
```

Figure 5.16 EasyOCR to Detect English Characters

5.3.3 Configuration of Traffic Violation Detection System

1. **Initialize OpenVino Runtime**

   At the beginning of the process, the OpenVino Runtime was initialized, and a plate list was created to store the license plates detected in each frame. The plate list was programmed to be cleared every 30 frames. In order to accommodate the model's required input image size of 256x256, the global variables Height and Width were set to 256. Additionally, the current date was stored in a variable called "Now".

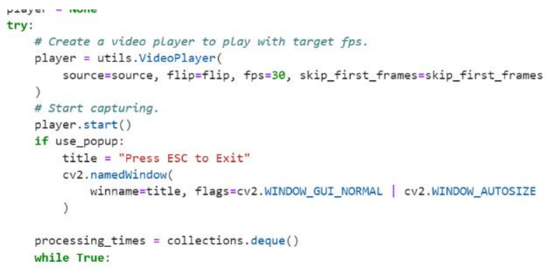


```
converted_model_path = f"model/vehicle-license-plate-detection-barrier-0123/FP32/vehicle-license-plate-detection-barrier-0123.xml"

classes = ['','car','plate']
# Initialize OpenVINO Runtime.
ie_core = Core()
# Read the network and corresponding weights from a file.
model = ie_core.read_model(model=converted_model_path)
# Compile the model for CPU (you can choose manually CPU, GPU, MYRIAD etc.)
# or let the engine choose the best available device (AUTO).
compiled_model = ie_core.compile_model(model=model, device_name="CPU")

# Get the input and output nodes.
input_layer = compiled_model.input(0)
output_layer = compiled_model.output(0)

# Get the input size.
#height, width = list(input_layer.shape)[1:3]
height, width=256,256

plate_list= []

now = datetime.datetime.now()
today_date = now.strftime("%Y-%m-%d")
```

Figure 5.17 OpenVino Runtime Initialization

2. **Start Video Capture**

   During this phase, the system captures video using the specified "source". Additionally, a collection called "processing_time Variable" is created to store the processing time data for further analysis.

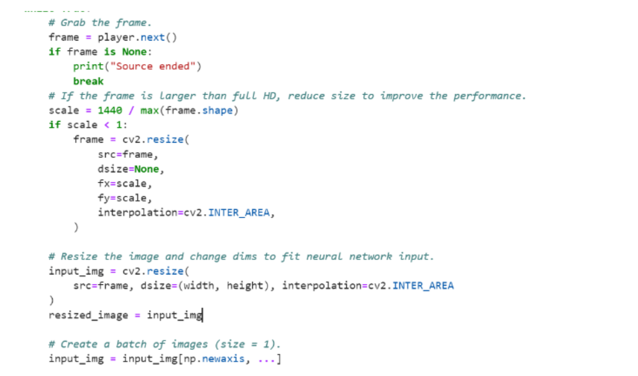


```
try:
    # Create a video player to play with target fps.
    player = utils.VideoPlayer(
        source=source, flip=flip, fps=30, skip_first_frames=skip_first_frames
    )
    # Start capturing.
    player.start()
    if use_popup:
        title = "Press ESC to Exit"
        cv2.namedWindow(
            winname=title, flags=cv2.WINDOW_GUI_NORMAL | cv2.WINDOW_AUTOSIZE
        )

    processing_times = collections.deque()
    while True:
```

Figure 5.18 Start Capturing

3. **<u>Rescaling the frame resolution limit & Pre-processing</u>**

   To optimize performance, the system limits the maximum frame resolution to 1440x1440p. The frames are then reformatted to [Batch,Height,Width,Channel] to match the input requirements of the neural network.

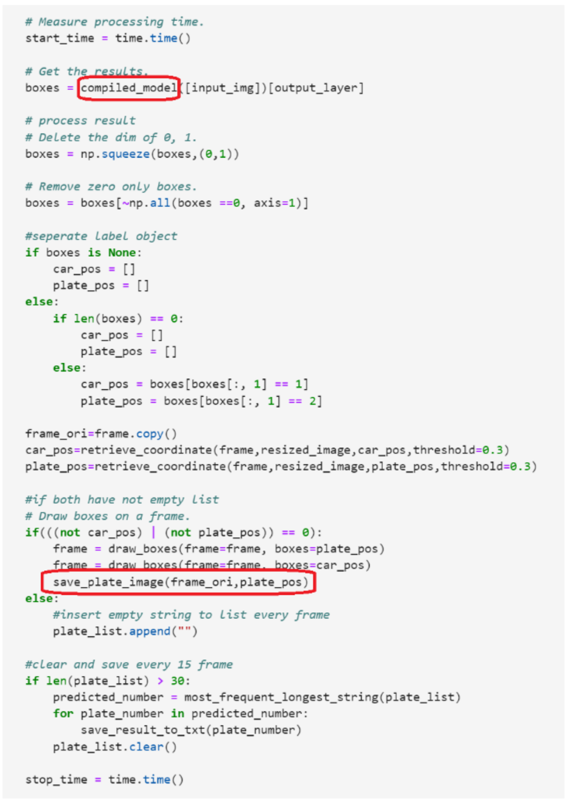


```
    # Grab the frame.
    frame = player.next()
    if frame is None:
        print("Source ended")
        break
    # If the frame is larger than full HD, reduce size to improve the performance.
    scale = 1440 / max(frame.shape)
    if scale < 1:
        frame = cv2.resize(
            src=frame,
            dsize=None,
            fx=scale,
            fy=scale,
            interpolation=cv2.INTER_AREA,
        )

    # Resize the image and change dims to fit neural network input.
    input_img = cv2.resize(
        src=frame, dsize=(width, height), interpolation=cv2.INTER_AREA
    )
    resized_image = input_img

    # Create a batch of images (size = 1).
    input_img = input_img[np.newaxis, ...]
```

Figure 5.19 Rescaling and Pre-processing

4. **<u>Perform Vehicle and Carplate Detection (Main Function)</u>**

   The main function begins by initializing the "start_time" and "end_time" variables to measure the overall performance of the program, which includes front-facing vehicle and car plate detection as well as OCR. The "compiled_model" function is responsible for loading the model using OpenVino. The "Save_plate_image" function contains the OCR implementation.

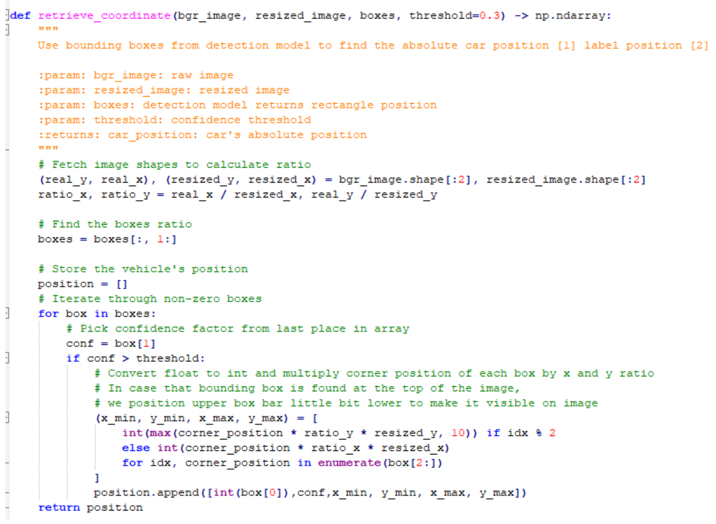


```
# Measure processing time.
start_time = time.time()

# Get the results.
boxes = compiled_model([input_img])[output_layer]

# process result
# Delete the dim of 0, 1.
boxes = np.squeeze(boxes,(0,1))

# Remove zero only boxes.
boxes = boxes[~np.all(boxes ==0, axis=1)]

#seperate label object
if boxes is None:
    car_pos = []
    plate_pos = []
else:
    if len(boxes) == 0:
        car_pos = []
        plate_pos = []
    else:
        car_pos = boxes[boxes[:, 1] == 1]
        plate_pos = boxes[boxes[:, 1] == 2]

frame_ori=frame.copy()
car_pos=retrieve_coordinate(frame,resized_image,car_pos,threshold=0.3)
plate_pos=retrieve_coordinate(frame,resized_image,plate_pos,threshold=0.3)

#if both have not empty list
# Draw boxes on a frame.
if(((not car_pos) | (not plate_pos)) == 0):
    frame = draw_boxes(frame=frame, boxes=plate_pos)
    frame = draw_boxes(frame=frame, boxes=car_pos)
    save_plate_image(frame_ori,plate_pos)
else:
    #insert empty string to list every frame
    plate_list.append("")

#clear and save every 15 frame
if len(plate_list) > 30:
    predicted_number = most_frequent_longest_string(plate_list)
    for plate_number in predicted_number:
        save_result_to_txt(plate_number)
    plate_list.clear()

stop_time = time.time()
```

Figure 5.20 Main Function

5. <u>Function to Retrieve Coordinate (Result Post Processing)</u>

The purpose of this function is to obtain the precise coordinates of the bounding box and then return them in the form of a list.

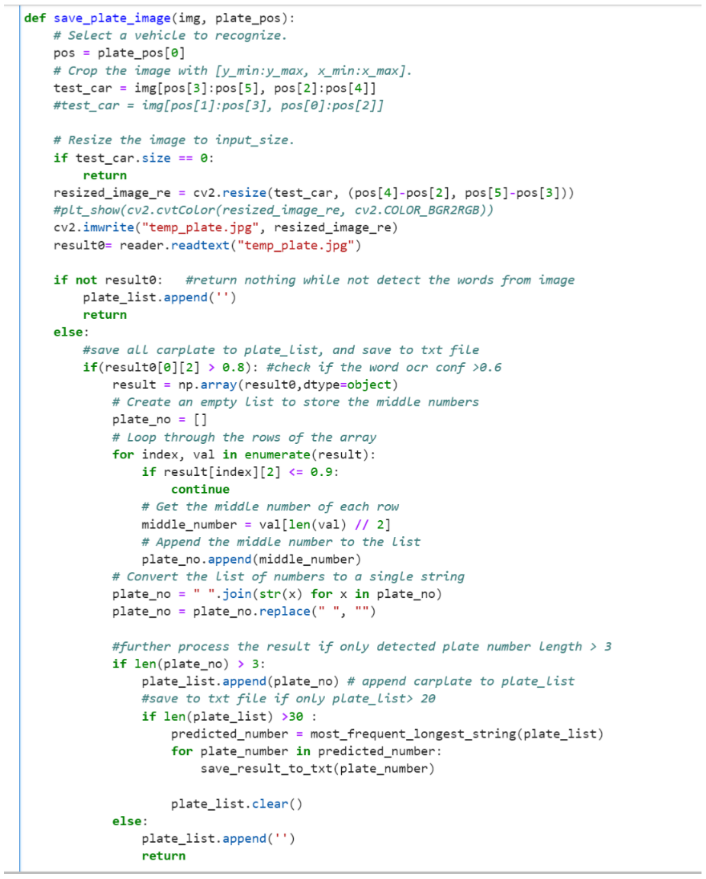


```
def retrieve_coordinate(bgr_image, resized_image, boxes, threshold=0.3) -> np.ndarray:
    """
    Use bounding boxes from detection model to find the absolute car position [1] label position [2]

    :param: bgr_image: raw image
    :param: resized_image: resized image
    :param: boxes: detection model returns rectangle position
    :param: threshold: confidence threshold
    :returns: car_position: car's absolute position
    """
    # Fetch image shapes to calculate ratio
    (real_y, real_x), (resized_y, resized_x) = bgr_image.shape[:2], resized_image.shape[:2]
    ratio_x, ratio_y = real_x / resized_x, real_y / resized_y

    # Find the boxes ratio
    boxes = boxes[:, 1:]

    # Store the vehicle's position
    position = []
    # Iterate through non-zero boxes
    for box in boxes:
        # Pick confidence factor from last place in array
        conf = box[1]
        if conf > threshold:
            # Convert float to int and multiply corner position of each box by x and y ratio
            # In case that bounding box is found at the top of the image,
            # we position upper box bar little bit lower to make it visible on image
            (x_min, y_min, x_max, y_max) = [
                int(max(corner_position * ratio_y * resized_y, 10)) if idx % 2
                else int(corner_position * ratio_x * resized_x)
                for idx, corner_position in enumerate(box[2:])
            ]
            position.append([int(box[0]),conf,x_min, y_min, x_max, y_max])
    return position
```

Figure 5.21 Function of “retrieve_cordinate()”

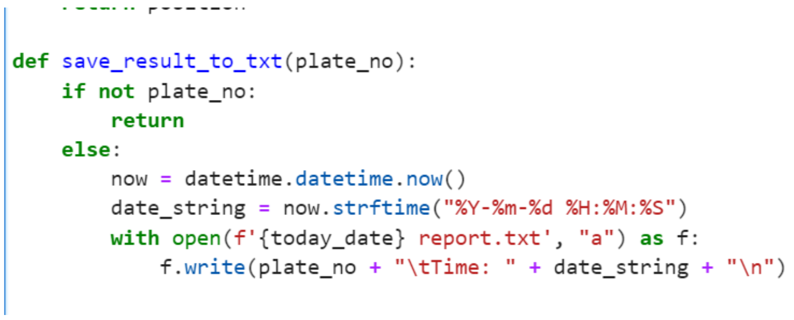
6. **<u>Function to perform Car Plate Number Recognition (save_plate_image(), save_result_to_txt())</u>**

The purpose of these functions is to carry out Optical Character Recognition (OCR) to identify the car plate number. Only results with an accuracy of 80% or higher will be saved to the report file.



```python
def save_plate_image(img, plate_pos):
    # Select a vehicle to recognize.
    pos = plate_pos[0]
    # Crop the image with [y_min:y_max, x_min:x_max].
    test_car = img[pos[3]:pos[5], pos[2]:pos[4]]
    #test_car = img[pos[1]:pos[3], pos[0]:pos[2]]

    # Resize the image to input_size.
    if test_car.size == 0:
        return
    resized_image_re = cv2.resize(test_car, (pos[4]-pos[2], pos[5]-pos[3]))
    #plt_show(cv2.cvtColor(resized_image_re, cv2.COLOR_BGR2RGB))
    cv2.imwrite("temp_plate.jpg", resized_image_re)
    result0= reader.readtext("temp_plate.jpg")

    if not result0:   #return nothing while not detect the words from image
        plate_list.append('')
        return
    else:
        #save all carplate to plate_list, and save to txt file
        if(result0[0][2] > 0.8): #check if the word ocr conf >0.6
            result = np.array(result0,dtype=object)
            # Create an empty list to store the middle numbers
            plate_no = []
            # Loop through the rows of the array
            for index, val in enumerate(result):
                if result[index][2] <= 0.9:
                    continue
                # Get the middle number of each row
                middle_number = val[len(val) // 2]
                # Append the middle number to the list
                plate_no.append(middle_number)
            # Convert the list of numbers to a single string
            plate_no = " ".join(str(x) for x in plate_no)
            plate_no = plate_no.replace(" ", "")

            #further process the result if only detected plate number length > 3
            if len(plate_no) > 3:
                plate_list.append(plate_no) # append carplate to plate_list
                #save to txt file if only plate_list> 20
                if len(plate_list) >30 :
                    predicted_number = most_frequent_longest_string(plate_list)
                    for plate_number in predicted_number:
                        save_result_to_txt(plate_number)

                    plate_list.clear()
            else:
                plate_list.append('')
                return
```

Figure 5.22 Function of “save_plate_image()”

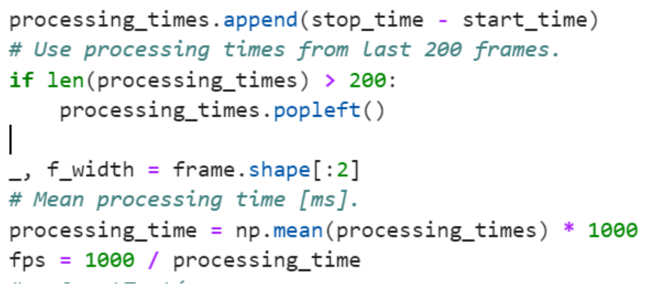


```python
def save_result_to_txt(plate_no):
    if not plate_no:
        return
    else:
        now = datetime.datetime.now()
        date_string = now.strftime("%Y-%m-%d %H:%M:%S")
        with open(f'{today_date} report.txt', "a") as f:
            f.write(plate_no + "\tTime: " + date_string + "\n")
```

Figure 5.23 Function of “save_result_to_txt()”

7. **<u>Generate FPS (End of Main)</u>**

This step involves generating the average processed frames per second (FPS) to evaluate the performance of the system.



```
processing_times.append(stop_time - start_time)
# Use processing times from last 200 frames.
if len(processing_times) > 200:
    processing_times.popleft()

_, f_width = frame.shape[:2]
# Mean processing time [ms].
processing_time = np.mean(processing_times) * 1000
fps = 1000 / processing_time
```

Figure 5.24 Generate Average Processed FPS

5.4. Result of the Traffic Violation Detection System

Below is the sample result of the application in different condition:

1. Front-facing Car (illegal driving) will be detected

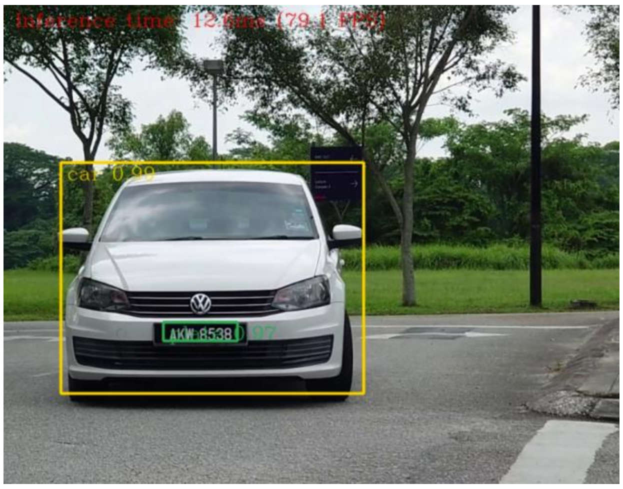


Figure 5.25 Front-Facing (illegal driving) white car

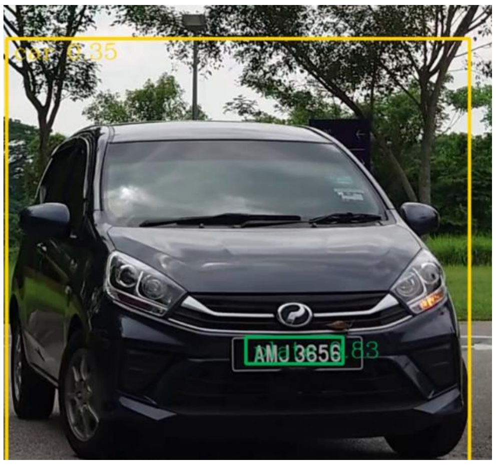


Figure 5.26 Front-facing (Illegal Driving) Black Car

2. Back-facing (legal driving) car will not be detected

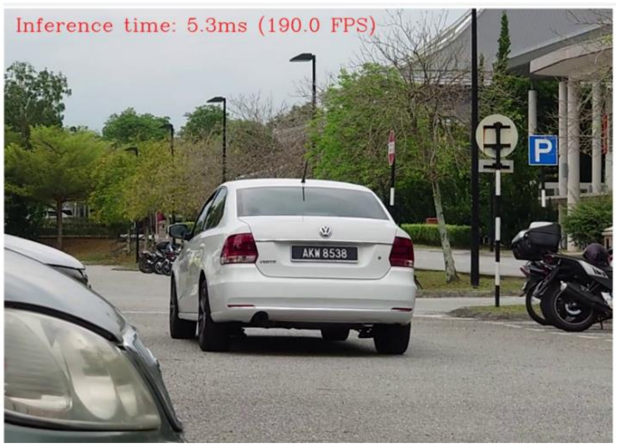


Figure 5.27 Back Facing (legal Driving) Back-facing Car

3. Front and back facing car: only front-facing car will be detected

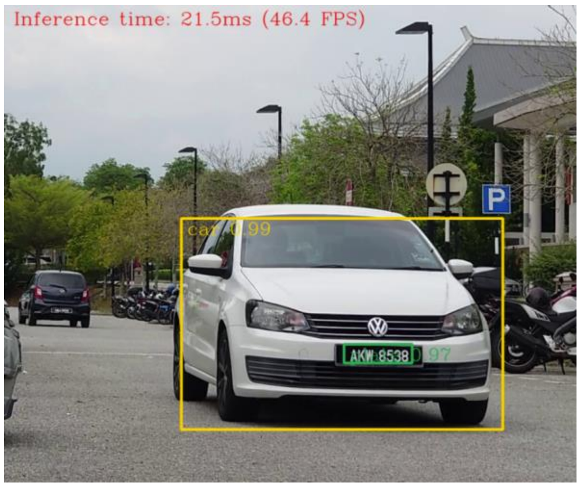
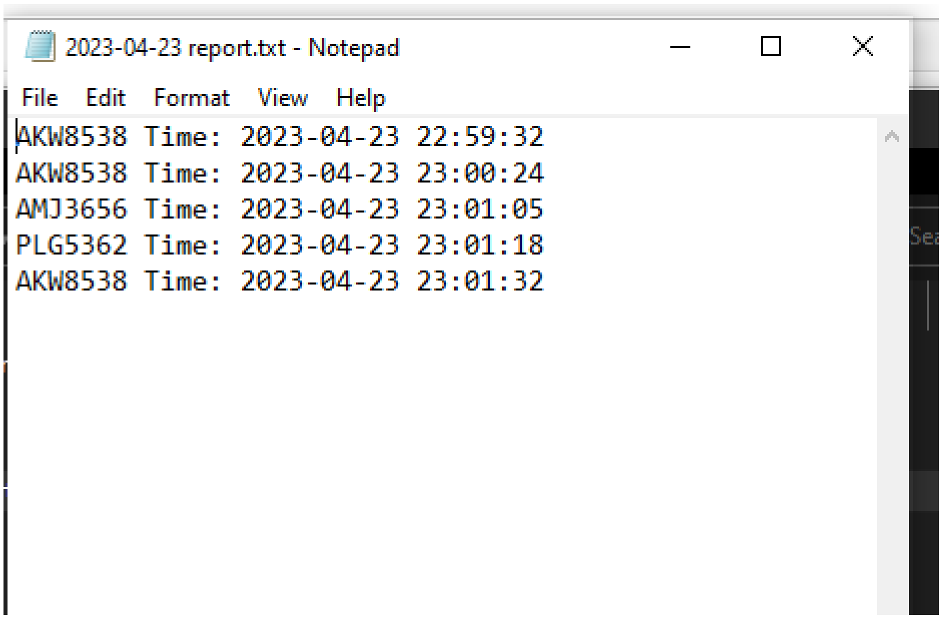


Figure5.28 Front and Back Facing Car

4. Text File as a Daily Report in the System, date and time is recorded



```
AKW8538 Time: 2023-04-23 22:59:32
AKW8538 Time: 2023-04-23 23:00:24
AMJ3656 Time: 2023-04-23 23:01:05
PLG5362 Time: 2023-04-23 23:01:18
AKW8538 Time: 2023-04-23 23:01:32
```

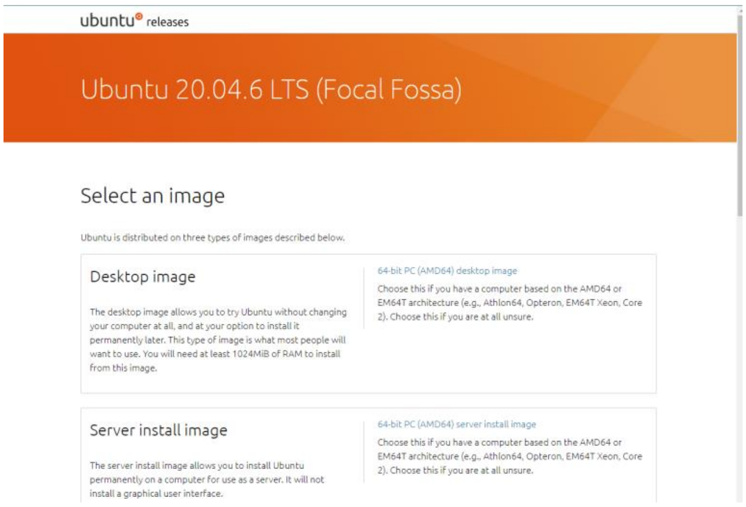
Figure 5.29 Daily report.txt

5.4 Hardware Setup (Intel UP Board)

5.1.1 OS Installation

To complete the installation process, you will require a USB drive, a monitor, a keyboard, a mouse, an Ethernet cable connected to the Intel UP Board.

1. Download Ubuntu Desktop 20.04 LTS from Ubuntu Official Website (https://ubuntu.com/download/desktop)



ubuntu® releases

Ubuntu 20.04.6 LTS (Focal Fossa)

Select an image

Ubuntu is distributed on three types of images described below.

Desktop image

The desktop image allows you to try Ubuntu without changing your computer at all, and at your option to install it permanently later. This type of image is what most people will want to use. You will need at least 1024MiB of RAM to install from this image.

64-bit PC (AMD64) desktop image

Choose this if you have a computer based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron, EM64T Xeon, Core 2). Choose this if you are at all unsure.

Server install image

The server install image allows you to install Ubuntu permanently on a computer for use as a server. It will not install a graphical user interface.

64-bit PC (AMD64) server install image

Choose this if you have a computer based on the AMD64 or EM64T architecture (e.g., Athlon64, Opteron, EM64T Xeon, Core 2). Choose this if you are at all unsure.

Figure 5.30 Ubuntu Desktop Image Download

2. Create bootable USB Drive with the Ubuntu Image using Rufus and then connect the USB Drive to Intel UP Board and boot Ubuntu from the USB Drive.

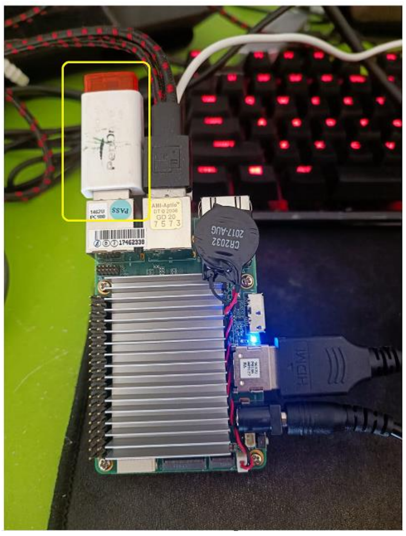
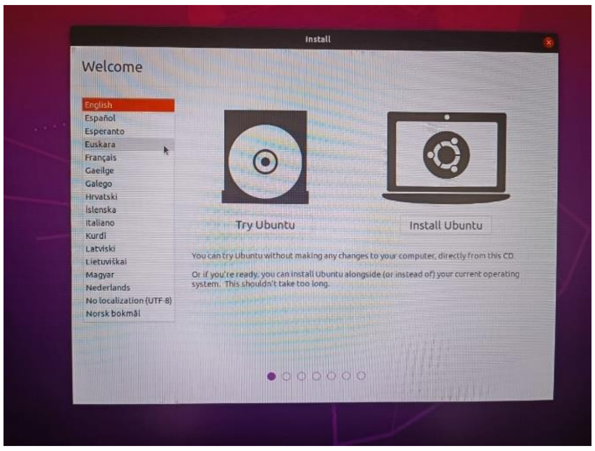


figure 5.31 Bootable Drive with Intel UP Board

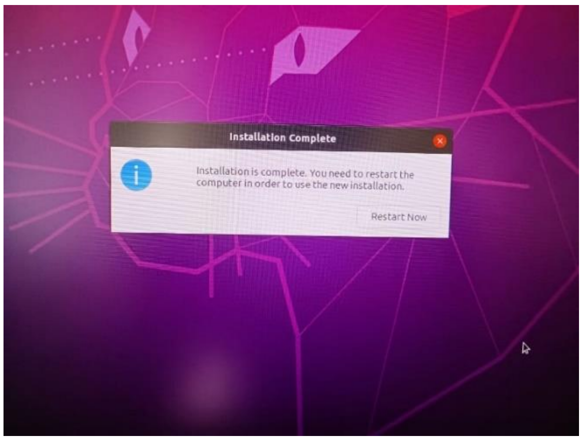
3. After booting from the USB drive, click on the "Install Ubuntu" option to initiate the installation process of Ubuntu OS onto the internal storage of the Intel UP Board. Follow the on-screen instructions to complete the installation process.



You can try Ubuntu without making any changes to your computer, directly from this CD.

Or if you're ready, you can install Ubuntu alongside (or instead of) your current operating system. This shouldn't take too long.

Figure 5.32 Install Ubuntu



Installation is complete. You need to restart the computer in order to use the new installation.

Figure 5.33 (Ubuntu) Installation Complete

4. Click “restart” and unplug the USB Drive. Check if the ubuntu is successfully install. As you can now see the ubuntu is installed in System Partition for booting Ubuntu OS.

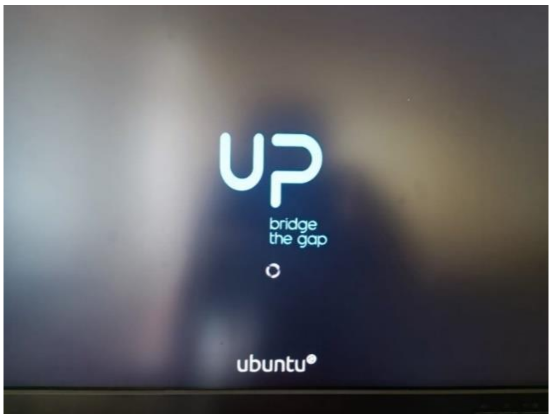
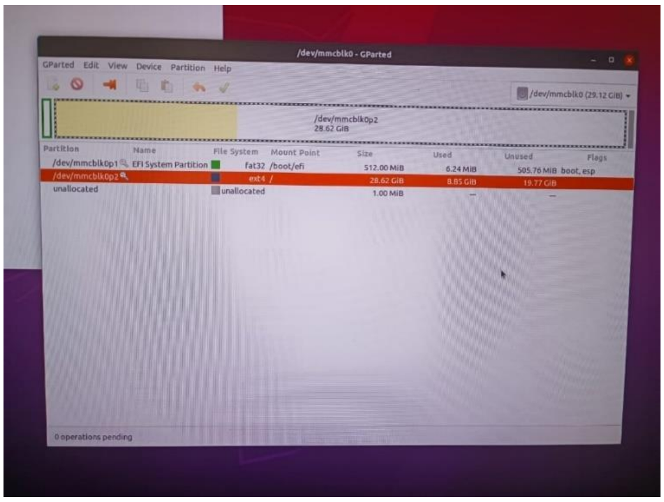


Figure 5.34 Booting Ubuntu on Edge Device without Bootable Drive



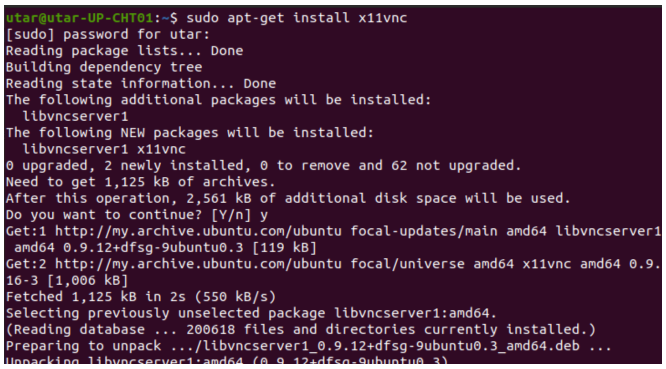
Partition	Name	File System	Mount Point	Size	Used	Unused	Flags
/dev/mmcblk0p1	EFI System Partition	🟩 fat32	/boot/efi	512.00 MiB	6.24 MiB	505.76 MiB	boot, esp
/dev/mmcblk0p2		🟦 ext4	/	28.62 GiB	8.85 GiB	19.77 GiB	
unallocated		▨ unallocated		1.00 MiB	—	—	

0 operations pending

Figure 5.35 System Partition (Flag: boot)

5.1.2 Installation of VNC Server (Accessing to Edge Device without Monitor)

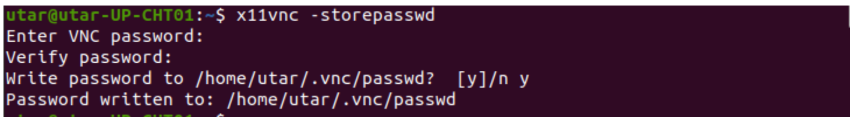
1. Open Terminal and type “sudo apt-get install x11vnc” to install VNC server



```
utar@utar-UP-CHT01:~$ sudo apt-get install x11vnc
[sudo] password for utar:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libvncserver1
The following NEW packages will be installed:
  libvncserver1 x11vnc
0 upgraded, 2 newly installed, 0 to remove and 62 not upgraded.
Need to get 1,125 kB of archives.
After this operation, 2,561 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://my.archive.ubuntu.com/ubuntu focal-updates/main amd64 libvncserver1
 amd64 0.9.12+dfsg-9ubuntu0.3 [119 kB]
Get:2 http://my.archive.ubuntu.com/ubuntu focal/universe amd64 x11vnc amd64 0.9.
16-3 [1,006 kB]
Fetched 1,125 kB in 2s (550 kB/s)
Selecting previously unselected package libvncserver1:amd64.
(Reading database ... 200618 files and directories currently installed.)
Preparing to unpack .../libvncserver1_0.9.12+dfsg-9ubuntu0.3_amd64.deb ...
Unpacking libvncserver1:amd64 (0.9.12+dfsg-9ubuntu0.3)
```

Figure 5.36 VNC Server Installation

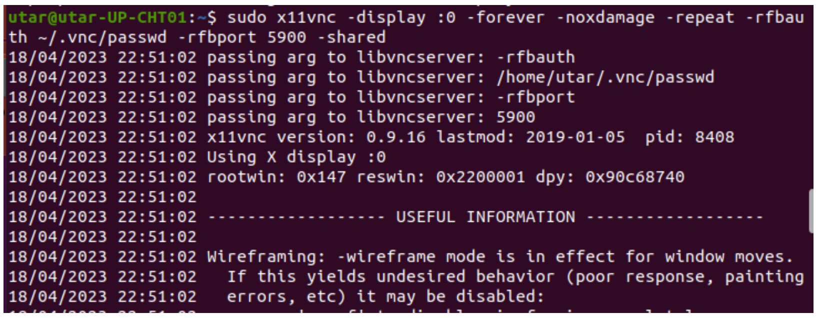
2. Set password to VNC server using “x11vnc -storepasswd”



```
utar@utar-UP-CHT01:~$ x11vnc -storepasswd
Enter VNC password:
Verify password:
Write password to /home/utar/.vnc/passwd?  [y]/n y
Password written to: /home/utar/.vnc/passwd
```

Figure 5.37 Set Password to VNC Server

3. Activate server using this command



```
utar@utar-UP-CHT01:~$ sudo x11vnc -display :0 -forever -noxdamage -repeat -rfbau
th ~/.vnc/passwd -rfbport 5900 -shared
18/04/2023 22:51:02 passing arg to libvncserver: -rfbauth
18/04/2023 22:51:02 passing arg to libvncserver: /home/utar/.vnc/passwd
18/04/2023 22:51:02 passing arg to libvncserver: -rfbport
18/04/2023 22:51:02 passing arg to libvncserver: 5900
18/04/2023 22:51:02 x11vnc version: 0.9.16 lastmod: 2019-01-05  pid: 8408
18/04/2023 22:51:02 Using X display :0
18/04/2023 22:51:02 rootwin: 0x147 reswin: 0x2200001 dpy: 0x90c68740
18/04/2023 22:51:02
18/04/2023 22:51:02 ------------------ USEFUL INFORMATION ------------------
18/04/2023 22:51:02
18/04/2023 22:51:02 Wireframing: -wireframe mode is in effect for window moves.
18/04/2023 22:51:02   If this yields undesired behavior (poor response, painting
18/04/2023 22:51:02   errors, etc) it may be disabled:
```

Figure 5.38 Activate VNC server

4. To access the Edge device from your own device, ensure that both devices are connected to the same local network. Then, use the VNC client on your device and enter the IP address of the Edge device to establish the connection.

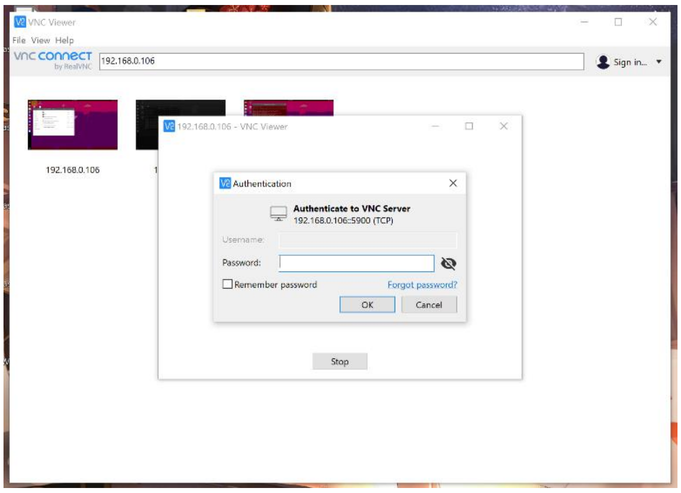


Figure 5.39 Remote Desktop Connection to Edge Device 1

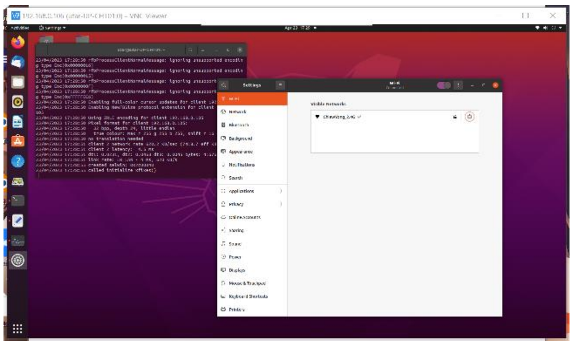
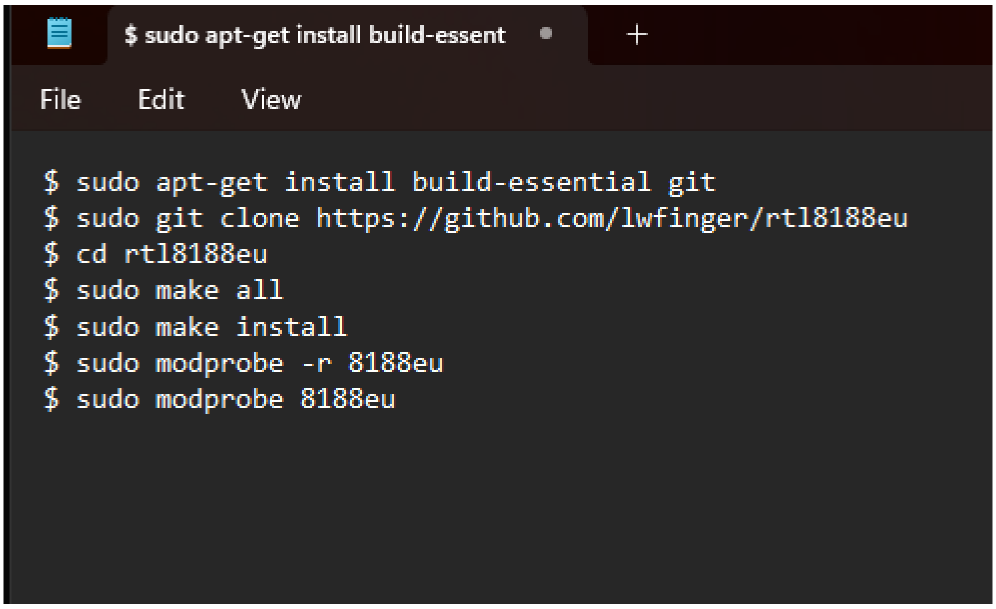


Figure 5.40 Remote Desktop Connection to Edge Device 2

5.1.3 Driver Installation for D-Link DWA-123 (Accessing Internet without RJ45 Ethernet cable)

For this project, we connect the Edge device to the internet using a D-Link DWA-123 Wi-Fi USB adapter. To install the necessary driver for the adapter on Ubuntu 20.04 LTS, we follow the instructions provided on GitHub at the [12].



```
$ sudo apt-get install build-essential git
$ sudo git clone https://github.com/lwfinger/rtl8188eu
$ cd rtl8188eu
$ sudo make all
$ sudo make install
$ sudo modprobe -r 8188eu
$ sudo modprobe 8188eu
```

Figure 5.41 Command to Install Wi-Fi Adapter’s Driver

5.1.4 Final Product of System Hardware Components.

At the end of the process, the system's components' final output can be observed. It is worth noting that a power bank was used as a portable power supply for the Edge device, and a webcam was utilized as an IoT camera to minimize expenses. However, for optimal performance, it is recommended to use professional equipment.

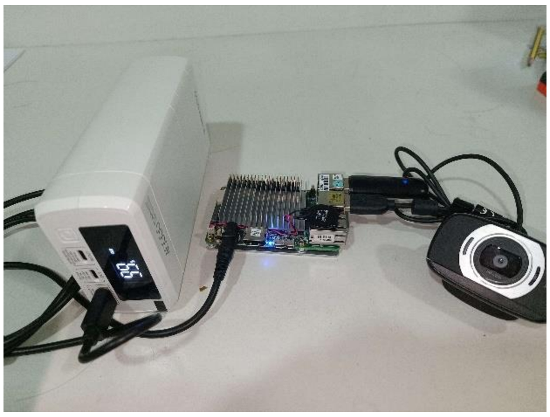
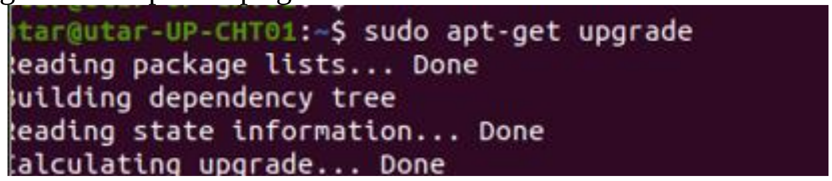


Figure 5.42 Final Product of System Hardware Components

5.5 Software Setup (Intel UP Board)

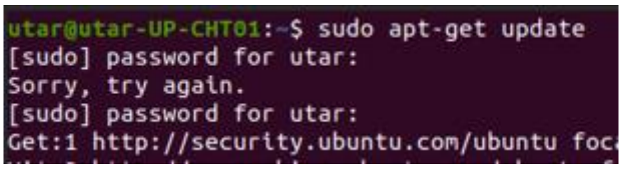
5.5.1 OpenVino Environment

1. Upgrade and update apt-get



```
tar@utar-UP-CHT01:~$ sudo apt-get upgrade
eading package lists... Done
uilding dependency tree
eading state information... Done
alculating upgrade... Done
```

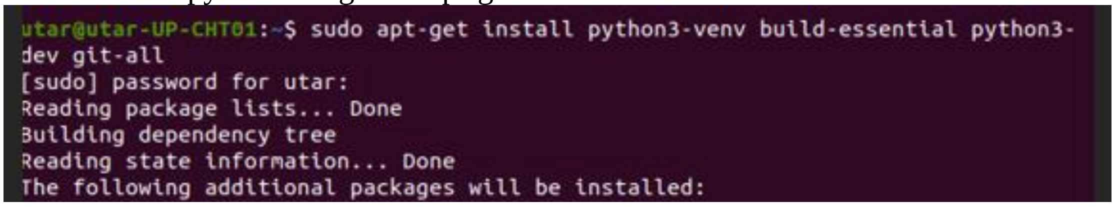
Figure 5.43 Upgrade apt-get



```
utar@utar-UP-CHT01:~$ sudo apt-get update
[sudo] password for utar:
Sorry, try again.
[sudo] password for utar:
Get:1 http://security.ubuntu.com/ubuntu foc
```

Figure 5.44 Update apt-get

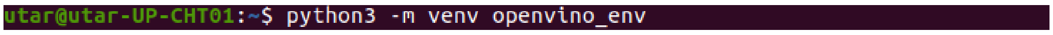
2. Install python and git via apt-get



```
utar@utar-UP-CHT01:~$ sudo apt-get install python3-venv build-essential python3-
dev git-all
[sudo] password for utar:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
```

Figure 5.45 Install Git and Python Environment

3. Create OpenVino enviroment as Project Directory

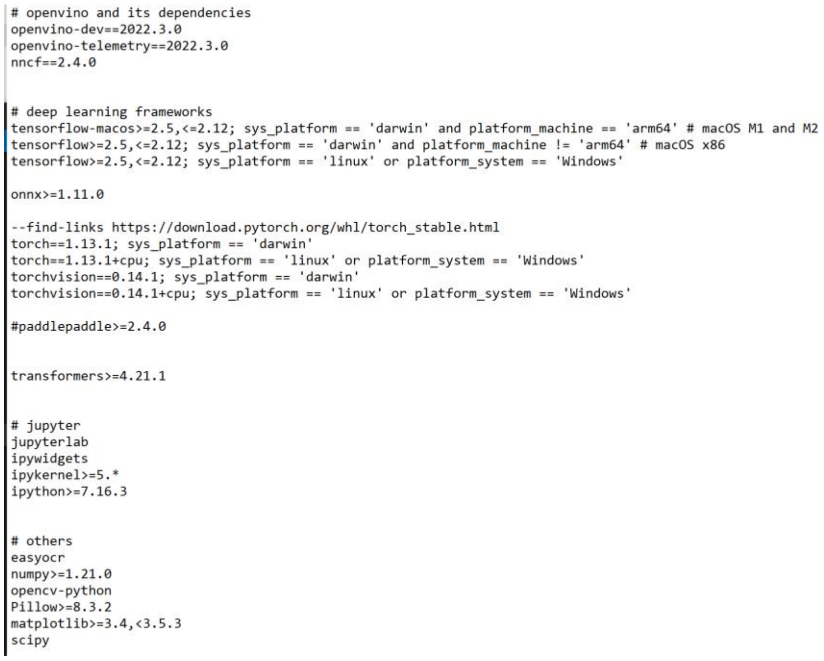


```
utar@utar-UP-CHT01:~$ python3 -m venv openvino_env
```

Figure 5.46 Create OpenVino Environment

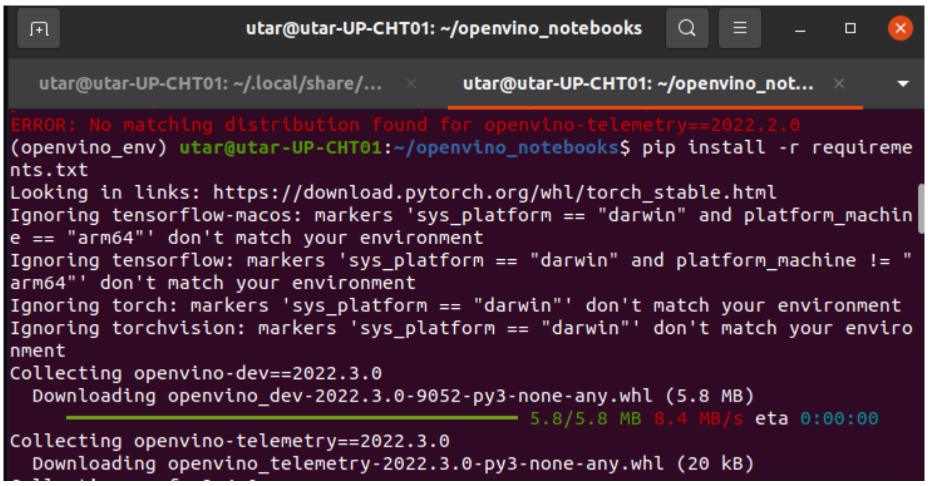
5.5.2 Install all dependencies and required libraries.

1. Install all requirement libraries and dependencies from the below text file (requirement.txt) using “pip -r requirements.txt”. The OpenVino we install is differ from our workstation since Linux only have the version of 2022.3.



```
# openvino and its dependencies
openvino-dev==2022.3.0
openvino-telemetry==2022.3.0
nncf==2.4.0

# deep learning frameworks
tensorflow-macos>=2.5,<=2.12; sys_platform == 'darwin' and platform_machine == 'arm64' # macOS M1 and M2
tensorflow>=2.5,<=2.12; sys_platform == 'darwin' and platform_machine != 'arm64' # macOS x86
tensorflow>=2.5,<=2.12; sys_platform == 'linux' or platform_system == 'Windows'

onnx>=1.11.0

--find-links https://download.pytorch.org/whl/torch_stable.html
torch==1.13.1; sys_platform == 'darwin'
torch==1.13.1+cpu; sys_platform == 'linux' or platform_system == 'Windows'
torchvision==0.14.1; sys_platform == 'darwin'
torchvision==0.14.1+cpu; sys_platform == 'linux' or platform_system == 'Windows'

#paddlepaddle>=2.4.0


transformers>=4.21.1


# jupyter
jupyterlab
ipywidgets
ipykernel>=5.*
ipython>=7.16.3


# others
easyocr
numpy>=1.21.0
opencv-python
Pillow>=8.3.2
matplotlib>=3.4,<3.5.3
scipy
```

Figure 5.47 requirement.txt

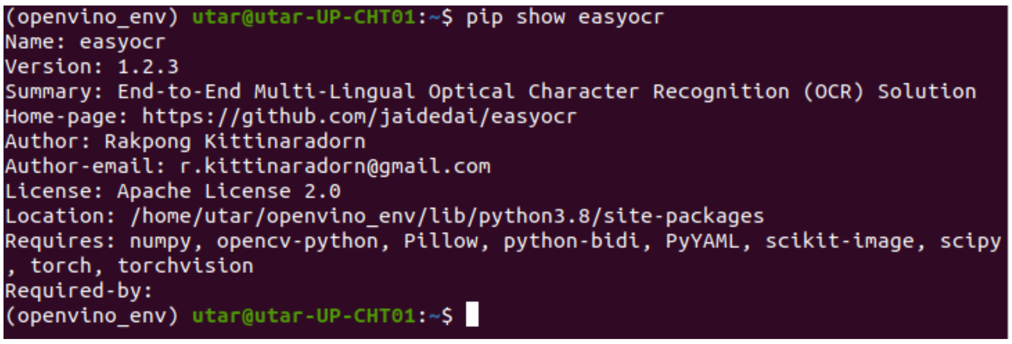


utar@utar-UP-CHT01: ~/openvino_notebooks
utar@utar-UP-CHT01: ~/.local/share/...
utar@utar-UP-CHT01: ~/openvino_not...

```
ERROR: No matching distribution found for openvino-telemetry==2022.2.0
(openvino_env) utar@utar-UP-CHT01:~/openvino_notebooks$ pip install -r requireme
nts.txt
Looking in links: https://download.pytorch.org/whl/torch_stable.html
Ignoring tensorflow-macos: markers 'sys_platform == "darwin" and platform_machin
e == "arm64"' don't match your environment
Ignoring tensorflow: markers 'sys_platform == "darwin" and platform_machine != "
arm64"' don't match your environment
Ignoring torch: markers 'sys_platform == "darwin"' don't match your environment
Ignoring torchvision: markers 'sys_platform == "darwin"' don't match your enviro
nment
Collecting openvino-dev==2022.3.0
  Downloading openvino_dev-2022.3.0-9052-py3-none-any.whl (5.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.8/5.8 MB 8.4 MB/s eta 0:00:00
Collecting openvino-telemetry==2022.3.0
  Downloading openvino_telemetry-2022.3.0-py3-none-any.whl (20 kB)
```

Figure 5.48 Install from requirement.txt

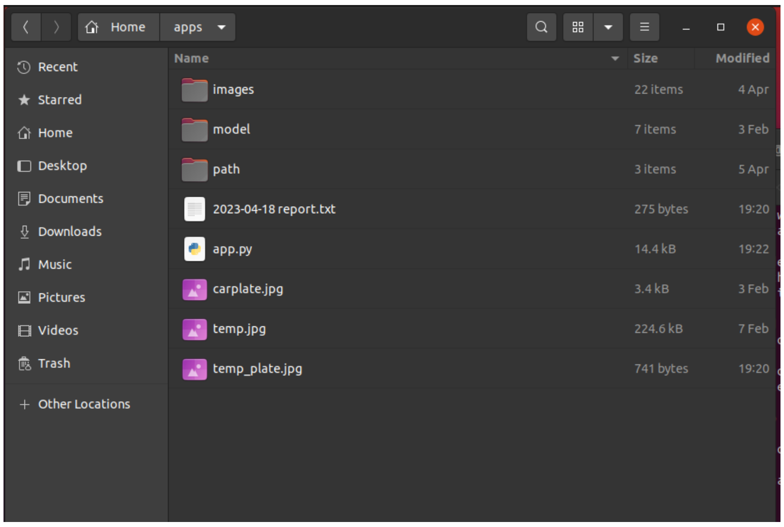
2. Due to the limitation of the processor on my Edge device, it does not support AVX instructions required by the latest version of EasyOCR. To overcome this issue, I have used an older version of EasyOCR, namely 1.2.3, that does not require AVX instructions.



```
(openvino_env) utar@utar-UP-CHT01:~$ pip show easyocr
Name: easyocr
Version: 1.2.3
Summary: End-to-End Multi-Lingual Optical Character Recognition (OCR) Solution
Home-page: https://github.com/jaidedai/easyocr
Author: Rakpong Kittinaradorn
Author-email: r.kittinaradorn@gmail.com
License: Apache License 2.0
Location: /home/utar/openvino_env/lib/python3.8/site-packages
Requires: numpy, opencv-python, Pillow, python-bidi, PyYAML, scikit-image, scipy
, torch, torchvision
Required-by:
(openvino_env) utar@utar-UP-CHT01:~$
```

Figure 5.49 Version of EasyOCR

4. Copy the application directory to edge device.



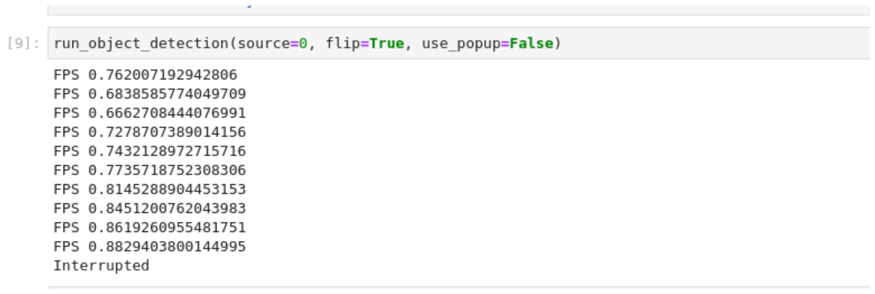
Name	Size	Modified
images	22 items	4 Apr
model	7 items	3 Feb
path	3 items	5 Apr
2023-04-18 report.txt	275 bytes	19:20
app.py	14.4 kB	19:22
carplate.jpg	3.4 kB	3 Feb
temp.jpg	224.6 kB	7 Feb
temp_plate.jpg	741 bytes	19:20

Figure 5.50 Apps Project Directory

5.6 Testing the Effectiveness of OpenVino on an Edge Device

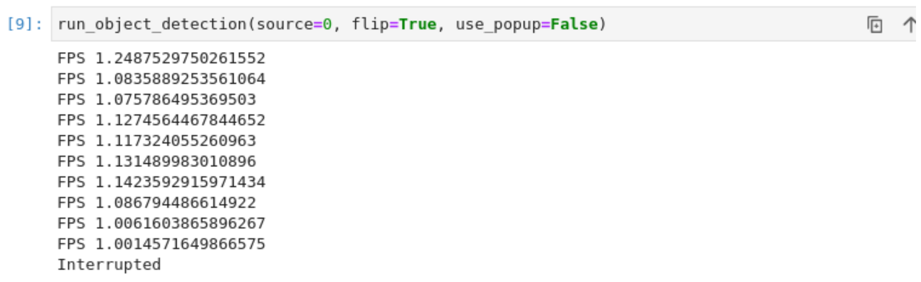
During the testing phase, we utilized the onnx yolov6 object detection model described in chapter 5.2.3 to assess the degree of performance improvement achieved by OpenVino on the edge device.

The experiment conducted to evaluate the effectiveness of OpenVino on the edge device yielded promising results. The original model achieved an average FPS of 0.8, while the OpenVino-optimized model achieved an average FPS of 1.1. This translates to a significant improvement of approximately 37.5% in performance. The findings indicate that OpenVino can effectively optimize models for deployment on edge devices, even with older generation processors.



```
[9]: run_object_detection(source=0, flip=True, use_popup=False)

FPS 0.762007192942806
FPS 0.6838585774049709
FPS 0.6662708444076991
FPS 0.7278707389014156
FPS 0.7432128972715716
FPS 0.7735718752308306
FPS 0.8145288904453153
FPS 0.8451200762043983
FPS 0.8619260955481751
FPS 0.8829403800144995
Interrupted
```

Figure 5.51 Performance of Original ONNX Model on Edge Device

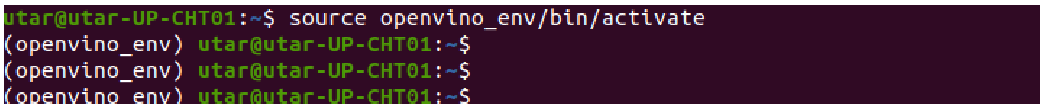


```
[9]: run_object_detection(source=0, flip=True, use_popup=False)

FPS 1.2487529750261552
FPS 1.0835889253561064
FPS 1.075786495369503
FPS 1.1274564467844652
FPS 1.117324055260963
FPS 1.131489983010896
FPS 1.1423592915971434
FPS 1.086794486614922
FPS 1.0061603865896267
FPS 1.0014571649866575
Interrupted
```

Figure 5.52 Performance of OpenVino-optimised Model on Edge Device

5.7 Result of the Traffic Violation Detection System on Edge Device

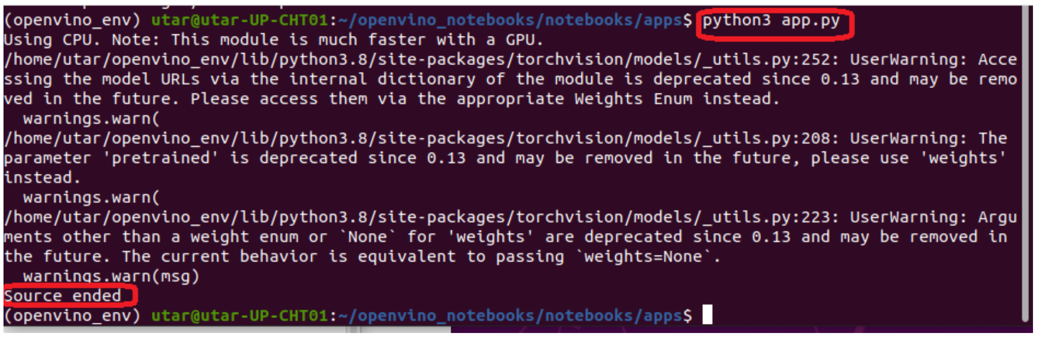
1. Activate OpenVino Environment on the Edge Device.



```
utar@utar-UP-CHT01:~$ source openvino_env/bin/activate
(openvino_env) utar@utar-UP-CHT01:~$
(openvino_env) utar@utar-UP-CHT01:~$
(openvino_env) utar@utar-UP-CHT01:~$
```

Figure 5.53 Activate OpenVino Environment

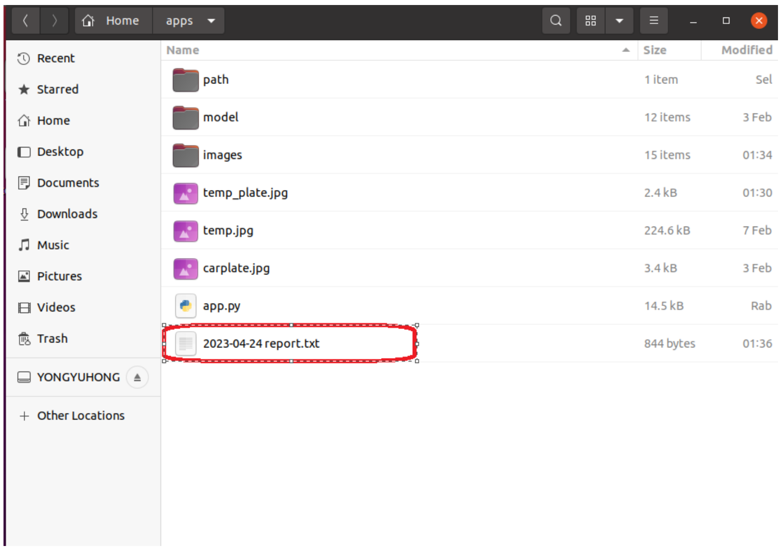
2. To run the traffic violation detection system, you need to first navigate to the project directory and execute the command "python3 app.py" in the command prompt or terminal. It is important to note that the source of the video reader in the app.py file is set to "sampleVideo.mp4" and not the camera for testing purposes.



```
(openvino_env) utar@utar-UP-CHT01:~/openvino_notebooks/notebooks/apps$ python3 app.py
Using CPU. Note: This module is much faster with a GPU.
/home/utar/openvino_env/lib/python3.8/site-packages/torchvision/models/_utils.py:252: UserWarning: Acce
ssing the model URLs via the internal dictionary of the module is deprecated since 0.13 and may be remo
ved in the future. Please access them via the appropriate Weights Enum instead.
  warnings.warn(
/home/utar/openvino_env/lib/python3.8/site-packages/torchvision/models/_utils.py:208: UserWarning: The
parameter 'pretrained' is deprecated since 0.13 and may be removed in the future, please use 'weights'
instead.
  warnings.warn(
/home/utar/openvino_env/lib/python3.8/site-packages/torchvision/models/_utils.py:223: UserWarning: Argu
ments other than a weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed in
the future. The current behavior is equivalent to passing `weights=None`.
  warnings.warn(msg)
Source ended
(openvino_env) utar@utar-UP-CHT01:~/openvino_notebooks/notebooks/apps$
```

Figure 5.54 Run "app.py" file in Terminal

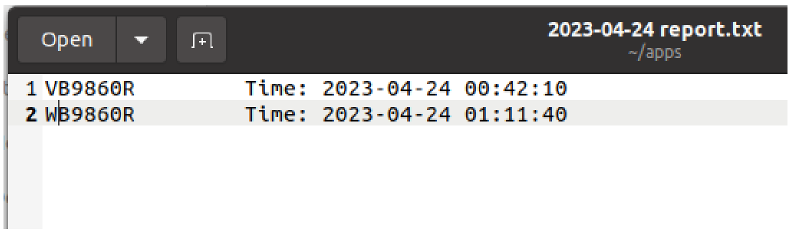
3. After running the "app.py" file using the command "python3 app.py", navigate to the directory and check for the "(TodayDate)_report file.txt". If the report file exists, it indicates that the Traffic Violation Detection System is running correctly on the edge device.



Name	Size	Modified
path	1 item	Sel
model	12 items	3 Feb
images	15 items	01:34
temp_plate.jpg	2.4 kB	01:30
temp.jpg	224.6 kB	7 Feb
carplate.jpg	3.4 kB	3 Feb
app.py	14.5 kB	Rab
2023-04-24 report.txt	844 bytes	01:36

Figure 5.55 Report Text File Generated

4. You can view the detected car plate from your sample data in the "(TodayDate)_report file.txt".

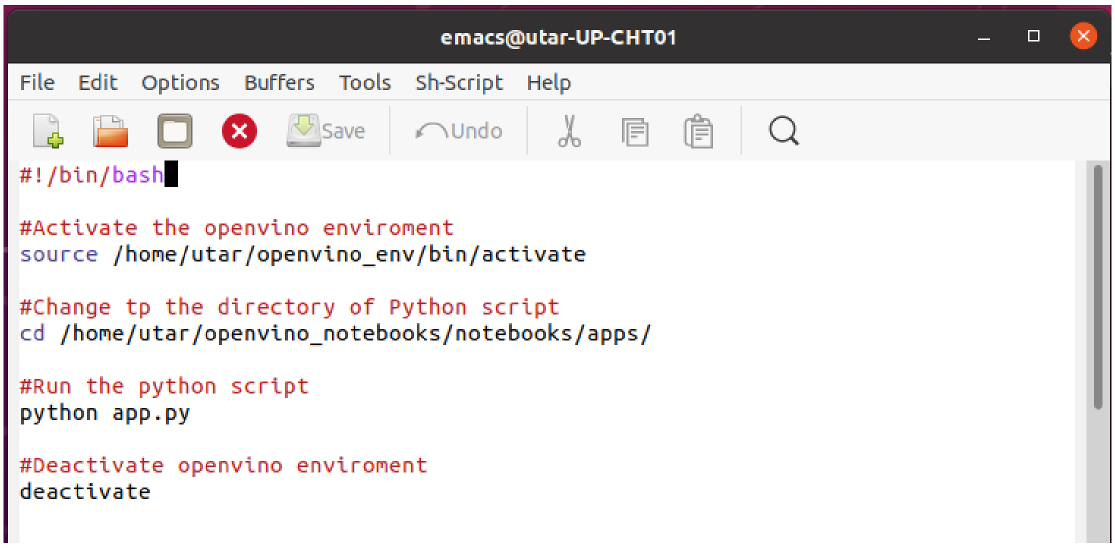


```
VB9860R          Time:  2023-04-24 00:42:10
WB9860R          Time:  2023-04-24 01:11:40
```

Figure 5.56 Content of Report.txt

5.8 Task Automation Configuration (Intel UP Board)

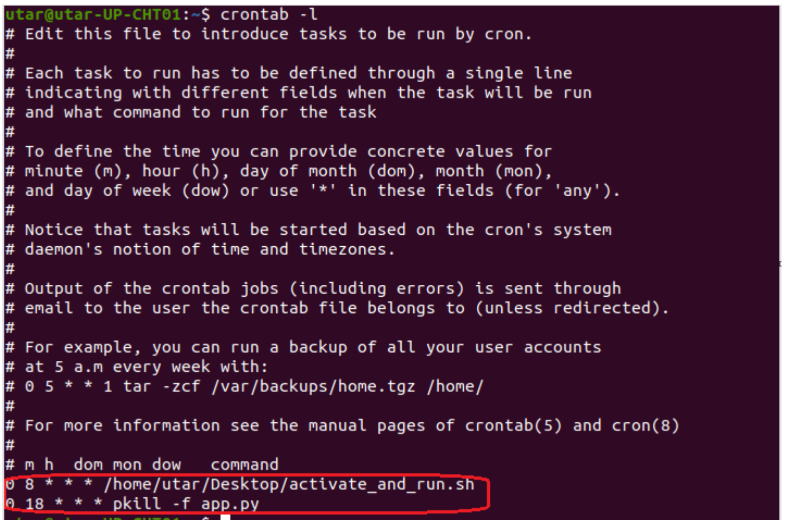
1. A bash script is created to automate the task of activating the OpenVino environment and running the traffic violation detection application in sequence.



```
#!/bin/bash

#Activate the openvino enviroment
source /home/utar/openvino_env/bin/activate

#Change tp the directory of Python script
cd /home/utar/openvino_notebooks/notebooks/apps/

#Run the python script
python app.py

#Deactivate openvino enviroment
deactivate
```

Figure 5.57 activate_and_run.sh

2. Configure Crontab to run the script automatically at every morning 8am and stop the program at evening 6pm.



```
utar@utar-UP-CHT01:~$ crontab -l
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h  dom mon dow   command
0 8 * * * /home/utar/Desktop/activate_and_run.sh
0 18 * * * pkill -f app.py
```

Figure 5.58 Crontab Setup

CHAPTER 6 System Evaluation and Discussion

6.1 System Testing and Performance Metrics

To evaluate the performance of the traffic violation detection system, the verification plan includes measuring the processed frames per second (FPS) and inference time of each frame. In addition, accuracy is used as a metric to check the performance of the system.

To measure the processed FPS, a test dataset is used to simulate the car plate detection system in a real-time scenario. The FPS is then calculated by dividing the total number of processed frames by the total time taken. The inference time of each frame is also recorded and analyzed to identify any bottlenecks or areas for optimization.

Accuracy is based on comparing the system's output to ground truth labels. The evaluation is performed on a representative dataset with few videos and scenarios, and it involves multiple rounds of testing. Each round consists of processing one video sample and comparing the system's detections to the ground truth. If the system detects the car plate correctly in the sample, it is considered one correct detection; otherwise, it is considered one incorrect detection. The accuracy is calculated as the percentage of correct detections over the total number of rounds of testing.

The verification plan includes a series of tests to validate the performance of the car plate detection system. These tests are designed to ensure that the system meets the desired accuracy and FPS requirements while maintaining efficient inference times. Any discrepancies or issues are identified and addressed through optimization techniques.

Overall, the verification plan provides a comprehensive approach to validating the performance of the car plate detection system and ensures that the system meets the required specifications for accuracy and FPS.

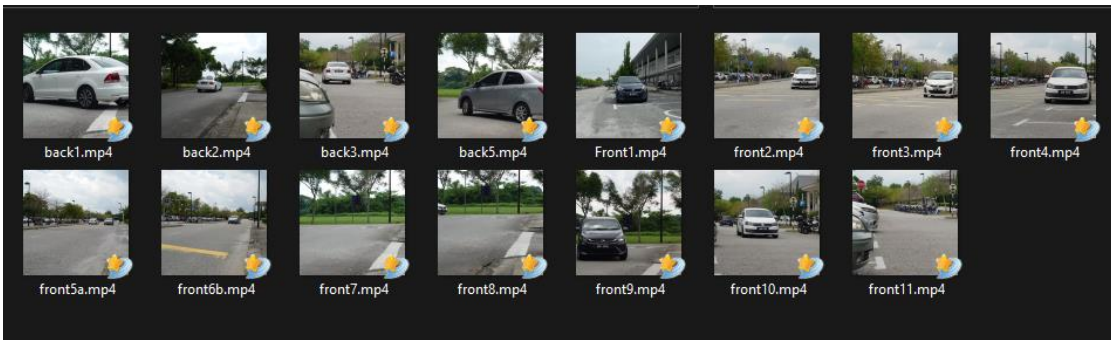
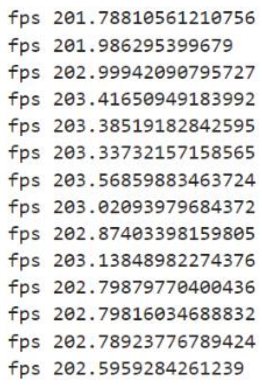


Figure 6.1 Dataset for Testing

6.2 Evaluation of Processed Frames per Second (FPS) Performance

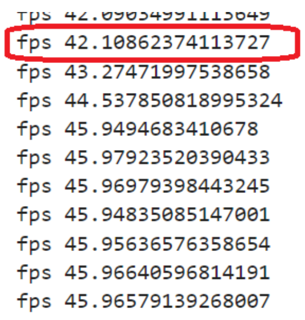
6.2.1 FPS Evaluation on Workstation

The system's average processed FPS for Vehicle and Car plate Detection Model OpenVino without performing OCR on 15 sample videos ranges from 188 to 221 FPS (on Workstation), with an average of 203 FPS. However, when OCR is active and performing its tasks, the FPS drops significantly to an average of 42 FPS at its lowest point.



```
fps 201.78810561210756
fps 201.986295399679
fps 202.99942090795727
fps 203.41650949183992
fps 203.38519182842595
fps 203.33732157158565
fps 203.56859883463724
fps 203.02093979684372
fps 202.87403398159805
fps 203.13848982274376
fps 202.79879770400436
fps 202.79816034688832
fps 202.78923776789424
fps 202.5959284261239
```

Figure 6.2 FPS on Detecting Vehicle (OpenVino)

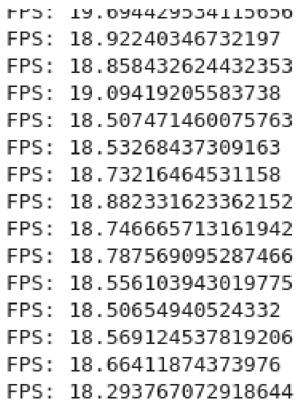


```
fps 42.09034991115649
fps 42.10862374113727
fps 43.27471997538658
fps 44.537850818995324
fps 45.9494683410678
fps 45.97923520390433
fps 45.96979398443245
fps 45.94835085147001
fps 45.95636576358654
fps 45.96640596814191
fps 45.96579139268007
```

Figure 6.3 FPS on OCR

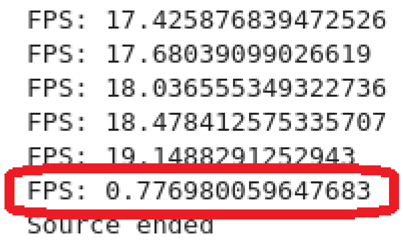
6.2.2 FPS Evaluation on Intel UP Board

The system's average processed FPS for Vehicle and Car plate Detection Model OpenVino without performing OCR on 15 sample videos ranges from 14 to 20 FPS (on Intel UP Board), with an average of 18.3 FPS. However, when OCR is active and performing its tasks, the FPS drops significantly to an average of 0.77 FPS at its lowest point.



```
FPS: 19.694429534115656
FPS: 18.92240346732197
FPS: 18.858432624432353
FPS: 19.09419205583738
FPS: 18.507471460075763
FPS: 18.53268437309163
FPS: 18.73216464531158
FPS: 18.882331623362152
FPS: 18.746665713161942
FPS: 18.787569095287466
FPS: 18.556103943019775
FPS: 18.50654940524332
FPS: 18.569124537819206
FPS: 18.66411874373976
FPS: 18.293767072918644
```

Figure 6.4 FPS on Detecting Vehicle (Intel UP Board)



```
FPS: 17.425876839472526
FPS: 17.68039099026619
FPS: 18.036555349322736
FPS: 18.478412575335707
FPS: 19.1488291252943
FPS: 0.776980059647683
Source ended
```

Figure 6.5 Fps on OCR (Intel UP Board)

6.3 Evaluation of Accuracy Performance

6.3.1 Accuracy Evaluation on Workstation

Upon executing the 15-sample dataset on the workstation, the system achieved a 100% success rate in identifying illegal driving vehicles (Vehicle and Car plate Detection Model OpenVino). However, the accuracy of the car plate number character recognition system (OCR Model) was only 81.81%.

Table 6.1 Accuracy of Models on Workstation

Model	Accuracy (Workstation)
Vehicle & Car plate Detection (OpenVino)	100%
EasyOCR	81.81%

6.3.2 Accuracy Evaluation on Intel UP Board

When the 15-sample dataset was run on the workstation, the system successfully identified illegal driving vehicles with a success rate of 93.33% using the Vehicle and Car Plate Detection model in OpenVino. However, the accuracy of the OCR model in recognizing car plate numbers was only 36.36% due to its older version and the edge device's processor not supporting AVX instruction for running the latest easy OCR. This indicates that there is a potential for improving the system's ability to accurately identify number plates, especially given the limitations posed by the use of an older OCR model and the lack of AVX instruction support on the edge device's processor.

Table 6.2 Accuracy of Models on Edge Device

Model	Accuracy (Intel Up Board)
Vehicle & Car plate Detection (OpenVino)	93.33%
EasyOCR	36.36%

6.3.3 Overview of Performance

Figures 6.6 and 6.7 provide an overview of the performance of Accuracy and FPS, respectively, for the Workstation and Edge Device (Intel UP Board).

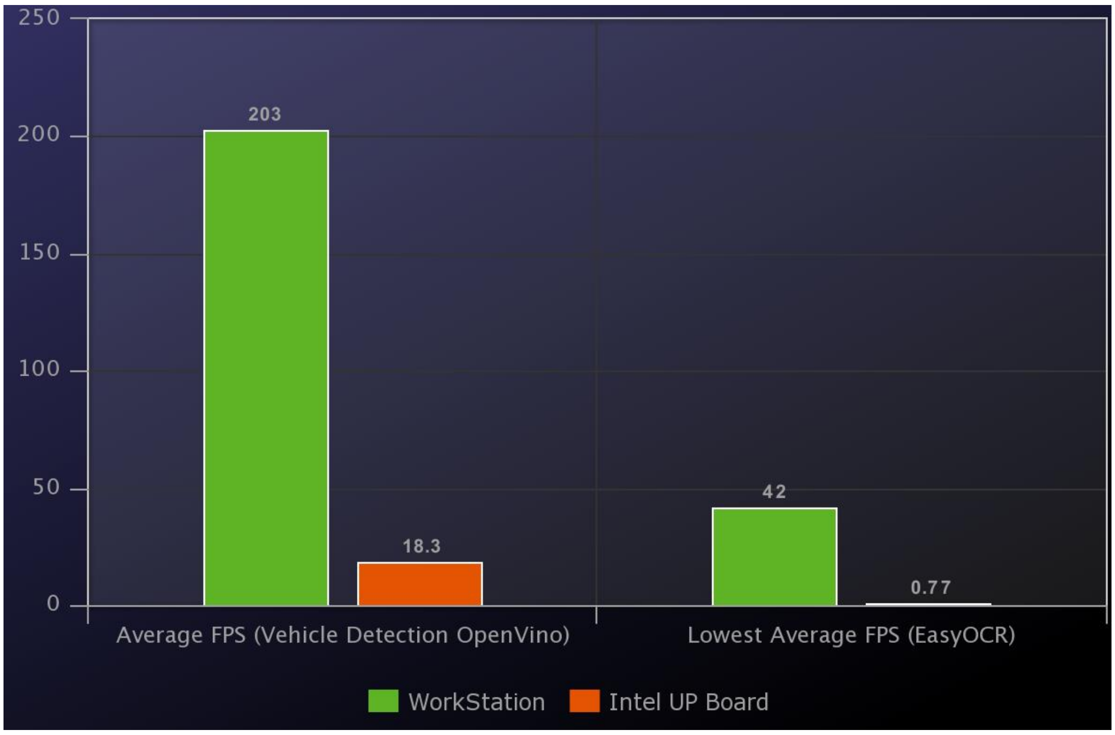


Figure 6.6 FPS Overview of FPS Performance

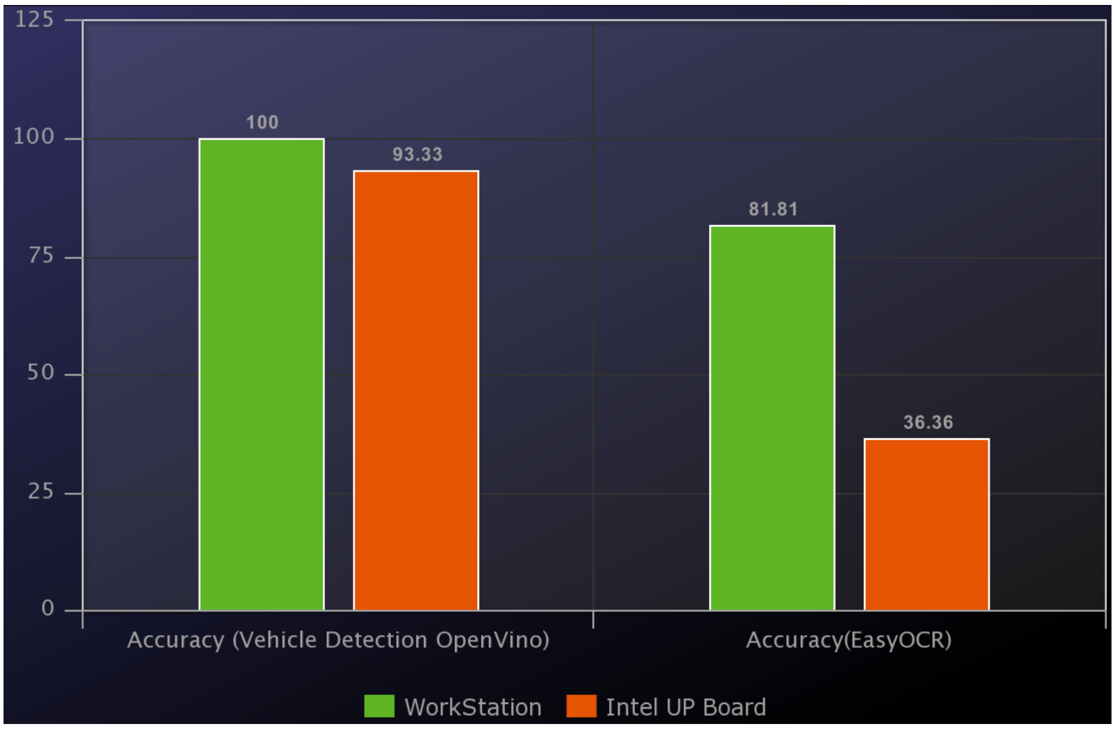


Figure 6.7 Overview of Accuracy Performance

6.3 Limitation and Future Improvements

The evaluation of the traffic violation detection system revealed several limitations that could be addressed through future improvements. Firstly, the system's accuracy in recognizing car plate numbers was limited by the use of an older OCR model and the lack of AVX instruction support on the edge device's processor. One possible solution to this issue is to upgrade to a more recent OCR model that is compatible with the processor's AVX instructions, which could lead to a significant improvement in the system's ability to accurately identify number plates.

Another limitation observed in the evaluation was the system's performance in terms of processed FPS. When OCR was active and performing its tasks, the FPS dropped significantly, which could lead to delays in processing frames in a real-time scenario. To address this limitation, one possible solution is to optimize the OCR model's performance by utilizing more efficient algorithms or by implementing hardware acceleration techniques to reduce the processing time.

Furthermore, the evaluation identified some limitations related to the system's hardware configuration. The Intel UP Board showed significantly lower performance compared to the workstation, which could be due to its lower computing power. To improve the system's overall performance, upgrading to a more powerful hardware configuration or utilizing distributed computing techniques could be explored.

However, the evaluation was performed on a representative dataset with few videos and scenarios, which may not fully capture the system's performance in more diverse or complex real-world scenarios. Expanding the evaluation to include a more diverse and extensive dataset with various scenarios, lighting conditions, and vehicle types could provide a more comprehensive understanding of the system's performance and potential limitations.

Overall, by addressing these limitations and expanding the evaluation, the traffic violation detection system could be further optimized to meet the desired accuracy and FPS requirements while maintaining efficient inference times, leading to more effective traffic monitoring and enforce.

CHAPTER 7 Conclusion and Recommendation

7.1 Conclusion

In this study, we developed a traffic violation detection system using computer vision techniques that can detect and identify various traffic violations, including illegal driving and car plate detection. The system was implemented using a deep learning approach and evaluated on both a workstation and an edge device, Intel UP Board. The results showed that the system met the required specifications for accuracy and FPS on the workstation. However, the edge device's performance was limited by the lack of AVX instruction support, lower computing power, and significant drops in FPS when OCR was active. We recommend upgrading the edge device's hardware configuration or utilizing distributed computing techniques to distribute tasks between the edge device and the workstation. Additionally, we suggest expanding the evaluation to include a more diverse and extensive dataset with various scenarios, lighting conditions, and vehicle types to provide a more comprehensive understanding of the system's performance and potential limitations.

7.2 Recommendation

To lower the edge device's resource stress and improve system performance, we suggest distributing the task to two different devices. The edge device could perform the initial task of detecting illegal driving and car plate detection and store the result in internal storage. The picture of an illegally parked vehicle can be sent to the workstation to perform heavy load of OCR, while the car plate number can be identified on the edge device. Moreover, we found that the implementation of the OpenVino toolkit provided significant advantages for the edge device in terms of performance and efficiency. The toolkit allowed for the optimization of the deep learning models for the specific hardware, resulting in faster inference times and reduced computational resources. We suggest utilizing OpenVino for developing and deploying computer vision-based systems on edge devices with limited computational resources, opening up new possibilities for real-time applications in various domains, including traffic monitoring and enforcement. These insights contribute to the advancement and optimization of traffic monitoring and enforcement systems, leading to safer and more efficient transportation systems. Further research and evaluation can provide additional insights into the system's capabilities, potential limitations, and opportunities for improvement.

REFERENCES

[1] V. Meel, “What is OpenVino? - The Ultimate Overview in 2022,” *viso.ai*, Mar. 20, 2022. https://viso.ai/computer-vision/intel-OpenVino-toolkit-overview/#:~:text=The%20OpenVino%20toolkit%20covers%20both (accessed Sep. 01, 2022).

[2] N. A. Andriyanov, “Analysis of the Acceleration of Neural Networks Inference on Intel Processors Based on OpenVino Toolkit,” *2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, Jul. 2020, doi: 10.1109/synchroinfo49631.2020.9166067.

[3] Intel AI. (2019 May) AI Driven Medical Imaging Powered by Intel and Philips[Online]. Available: https://www.intel.ai/solutions/perform-aidriven-medical-imaging-efficiently-and-cost-effectively-on-intel-cpubased-systems/

[4] G. Mathew, S. Sindhu Ramachandran, and V. S. Suchithra, “Lung Nodule Detection from low dose CT scan using Optimization on Intel Xeon and Core processors with Intel Distribution of OpenVino Toolkit,” *IEEE Xplore*, Oct. 01, 2019. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8929612 (accessed Sep. 01, 2022).

[5] S. K. Teoh, Y. H. Wong, C. F. Leong, and L. Y. Tan, “Face Detection and Face Re-identification System Using Deep Learning and OpenVino,” *2021 2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS)*, Sep. 2021, doi: 10.1109/aidas53897.2021.9574201.

[6] V. V. Zunin, “Intel OpenVino Toolkit for Computer Vision: Object Detection and Semantic Segmentation,” *IEEE Xplore*, Sep. 01, 2021. https://ieeexplore.ieee.org/document/9537452/ (accessed Sep. 01, 2022).

[7] Intel, “OpenVinotoolkit/open_model_zoo: Pre-trained Deep Learning Models,” *GitHub*. [Online]. https://github.com/OpenVinotoolkit/open_model_zoo/tree/master/models/public/ssdlite_mobilenet_v2. [Accessed: 27-Nov-2022].

[8] Meituan, “Yolov6: A single-stage object detection framework dedicated to industrial applications.,” *GitHub*. [Online]. Available: https://github.com/meituan/YOLOv6/. [Accessed: 27-Nov-2022].

[9] Meituan, “Release yolov6 2.1 · Meituan/yolov6,” *GitHub*. [Online]. Available: https://github.com/meituan/YOLOv6/releases/tag/0.2.1. [Accessed: 27-Nov-2022].

[10] Meituan, “Yolov6/deploy/ONNX at main · meituan/YOLOV6,” *GitHub*. [Online]. Available: https://github.com/meituan/YOLOv6/tree/main/deploy/ONNX. [Accessed: 27-Nov-2022].

[11] JaidedAI, “JaidedAI/EasyOCR at main” *GitHub*. [Online]. Available: https://github.com/JaidedAI/EasyOCR. [Accessed: 27-January-2023].

[12] Triq, “mtaziz/D-link Wireless N150 USB Adapter dwa-123 rev D1 installation on ubuntu 14.04 at main” *GitHub*. [Online]. Available: https://gist.github.com/mtaziz/6b1c59972623a224743a. [Accessed: 27-January-2023].

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3Y3	**Study week no.:2**
Student Name & ID: Chiew Jing Cheng 20ACB02331	
Supervisor: Ts Wong Chee Siang	
Project Title: Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino	

1. WORK DONE:

1. Revise FYP1 and create a plan.

2. Research vehicle detection model using TensorFlow.

2. WORK TO BE DONE

1. Configure a system to detect vehicles involved in illegal driving

3. PROBLEMS ENCOUNTERED

1. Difficulty in finding a pre-trained model for detecting a specific-facing vehicle in vehicle detection.

4. SELF EVALUATION OF THE PROGRESS

1. More effort is needed to find pre-trained models, especially for TensorFlow, to further enhance performance in OpenVino.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3Y3	**Study week no.:4**
Student Name & ID: Chiew Jing Cheng 20ACB02331	
Supervisor: Ts Wong Chee Siang	
Project Title: Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino	

1. WORK DONE 1. Successfully configured OpenVINO model for detecting illegal driving vehicles and car plate numbers. 2. The configuration included pre-processing and post-processing steps for accurate vehicle detection.
2. WORK TO BE DONE 1. Investigating methods for collecting car features.
3. PROBLEMS ENCOUNTERED
4. SELF EVALUATION OF THE PROGRESS 1. Good progress in my research and discovered a vehicle detection model that specifically detects front-facing vehicles.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3Y3	**Study week no.:6**
Student Name & ID: Chiew Jing Cheng 20ACB02331	
Supervisor: Ts Wong Chee Siang	
Project Title: Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino	

1. WORK DONE 1. Chose OCR as the method for car number recognition 2. Conducted pre-processing for OCR method.
2. WORK TO BE DONE 1. Post processing for OCR method
3. PROBLEMS ENCOUNTERED 1. Spent considerable time searching for an OCR model suitable for accurately recognizing car plate numbers. Many of the models tested did not perform well enough on the detected car plates.
4. SELF EVALUATION OF THE PROGRESS 1. The progress is still on track and testing a good OCR model only took a short amount of time.

[illegible] [illegible]

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3Y3	**Study week no.:8**
Student Name & ID: Chiew Jing Cheng 20ACB02331	
Supervisor: Ts Wong Chee Siang	
Project Title: Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino	

1. WORK DONE 1. The traffic violation detection system has been fully configured and is now able to generate a report of the day's detected illegal drivers.
2. WORK TO BE DONE 1. Configuring Intel UP Board
3. PROBLEMS ENCOUNTERED 1. Choosing OS for Intel UP Board
4. SELF EVALUATION OF THE PROGRESS 1. Good, the traffic violation detection system has been successfully configured and is meeting the desired performance standards.

[illegible] [illegible]

Supervisor's signature Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3Y3	**Study week no.:10**
Student Name & ID: Chiew Jing Cheng 20ACB02331	
Supervisor: Ts Wong Chee Siang	
Project Title: Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino	

1. WORK DONE 1. configure OS on the edge device 2. configure drivers on the edge device 3. Set up OpenVino Environment on the edge device
2. WORK TO BE DONE 1. Moving whole project (traffic violation detection) to edge device and making it deployable.
3. PROBLEMS ENCOUNTERED 1. the OCR model cannot be used on the edge device as its CPU is too old to support AVX instruction.
4. SELF EVALUATION OF THE PROGRESS 1. Overall Good, first time configuring for edge device, learn many from it.

[illegible]

Supervisor's signature

[illegible]

Student's signature

FINAL YEAR PROJECT WEEKLY REPORT

(Project II)

Trimester, Year: T3Y3	**Study week no.:12**
Student Name & ID: Chiew Jing Cheng 20ACB02331	
Supervisor: Ts Wong Chee Siang	
Project Title: Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino	

1. WORK DONE 1.edge device is successfully configured 2. Configured VNC Viewer 3. Configured bash script for task automation
2. WORK TO BE DONE 1. Report writing
3. PROBLEMS ENCOUNTERED 1. Edge device performance is not good enough
4. SELF EVALUATION OF THE PROGRESS 1. try to optimizing output by reducing unnecessary frames and functions to improve CPU resource utilization.

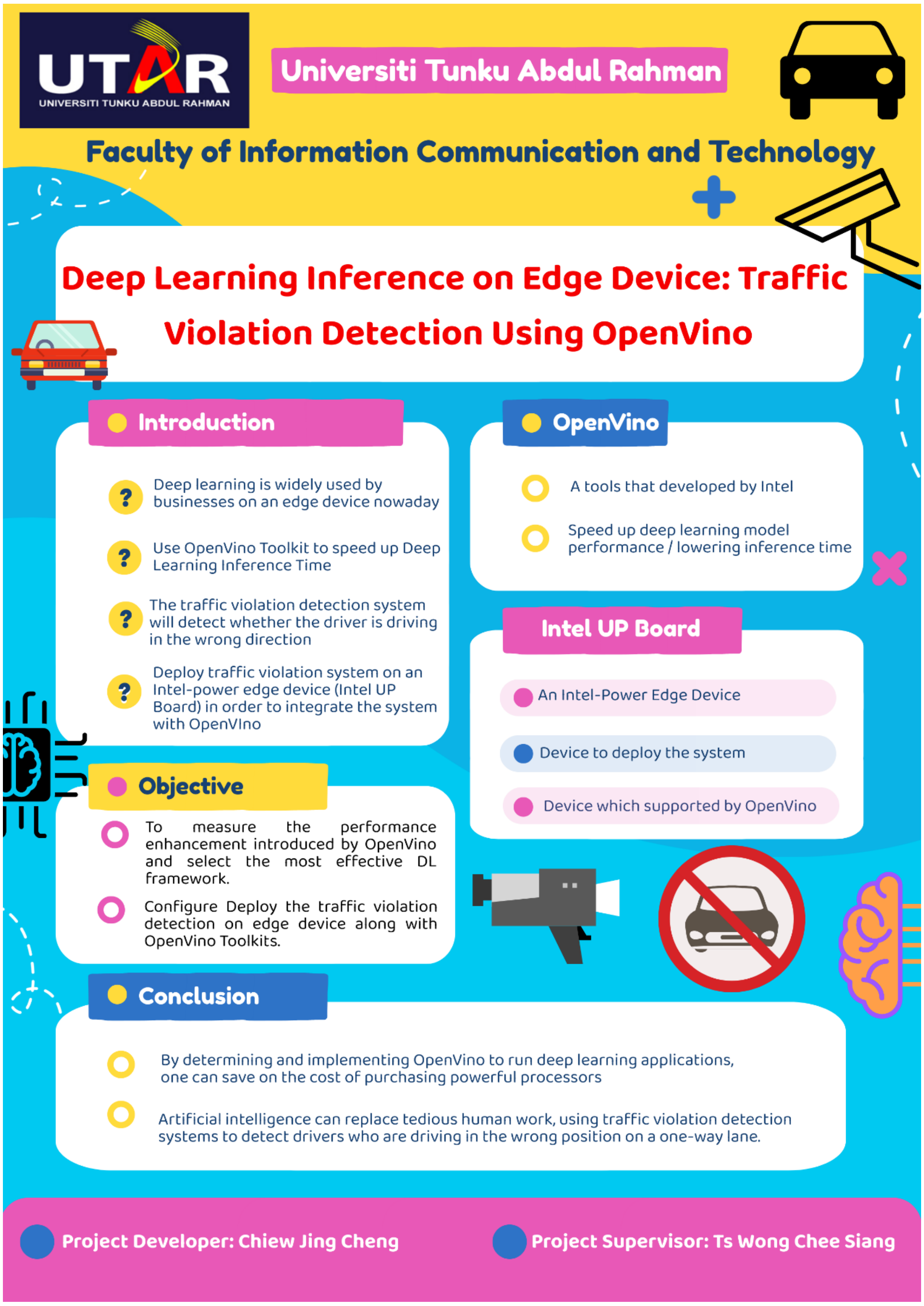
[illegible]

Supervisor's signature

[illegible]

Student's signature

POSTER



UTAR
UNIVERSITI TUNKU ABDUL RAHMAN

Universiti Tunku Abdul Rahman

Faculty of Information Communication and Technology

Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino

Introduction

- Deep learning is widely used by businesses on an edge device nowaday
- Use OpenVino Toolkit to speed up Deep Learning Inference Time
- The traffic violation detection system will detect whether the driver is driving in the wrong direction
- Deploy traffic violation system on an Intel-power edge device (Intel UP Board) in order to integrate the system with OpenVIno

OpenVino

- A tools that developed by Intel
- Speed up deep learning model performance / lowering inference time

Intel UP Board

- An Intel-Power Edge Device
- Device to deploy the system
- Device which supported by OpenVino

Objective

- To measure the performance enhancement introduced by OpenVino and select the most effective DL framework.
- Configure Deploy the traffic violation detection on edge device along with OpenVino Toolkits.

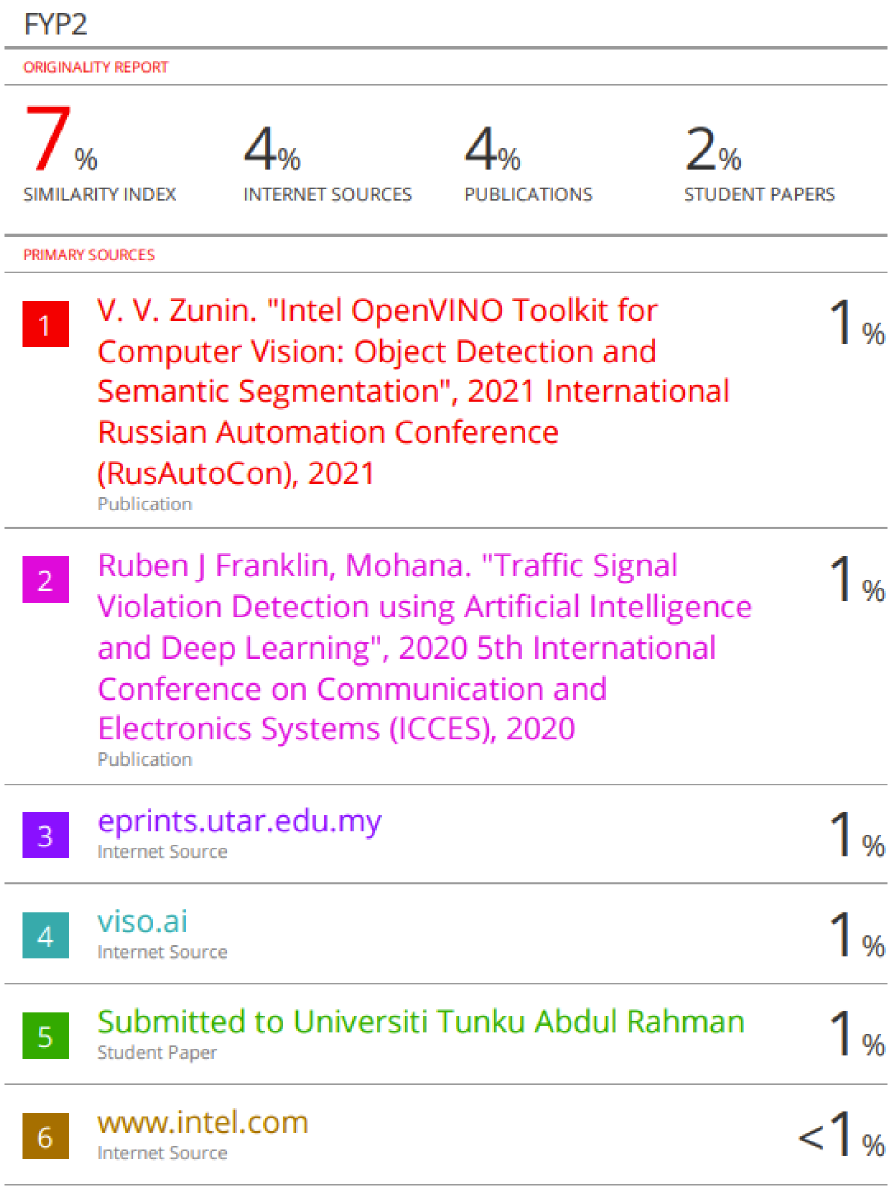
Conclusion

- By determining and implementing OpenVino to run deep learning applications, one can save on the cost of purchasing powerful processors
- Artificial intelligence can replace tedious human work, using traffic violation detection systems to detect drivers who are driving in the wrong position on a one-way lane.

Project Developer: Chiew Jing Cheng

Project Supervisor: Ts Wong Chee Siang

PLAGIARISM CHECK RESULT



FYP2

ORIGINALITY REPORT

7%	4%	4%	2%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	V. V. Zunin. "Intel OpenVINO Toolkit for Computer Vision: Object Detection and Semantic Segmentation", 2021 International Russian Automation Conference (RusAutoCon), 2021 Publication	1%
2	Ruben J Franklin, Mohana. "Traffic Signal Violation Detection using Artificial Intelligence and Deep Learning", 2020 5th International Conference on Communication and Electronics Systems (ICCES), 2020 Publication	1%
3	eprints.utar.edu.my Internet Source	1%
4	viso.ai Internet Source	1%
5	Submitted to Universiti Tunku Abdul Rahman Student Paper	1%
6	www.intel.com Internet Source	<1%

7	N. A. Andriyanov. "Analysis of the Acceleration of Neural Networks Inference on Intel Processors Based on OpenVINO Toolkit", 2020 Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO), 2020 Publication	<1 %
8	Shen Khang Teoh, Yiek Heng Wong, Chun Farn Leong, Lyk Yin Tan. "Face Detection and Face Re-identification System Using Deep Learning and OpenVINO", 2021 2nd International Conference on Artificial Intelligence and Data Sciences (AiDAS), 2021 Publication	<1 %
9	Gina Mathew, S. Sindhu Ramachandran, V.S. Suchithra. "Lung Nodule Detection from low dose CT scan using Optimization on Intel Xeon and Core processors with Intel Distribution of OpenVINO Toolkit", TENCON 2019 - 2019 IEEE Region 10 Conference (TENCON), 2019 Publication	<1 %
10	ieeexplore.ieee.org Internet Source	<1 %
11	publications.hse.ru Internet Source	<1 %
12	www.ncbi.nlm.nih.gov Internet Source	

	Publication	
23	"Proceedings of the International Conference on Data Engineering and Communication Technology", Springer Science and Business Media LLC, 2017 Publication	<1%
24	Fatima Zahra Guerrouj, Mohamed ABOUZAHIR, Mustapha RAMZI, El Mehdi ABDALI. "Analysis of the acceleration of deep learning inference models on a heterogeneous architecture based on OpenVINO", 2021 4th International Symposium on Advanced Electrical and Communication Technologies (ISAECT), 2021 Publication	<1%
25	Mau-Luen Tham, Wei Kun Tan. "IoT Based License Plate Recognition System Using Deep Learning and OpenVINO", 2021 4th International Conference on Sensors, Signal and Image Processing, 2021 Publication	<1%
26	Riya Banerjee, Saswati Chakladar, Ashok Mohanty, Shyamal Kumar Chattopadhyay, Sanchita Chakravarty. "Leaching characteristics of rare earth elements from coal ash using organosulphonic acids", Minerals Engineering, 2022 Publication	<1%

27	Xiaoxuan Wang, Feiyu Zhao, Ping Lin, Yongming Chen. "Evaluating computing performance of deep neural network models with different backbones on IoT-based edge and cloud platforms", Internet of Things, 2022 Publication	<1 %
28	Hanan Al-Hadeethi, Shahab Abdulla, Mohammed Diykh, Ravinesh C. Deo, Jonathan H. Green. "An Eigenvalues-Based Covariance Matrix Bootstrap Model Integrated With Support Vector Machines for Multichannel EEG Signals Analysis", Frontiers in Neuroinformatics Internet Source	<1 %
29	Wolfgang Rankl, Wolfgang Effing. "Smart Card Handbook", Wiley, 2010 Publication	<1 %
30	www.dell.com Internet Source	<1 %

Exclude quotes	On	Exclude matches	< 8 words
Exclude bibliography	On		

		<1 %
13	Submitted to Mississippi State Board for Community & Junior Colleges Student Paper	<1 %
14	fict.utar.edu.my Internet Source	<1 %
15	www.mdpi.com Internet Source	<1 %
16	etd.aau.edu.et Internet Source	<1 %
17	Submitted to Middlesex University Student Paper	<1 %
18	Submitted to Technological Institute of the Philippines Student Paper	<1 %
19	azpdf.org Internet Source	<1 %
20	www.researchgate.net Internet Source	<1 %
21	www2.utar.edu.my Internet Source	<1 %
22	"Algorithms and Architectures for Parallel Processing", Springer Science and Business Media LLC, 2014	<1 %

PLAGIARISM CHECK RESULT

Universiti Tunku Abdul Rahman			
Form Title : Supervisor's Comments on Originality Report Generated by Turnitin for Submission of Final Year Project Report (for Undergraduate Programmes)			
Form Number: FM-IAD-005	Rev No.: 0	Effective Date: 01/10/2013	Page No.: 1of 1

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Full Name(s) of Candidate(s)	Chiew Jing Cheng
ID Number(s)	010219-14-1449
Programme / Course	CS
Title of Final Year Project	Deep Learning Inference on Edge Device: Traffic Violation Detection Using OpenVino

Similarity	**Supervisor's Comments (Compulsory if parameters of originality exceed the limits approved by UTAR)**
Overall similarity index: 7 % **Similarity by source** Internet Sources: 4 % Publications: 4 % Student Papers: 2 %	
Number of individual sources listed of more than 3% similarity: 0	
Parameters of originality required, and limits approved by UTAR are as Follows: (i) **Overall similarity index is 20% and below, and** (ii) **Matching of individual sources listed must be less than 3% each, and** (iii) **Matching texts in continuous block must not exceed 8 words** *Note: Parameters (i) – (ii) shall exclude quotes, bibliography and text matches which are less than 8 words.*	

Note: Supervisor/Candidate(s) is/are required to provide softcopy of full set of the originality report to Faculty/Institute

Based on the above results, I hereby declare that I am satisfied with the originality of the Final Year Project Report submitted by my student(s) as named above.

[illegible]

Signature of Supervisor

Name: Ts. Wong Chee Siang

Date: 25 APRIL 2023

Signature of Co-Supervisor

Name:

Date:

UNIVERSITI TUNKU ABDUL RAHMAN

FACULTY OF INFORMATION & COMMUNICATION TECHNOLOGY (KAMPAR CAMPUS)

CHECKLIST FOR FYP2 THESIS SUBMISSION

Student Id	20ACB02331
Student Name	Chiew Jing Cheng
Supervisor Name	TS Wong Chee Siang

TICK (√)	**DOCUMENT ITEMS** Your report must include all the items below. Put a tick on the left column after you have checked your report with respect to the corresponding item.
	Front Cover
√	Title Page
√	Signed Report Status Declaration Form
√	Signed FYP Thesis Submission Form
√	Signed form of the Declaration of Originality
√	Acknowledgement
√	Abstract
√	Table of Contents
√	List of Figures (if applicable)
√	List of Tables (if applicable)
√	List of Symbols (if applicable)
√	List of Abbreviations (if applicable)
√	Chapters / Content
√	Bibliography (or References)
√	All references in bibliography are cited in the thesis, especially in the chapter of literature review
√	Appendices (if applicable)
√	Weekly Log
√	Poster
√	Signed Turnitin Report (Plagiarism Check Result - Form Number: FM-IAD-005)
√	I agree 5 marks will be deducted due to incorrect format, declare wrongly the ticked of these items, and/or any dispute happening for these items in this report.

*Include this form (checklist) in the thesis (Bind together as the last page)

I, the author, have checked and confirmed all the items listed in the table are included in my report.

[illegible]
____
(Signature of Student)
Date:23 April 2023