

**STUDY OF VISION BASED TRAFFIC
CONGESTION CLASSIFICATION
MONITORING SYSTEM (VBTCCMS)**

SHAMRAO A/L RAMASAMY

**BACHELOR OF ELECTRICAL ENGINEERING
(ELECTRONICS) WITH HONOURS**

UNIVERSITI MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

DECLARATION OF THESIS AND COPYRIGHT

Author's Full Name : SHAMRAO A/L RAMASAMY

Date of Birth : 9 FEBRUARY 1997

Title : STUDY OF VISION BASED TRAFFIC CONGESTION

CLASSIFICATION MONITORING SYSTEM (VBTCMS)

Academic Session : _____

I declare that this thesis is classified as:

- CONFIDENTIAL (Contains confidential information under the Official Secret Act 1997)*
- RESTRICTED (Contains restricted information as specified by the organization where research was done)*
- OPEN ACCESS I agree that my thesis to be published as online open access (Full Text)

I acknowledge that Universiti Malaysia Pahang reserves the following rights:

1. The Thesis is the Property of Universiti Malaysia Pahang
2. The Library of Universiti Malaysia Pahang has the right to make copies of the thesis for the purpose of research only.
3. The Library has the right to make copies of the thesis for academic exchange.

Certified by:



(Student's Signature)



(Supervisor's Signature)

970209-14-5567
New IC/Passport Number
Date: 10 FEBRUARY 2022

DR. Ahmad Afif Bin Mohd Faudzi
Name of Supervisor
Date: 10 FEBRUARY 2022

NOTE: * If the thesis is CONFIDENTIAL or RESTRICTED, please attach a thesis declaration letter.

THESIS DECLARATION LETTER (OPTIONAL)

Librarian,
Perpustakaan Universiti Malaysia Pahang,
Universiti Malaysia Pahang,
Lebuhraya Tun Razak,
26300, Gambang, Kuantan.

Dear Sir,

CLASSIFICATION OF THESIS AS RESTRICTED

Please be informed that the following thesis is classified as RESTRICTED for a period of three (3) years from the date of this letter. The reasons for this classification are as listed below.

Author's Name	SHAMRAO A/L RAMASAMY
Thesis Title	STUDY OF VISION BASED TRAFFIC CONGESTION CLASSIFICATION MONITORING SYSTEM (VBTCCMS)

Reasons	(i)
	(ii)
	(iii)

Thank you.

Yours faithfully,



(Supervisor's Signature)

Date: 10 FEBRUARY 2022

Stamp: DR. AHMAD AFIF BIN MOHD FAUDZI
SENIOR LECTURER
DEPARTMENT OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITI MALAYSIA PAHANG
LEBUHRAYA TUN RAZAK
26300 GAMBANG, KUANTAN, PAHANG
TEL: +609-424 6154

Note: This letter should be written by the supervisor, addressed to the Librarian, *Perpustakaan Universiti Malaysia Pahang* with its copy attached to the thesis.



SUPERVISOR'S DECLARATION

I hereby declare that I have checked this thesis and, in my opinion, this thesis is adequate in terms of scope and quality for the award of the degree of Electrical Engineering (Electronics) with Honours.

(Supervisor's Signature)

Full Name : DR. AHMAD AFIF BIN MOHD FAUDZI
SENIOR LECTURER
DEPARTMENT OF ELECTRICAL ENGINEERING
COLLEGE OF ENGINEERING
UNIVERSITI MALAYSIA PAHANG
Position : LEBUHRAYA TUN RAZAK
26300 GAMBANG, KUANTAN, PAHANG
TEL: +609-424 6154
Date : 10 FEBRUARY 2022

(Co-supervisor's Signature)

Full Name :
Position :
Date :



STUDENT'S DECLARATION

I hereby declare that the work in this thesis is based on my original work except for quotations and citations which have been duly acknowledged. I also declare that it has not been previously or concurrently submitted for any other degree at Universiti Malaysia Pahang or any other institutions.

A handwritten signature in black ink, appearing to read 'Shamrao', is written over a horizontal line that spans the width of the signature area.

(Student's Signature)

Full Name : SHAMRAO A/L RAMASAMY

ID Number : EA18033

Date : 10 FEBRUARY 2022

STUDY OF VISION BASED TRAFFIC CONGESTION CLASSIFICATION
MONITORING SYSTEM (VBTCCMS)

SHAMRAO A/L RAMASAMY

Thesis submitted in fulfillment of the requirements
for the award of the
Bachelor of Electrical Engineering (Electronics) with Honours

College of Engineering
UNIVERSITI MALAYSIA PAHANG

FEBRUARY 2022

ACKNOWLEDGEMENTS

First and foremost, I would like to express my appreciation to God for allowing me to finally finish my Final Year Project Thesis during this time. I would be nowhere today if it weren't for his blessings. Secondly, I would like to express my thanks to Dr. Ahmad Afif bin Mohd Faudzi, my project supervisor, for his direction and critical remarks while I worked on my project thesis. He has been really inspiring, and he has piqued my interest in finishing this project thesis.

I would like to express my gratitude to my parents, Mr. Ramasamy A/L Muthaloo and Mrs. Parvathy A/P Ramaloo, as well as my family members, for their unwavering love and support during my studies and project progress.

Last but not least, I would like to express my gratitude to Dhilen A/L Manimaran, my best friend, for his unwavering support, collaboration, and ideas. One of the keys to my achievement was the assistance and inspiration. Thank you all so much.

ABSTRAK

Pengelasan imej ialah tugas untuk mengenali item atau subjek mengikut kelas yang telah diperuntukkan dalam dunia komputer. Begitu juga, tesis ini membincangkan kerja-kerja yang telah dilakukan dan bagaimana kesesakan lalu lintas diklasifikasikan menggunakan video CCTV tepi jalan. Matlamatnya adalah untuk menggunakan seni bina untuk menyiasat dan mengklasifikasikan pembolehubah kesesakan lalu lintas kepada tiga kategori: kesesakan rendah, kesesakan sederhana dan kesesakan yang berlebihan. Reka bentuk memerlukan kajian seni bina serta aplikasi untuk mengesan setiap kelas. Kajian faktor kesesakan menggunakan YOLO dan Deep Sort, yang dibina menggunakan platform TensorFlow dan Keras, akan dibincangkan dalam tesis ini. Tujuan utama projek ini adalah untuk membangunkan sistem untuk mengklasifikasikan kesesakan lalu lintas di laluan yang sibuk, dengan sistem itu dapat membahagikan kesesakan lalu lintas kepada tiga kategori: rendah, sederhana dan tinggi. Keseluruhan proses pengkategorian dijalankan oleh sistem menggunakan penglihatan. TensorFlow ialah rangka kerja pengaturcaraan sumber terbuka yang menyediakan pelbagai seni bina serta antara muka yang mudah digunakan untuk aplikasi masa hadapan.

ABSTRACT

Image classification is the task of recognising an item or subject according to the class to which it has been allocated in the computer world. Similarly, this thesis discusses the work that was done and how traffic congestion was classified using roadside CCTV video. The goal is to use an architecture to investigate and classify traffic congestion variables into three categories: low congestion, medium congestion, and excessive congestion. The design entails a study of architecture as well as an application for detecting each class. The study of congestion factor utilising YOLO and Deep Sort, which was constructed using TensorFlow and Keras platform, will be covered in this thesis. The major purpose of this project is to develop a system for classifying traffic congestion on a busy route, with the system being able to classify traffic congestion into three categories: low, medium, and high. The entire categorization process is carried out by the system using vision. TensorFlow is an open source programming framework that provides a variety of architectures as well as an easy-to-use interface for future applications.

TABLE OF CONTENT

DECLARATION	
TITLE PAGE	
ACKNOWLEDGEMENTS	ii
ABSTRAK	iii
ABSTRACT	iv
TABLE OF CONTENT	v
LIST OF TABLES	viii
LIST OF FIGURES	ix
LIST OF ABBREVIATIONS	x
CHAPTER 1 INTRODUCTION	1
1.1 Background overview	1
1.2 Problem statement	2
1.3 Objective	2
1.4 Scope	3
CHAPTER 2 LITERATURE REVIEW	4
2.1 Introduction	4
2.2 TensorFlow	4
2.2.1 Tensors	5
2.2.2 Variables	5
2.2.3 Lite	6
2.2.4 Session	6

2.2.5	Evaluation of Accuracy	7
2.3	Keras Documentation	7
2.4	Python	9
2.5	Anaconda	9
2.6	Spyder	10
2.7	Article/Journal Reviews	10
2.7.1	London : Congestion Charge	10
2.7.2	Baton Rouge: Continuous Intersection Flow	12
2.7.3	Traffic Congestion in China	13
2.7.4	Traffic Congestion of Urban Main Road under Accident Conditions	14
2.7.5	Traffic Congestion and Remedial Measures at Traffic Mor in Pabna City, Bangladesh	16
CHAPTER 3 METHODOLOGY		17
3.1	Introduction	17
3.2	Project architecture	17
3.2.1	YOLO (You Only Look Once) Model architecture.	17
3.2.2	DeepSort architecture	19
3.3	Visual Masking	20
3.3.1	Target-to-mask spatial separation	20
3.4	Project Block diagram	22
3.5	Project Flowchart diagram	23
CHAPTER 4 RESULTS AND DISCUSSION		26
4.1	Introduction	26
4.2	Dataset	26

4.3	Anaconda prompt	27
4.4	Results and Analysis	28
4.4.1	Vehicle Detection	29
4.4.2	Vehicle Identification and Counting	29
4.4.3	Average Speed Counting	30
4.4.4	Classifying Congestion Rate	31
CHAPTER 5 CONCLUSION		33
5.1	Introduction	33
5.2	Future Recommendation	33
REFERENCES		35
APPENDIX		37

LIST OF TABLES

Table 2.1 Classification table of average speed (km/h) of main road during peak hours.	15
--	----

LIST OF FIGURES

Figure 2.1 Initializing Variables for 'yolov4.weights'.	5
Figure 2.2 tf.lite Interpreter.	6
Figure 2.3 Session Initialization in TensorFlow.	7
Figure 2.4 Initializing Training Accuracy.	7
Figure 2.5 Image Augmentation Arrangement.	8
Figure 2.6 Congestion Charging Zones' Signs in London.	11
Figure 2.7 Map of Congestion Zone in London.	11
Figure 2.8 Continuous Flow Intersection (Lee, 2013).	12
Figure 2.9 Average congestion delay index in major cities from 2015 to 2017 (Amap, 2018).	14
Figure 2.10 Pabna city map.	16
Figure 3.1 Image Separation in Yolo Architecture.	18
Figure 3.2 Yolo Architecture.	18
Figure 3.3 Deep Sort Architecture	19
Figure 3.4 Initializing Masking Technique in 'OpenCV'.	20
Figure 3.5 Masking Technique used in this system.	21
Figure 3.6 Project Block diagram.	22
Figure 3.7 Project Flowchart diagram.	24
Figure 4.1 Classes in 'coco.names'.	27
Figure 4.2 Initializing Allowed Classes.	27
Figure 4.3 Anaconda Prompt window.	28
Figure 4.4 Vehicle detection result.	29
Figure 4.5 Altering the colour and the thickness of the square.	29
Figure 4.6 Vehicle Identification and Counting result.	29
Figure 4.7 Average Speed Counting result.	30
Figure 4.8 Formula for Average Speed.	30
Figure 4.9 Result classifying Low Congestion Rate.	31
Figure 4.10 Result classifying Medium Congestion Rate.	31

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
API	Application Program Interface
CCTV	Closed Circuit Television
CFI	Continuous Flow Intersection
CLI	Command Line Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
GPU	Graphic Processing Unit
IDE	Integrated Development Environment
ITS	Intelligent Transportation System
MOT	Multiple Object Tracking
RGB	Red Blue Green
tf	TensorFlow
YOLO	You Only Look Once

CHAPTER 1

INTRODUCTION

1.1 Background overview

Slower speeds, longer journey durations, and higher vehicular queueing are all symptoms of traffic congestion[1] in transportation. Traffic congestion on Malaysia's metropolitan road networks has significantly risen. Congestion occurs when traffic demand is high enough that the interaction between cars reduces the pace of the traffic stream. Engineers previously devised traffic lights[2] to alleviate traffic congestion in metropolitan areas. Many road users have found the traffic light to be quite helpful, particularly in minimising or preventing traffic accidents. In addition, it improves traffic flow and saves time for many individuals. However, as the population grows and car ownership rises, so does the need for road infrastructure, which frequently results in traffic congestion, travel delays, and wasteful vehicle discharge. Two approaches are likely to be used to address this problem. The first method is to expand capacity by providing new alternative routes, however this method can be expensive, time-consuming, and reduce capacity in the short term. The second strategy is to improve the efficiency of existing infrastructure and technologies, such as traffic congestion monitoring systems. We recommend the second option, which involves developing a new traffic congestion monitoring system using current advances in the field of Artificial Intelligence[3]. The following is how we want to solve the traffic congestion problem. What is the congestion rate of the traffic lane, what sort of vehicle is approaching the traffic lane, and how many cars are approaching the traffic lane, given the current condition of traffic on a particular road? We would have seen a lot more learning, but these systems don't use the data to its full potential. The major goal of this project is to develop a visual-based traffic congestion categorization system that uses the proper architecture to output the traffic lane's congestion rate and the number of cars approaching it depending on its classifications.

1.2 Problem statement

According to the literature[3], traffic congestion causes individuals, businesses, and the economy as a whole to incur additional costs and time, as well as stress. Individuals and society pay a price for the additional gasoline used and the higher-than-necessary level of car emissions.

If people need to adjust their routines to prevent delays or make concessions in case of unforeseen delays, their lives are disrupted. People are considerably more affected by unexpected delays if they are late for, or even miss, an event or appointment.

In addition, one of the major causes of traffic congestion is the expanding population and increasing vehicles ownership. As a result, the need for road infrastructure has increased, resulting in traffic congestion, travel delays, and wasteful vehicle discharge.

Because of the monetary and quality-of-life implications of traffic congestion, those in charge of designing and monitoring the highway network make every effort to reduce substantial congestion wherever feasible. If nothing is done to alleviate traffic congestion, circumstances will worsen, and travel times will become longer and less reliable. This has an impact on public transportation as well.

Last but not least, the existing traffic congestion control technology[4] (sensor-based) is costly and requires routine maintenance to ensure that it functions properly. The sensor-based system also has some issues recognising traffic congestion during wet seasons, and because it is sensor-based, it is unable to count or identify vehicle types.

1.3 Objective

The major goal of this research is to develop a visual-based traffic congestion classifier utilising CCTV camera installed along highways. The system will subsequently be able to determine the sort of traffic circumstances, such as low, medium, and high congestion, by counting the number of cars approaching the traffic lane depending on its classifications. As a result, the following goals breakdowns have been created to achieve this goal:

- i. Identify vehicles queuing up at the traffic lane and do counting of vehicles approaching the traffic lane and record the data.
- ii. Identify a suitable method for detection of congestion at a selected traffic light junction via video frames.
- iii. Identify methods for congestion classification.
- iv. Perform comparison for congestion classification.

1.4 Scope

The major goal of this system is to classify traffic congestion in Cyber Jaya, Selangor. This technology is capable of detecting traffic congestion (Low Congestion, Medium Congestion, High Congestion). The classification will be based on continuous image capture (video) from roadside CCTV cameras.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter covers the underlying notions relating to the software and functions that were utilized to construct this project. In addition, this chapter goes into further specifics regarding the project and research that has been done on a visual based intelligent traffic monitoring system and a traffic congestion system study by other researchers. This chapter will go through the previous approaches for analysing datasets with Convolutional Neural Networks (CNN) and the constraints of the prior work.

2.2 TensorFlow

TensorFlow[5] is an open source programming toolkit that uses information stream diagrams to do numerical computations. TensorFlow[5] was designed by Google Brain Team specialists and designers for the purposes of doing machine learning and deep neural network research within Google's Machine Intelligence research organisation. However, the framework is broad enough to be used in a variety of situations. TensorFlow's open source nature allows deep learning enthusiasts to use it for exploratory testing as well as evaluating novel architectures and models. TensorFlow[5] is compatible with the majority of operating systems, including Windows, Linux, and iOS Mac. For quicker and more effective training, it's ideal to use an NVIDIA GPU. Throughout the project, a Linux operating system with a CPU was used. TensorFlow's[5] runtime environment is supported by the Python programming language.

2.2.1 Tensors

A tensor is a categorization of matrices and vectors to a theoretically high dimension. Tensors are represented in TensorFlow as n-dimensional basic datatypes of arrays. The key programme that we alter throughout the creation of a TensorFlow[5] programme is `tf.Tensor`. The primary purpose of the `tf.Tensor` is to generate a value from an object that represents a partly specified calculation. TensorFlow[5] projects operate by creating a chart of `tf.Tensor` objects, stating how each tensor is represented in relation to the other tensors available, and then executing sections of this chart to get the desired outcomes. Tensors can be represented as vectors or scalars, with the dimensions of a tensor determining its rank. For the training purposes of this project, a 225-by-225 picture with RGB is utilised in the image dataset.

2.2.2 Variables

Variables are used in TensorFlow to express the weights inside a model. The model's weights are trained over numerous iterations in order to achieve the best outcome. As a result, the initial value assigned to them may alter rapidly throughout the training. The graphic below shows an example of how the `yolov4.weights`[6] initialises a variable, which was the major architecture employed in this project.

```
# Initialization operation from scratch for the new "fc8" layers
# 'get_variables' will only return the variables whose name starts with given pattern
fc8_variables = tf.contrib.framework.get_variables('yolov4.weights')
fc8-init = tf.variables_initializer(fc8_variables)
```

Figure 2.1 Initializing Variables for 'yolov4.weights'.

The preceding example simply implies that the programmer may provide the starting values, CPU devices, and the variable to be utilised. This improves the program's efficiency and usability.

2.2.3 Lite

TensorFlow Lite[5] is a suite of technologies that allows developers to execute their trained models on mobile, embedded, and IoT devices and desktops, enabling on-device machine learning. It works with embedded Linux, Android, iOS, and MCUs, among other platforms. As a result, the input details assigned to them may alter rapidly throughout the training. The graphic below shows an example of how the tf.lite interprets the input details , which was the major architecture employed in this project.

```
# load tflite model if flag is set
if FLAGS.framework == 'tflite':
    interpreter = tf.lite.Interpreter(model_path=FLAGS.weights)
    interpreter.allocate_tensors()
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    print(input_details)
    print(output_details)
# otherwise load standard tensorflow saved model
```

Figure 2.2 tf.lite Interpreter.

2.2.4 Session

TensorFlow employs a dataflow graph to express a user's computation in terms of interdependencies between separate operations. This encourages you to use a low-level programming style in which you first characterise the dataflow graph and then create a TensorFlow session to run sections of it on a local and remote device. The example of session initialization in TensorFlow[5] is shown in Figure 2.3 below.

```
# start a session and run graph on blank image
sess = tf.Session()
sess.run(init)
inputs = {input_tensor: np.zeros([1,224,224,3])}
outputs = [net.probs]
print(sess.run(outputs, feed_dict=inputs))
```

Figure 2.3 Session Initialization in TensorFlow.

2.2.5 Evaluation of Accuracy

In order to offer feedback to the model during or after training, TensorFlow must assess the output accuracy that has been produced. The output layer produces unnormalized probabilities for each digit as a consequence. An example of how to plot the training accuracy is shown in Figure 2.4.

```
tf.scalar_summary("Training Accuracy", accuracy)
tf.scalar_summary("SomethingElse", foo)
summary_op = tf.merge_all_summaries()
writer = tf.train.SummaryWriter('/me/mydir/', graph=sess.graph)
```

Figure 2.4 Initializing Training Accuracy.

2.3 Keras Documentation

Keras[5] is a Python-based high-level programming interface that is optimised for use with TensorFlow. It was created with the goal of facilitating fast experimentation. A key to completing effective research is having the ability to go from thinking to action with the shortest amount of time feasible. The following is why we are utilising Keras[5] in this study:

- i. Allows for easy and rapid prototyping, as well as being extremely user-friendly.

- ii. Allows conventional convolutional and recurrent networks, as well as their combinations.
- iii. Runs reliably on both CPU and GPU.

Keras[5] central information structure is a model, or a way of composing layers. The Sequential model, which is a linear stack of layers, is the simplest type of model. Using the Keras[5] functional API, which allows you to construct arbitrary graphs of layers, is the best way to go for more complex architectures. In deep learning, the computer can occasionally overgeneralize a picture, which is known as overfitting. To overcome this, we "enhance" the image by applying a number of random changes. This, in turn, aids us in making use of all our training images, albeit in a little way. An example of image augmentation arrangement is shown in Figure 2.5.

```
# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
    rescale=1. / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)
```

Figure 2.5 Image Augmentation Arrangement.

The 'from *keras.preprocessing.image* import *ImageDataGenerator*' class is used to perform the image augmentation procedure described above. The above data augmentations has the following function.

- i. *rescale* is an option that allows us to double the amount of data accessible before performing any picture processing. The original images in this project include RGB coefficients ranging from 0-255, however these values are too high for our models to comprehend, thus values between 0 and 1 are targeted instead by scaling with a 1/255 factor.
- ii. *shear_range* is used to apply shear transformations in a random order.
- iii. *zoom_range* is useful for zooming in and out of photos at random.

- iv. *horizontal_flip* allows you to flip photos horizontally or on a level plane.

2.4 Python

Python[7] is a widely used high-level programming language for collaborative programming that was created by Guido van Rossum and initially released in 1991. This language enables the creation of clear programmes on a small and large scale. Python[7] features a dynamic sort system and programmable memory management, as well as support for a variety of programming ideal models, like object-oriented, basic, functional, and procedural programming. It boasts a massive and diverse library. Furthermore, for the time being, only Python[7] is compatible with the TensorFlow[5] library. Python[7] is frequently associated with intense sentiments among developers owing to the increased profitability it provides. The edit-test-debug sequence is extraordinarily rapid because there is no compilation phase. Python[7] programme troubleshooting is straightforward: a bug or poor input will never result in a segmentation fault. When the software discovers a flaw, it raises a particular instance. When the application fails to recognise the specific instance, a stack follow is printed. Finally, adding a handful problematic print statements to the program code is frequently the quickest way to troubleshoot a programme: the rapid edit-test-debug sequence makes this simple technique quite practical.

2.5 Anaconda

Anaconda[8] is a Python[7] and R programming language distribution aimed for simplifying package management and deployment in scientific computing (data science, machine learning applications, large-scale data processing, predictive analytics, and so on). Data-science software for Windows, Linux, and iOS Mac are included in the release. Anaconda, Inc., created by Peter Wang and Travis Oliphant in 2012, is responsible for its development and maintenance. Anaconda Distribution or Anaconda Individual Edition are other Anaconda, Inc. goods, whereas Anaconda Team Edition and Anaconda Enterprise Edition, both of which are not free, are other Anaconda, Inc. products. Over 250 packages are installed by default in the Anaconda installation, and over 7,500 more

open-source packages, including the conda package as well as virtual environment manager, may be added via PyPI. Anaconda Navigator, a graphical user interface, is included as a backup to the command-line interface (CLI).

2.6 Spyder

Spyder is a cross-platform, open-source Integrated Development Environment (IDE) for scientific Python[7] programming. Spyder works with a variety of popular Python packages, including NumPy, SciPy, Matplotlib, pandas, IPython, SymPy, and Cython, as well as other open-source applications. Spyder may be extended by first- and third-party plugins, and features interactive data inspection tools as well as Python-specific code process improvement and introspection tools like Pyflakes, Pylint, and Rope. Anaconda[8] makes it cross-platform, including versions for Windows, iOS Mac, and major Linux distributions like Arch Linux, Debian, Fedora, Gentoo Linux, openSUSE, and Ubuntu.

2.7 Article/Journal Reviews

2.7.1 London : Congestion Charge

London launched the congestion charge[9] concept in February 2003 to combat traffic congestion in the city. Non-residents driving private vehicles in the city centre between the hours of 7:00 a.m. and 6:30 p.m. will be fined £5 (\$8.60). Taxis and public transportation are excluded from the fee. Security cameras are strategically installed across the city to enforce this restriction. These cameras are capable of capturing licence plates and verifying whether or not the driver has paid the charge. Violators are fined £80 (\$138) and their cars are sometimes seized.



Figure 2.6 Congestion Charging Zones' Signs in London.

The congestion charge zone in London spans an area of eight square miles. When entering or exiting the congestion zone, there are directional signals and road markers to alert you. More information regarding operation hours can be found on advance notice signs [9].

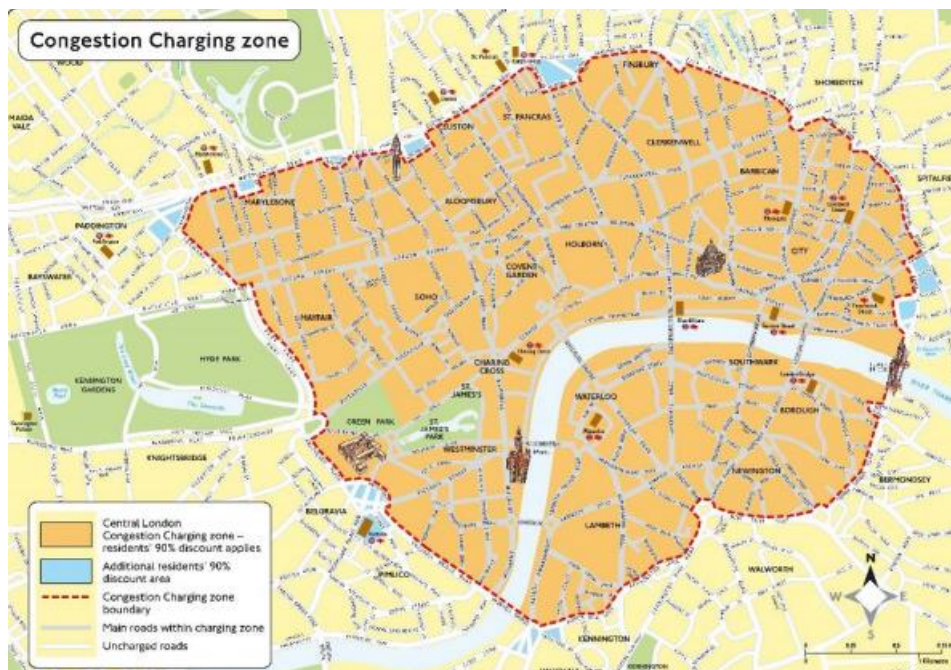


Figure 2.7 Map of Congestion Zone in London.

In London, the congestion charge is seen as a success. Within six months of deployment, the congestion rate had fallen by 30%. In the following years, fine revenue increased from \$138 million to \$172 million each year. This sum is utilised to improve the city's overall transportation system[9].Furthermore, the implementation of this legislation has had a good influence on the city by reducing greenhouse gas emissions by 20% yearly[9].

2.7.2 Baton Rouge: Continuous Intersection Flow

Baton Rouge is one city that has made significant progress in reducing traffic congestion. This city was the first to use Continuous Flow Intersection (CFI) road design[10].



Figure 2.8 Continuous Flow Intersection (Lee, 2013).

A Continuous Flow Intersection is an "innovative at-grade intersection that eliminates competing left turns at an intersection by positioning turning actions hundreds of feet ahead of the main intersection". This design allows for simultaneous left turn and mainline through movements, decreasing traffic congestion at the crossing. Furthermore, a large majority of junction accidents occur when making a left turn. "First 'Continuous-Flow' Intersection in Louisiana Opens in Baton Rouge," 2015, states that because this design prevents left turns, there would be fewer accidents at junctions. In addition, this design has been shown to be the most cost-effective. On Baton Rouge, the first CFI made

a significant improvement in traffic congestion. "During Pre-Hurricane Katrina evening rush-hour traffic, the average delay for each vehicle crossing the junction was 225 seconds, or roughly 4 minutes," Bruce added. The time elapsed is estimated to be 30 seconds per vehicle with the introduction of the CFI turn lanes, an almost 90% increase." [10]

2.7.3 Traffic Congestion in China

As the world's largest developing country, China has experienced severe traffic congestion. According to China's Ministry of Public Security (MPS), the country's motor vehicle inventory hit 325 million in 2018, up 15.56 million from the previous year's conclusion. The number of people who drive automobiles has risen to 407 million, up 22.36 million from the previous year. Meanwhile, the number of private automobiles has skyrocketed. In all, 187 million small and micro passenger cars were registered in the names of people, with private automobile ownership per 100 homes exceeding 40. (MPS, 2018). There were 61 cities with more than one million automobiles, 26 cities with more than two million cars, and eight cities with more than three million cars (MPS, 2018). For the past few years, China has committed to reducing traffic congestion by promoting Intelligent Transportation System (ITS) implementations, facilitating public transportation, and advancing infrastructure construction, as measured by a "Congestion Delay Index" (i.e., "the ratio of urban residents' average actual outgoing travel time to their travel time under free flowing") [11]. Despite the fact that China's national GDP and automobile ownership rate are increasing, its efforts appear to be paying off (MPS, 2018). According to the annual study produced by local navigation provider a map, which collaborates with the Ministry of Transportation's Scientific Research Institute and Alibaba Cloud, China's traffic situation improved last year. 51 of the 100 main cities have seen a reduction in traffic congestion, while 22 have seen an increase. The remaining 27 cities have remained unchanged from the previous year. The assessment is based on a ratio of time spent driving during peak commute periods to time spent commuting during free-flow hours, with a larger figure indicating more severe congestion. The index reveals that travelling during rush hours - from 7 to 9 a.m. and 5 to 7 p.m. - takes more than 1.8 times as long as travelling during off-peak hours in the 22 cities where traffic has gotten

worse. In China's largest cities, the Congestion Delay Index decreased by 2.45 percent in 2016 compared to 2015, which is equal to the level of 2015. (Figure 2.9).

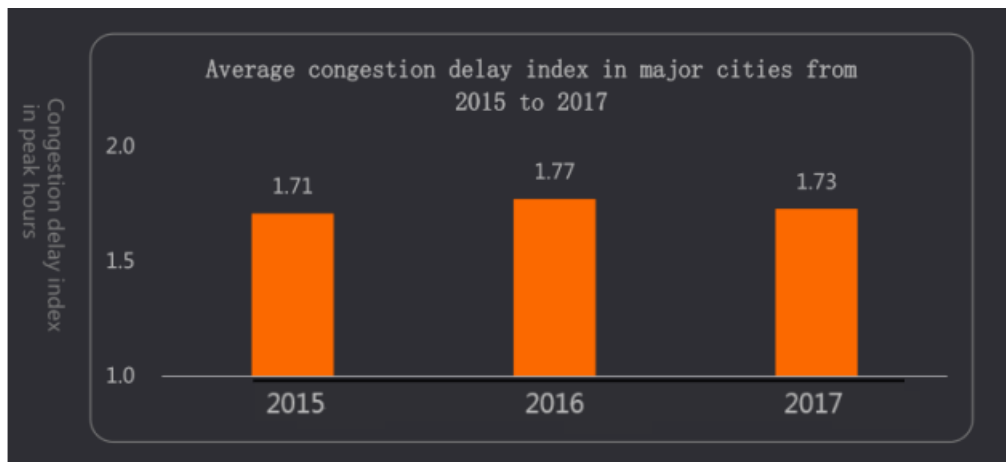


Figure 2.9 Average congestion delay index in major cities from 2015 to 2017 (Amap, 2018).

Shenzhen was rated fifth on the list of urban cities with reduced congestion, and it was also the city with the least congestion at the mega-city level. Apart from sophisticated ICT, rotating prohibitions on cars based on the final digit of the plate number are likely additional contributions to the improvement in several Chinese cities, according to the report's study [11].

2.7.4 Traffic Congestion of Urban Main Road under Accident Conditions

With China's continued urbanisation, automobile ownership is increasing, and urban traffic is becoming increasingly congested[11]. Simultaneously, urban traffic accidents occur often, causing traffic congestion, reduced road network operating efficiency, and societal economic losses. In order to dredge and minimise the chance of catastrophe, to lessen the effect of the road network, and to preserve the efficient and smooth operation of the road traffic patterns, it is vital to handle traffic issues on complex urban road networks. To address the issues, it is critical to have a thorough understanding and assessment of the development law, as well as the extent and severity of traffic congestion caused by accidents. This study examines the law of the formation and

expansion of road congestion under accident conditions, as well as the variables influencing the growth of the congestion situation and its mechanism, based on current research results in China and overseas. The severity of traffic congestion situation under the relevant traffic circumstances is analysed, and the forecast technique of the spatial influence scope generated by the propagation of accidental congestion on the road system space after the accident is explored.

Table 2.1 Classification table of average speed (km/h) of main road during peak hours.

Evaluation standard class	1st Clear	2nd Class	3rd Usually crowded	4th Usually crowded	5th Severe congestion
Very large/ A Class city	25 or higher	[22, 25)	[19, 22)	[16, 12)	[0 16 th)
B Class city	28 or higher	[25, 28)	[22, 25)	[19, 22)	[0, 3)
C/D Class city	30 or higher	[27, 30)	[24, 27)	[21, 24)	[0, 25)
Index	[22, 25)	[80,90)	[70, 80)	[60, 70)	[0, 60)

China's urban road traffic management evaluation index system (2012 edition) (Ministry of Public Security, 2012). As shown in table 2.1, some scholars use city scale, main peak time, and develop city proper scale of average velocity as traffic congestion measures for related research, evaluation class that corresponds to the smooth flow of traffic, first and second level 3 and level 4 for general crowded, five corresponding serious congestion [11].

2.7.5 Traffic Congestion and Remedial Measures at Traffic Mor in Pabna City, Bangladesh

Pabna is regarded as Bangladesh's most major commerce and manufacturing centre[12]. The area is under the influence of Rajshahi City, which is one of the country's fastest growing cities. Because of its convenient transit links to other regions of the nation, the neighbourhood is densely populated with industrial facilities. It is also critical for both Rajshahi and Pabna's food security. As a result, a study of the secondary city next to the metropolis is very important as part of the Rajshahi Metropolitan Development Program (RMDP). As a result, Pabna City (now Pabna Pourashava) has been chosen as the research area (Figure 2.10).



Figure 2.10 Pabna city map.

One of Pabna's most important intersections is Traffic Mor. In the larger perspective, the Mor road intersection is an important part of the city's current traffic infrastructure. At ground level, the Traffic Mor road intersection is a tee type road intersection[12]. The traffic flow at this crossroads is mixed, and both directions are two-way. The geometric aspects, traffic congestion, and traffic control devices at the Traffic Mor road intersection are investigated in this study.

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter goes into the approaches that were employed and the considerations that were considered for this project. It begins with a discussion of the project architecture, followed by a discussion of the project flow, as well as the techniques and Softwares used in this project to achieve the project's goal throughout the training purposes, as well as the importance of the project.

3.2 Project architecture

The YOLO Model[6], VGG16, VGG19, Xception, ResNet50, Inception50, InceptionV3, AlexNet, and other Convolutional Neural Network (CNN)[5] architectures may all be used for image training. In this study, the YOLO model architecture will be used because it already has learnt characteristics that are relevant to our classification challenge.

3.2.1 YOLO (You Only Look Once) Model architecture.

Using YOLO[6] to prepare photos is simple and straightforward. The system resizes the data image to 448 x 448 pixels, applies a single convolutional arrange on it, then edges the next place based on the model's certainty. YOLO[6] is a well-known deep learning-based method for picture identification problems. Figure 3.1 shows how it

separates the image into described bounding boxes and then performs a characterisation computation in parallel for these containers to determine which object class they belong to. After separating these classes, it proceeds to intelligently combine these examples to form an optimal bounding box around the objects.

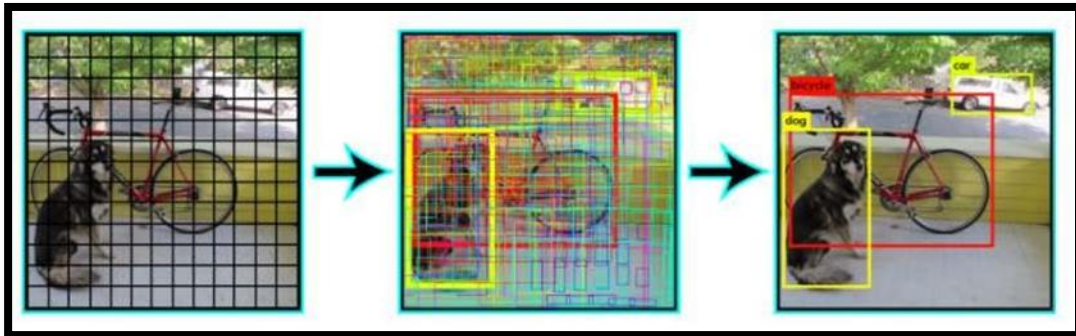


Figure 3.1 Image Separation in Yolo Architecture.

The entirety of this is performed in parallel so that it may continue to operate in a progressive manner, processing up to 40 images per second. Regardless of the fact that it has a lower level of execution than its RCNN companion, it has enough consistency to be useful in daily situations. The architecture of YOLO[6] is seen in Figure 3.2:

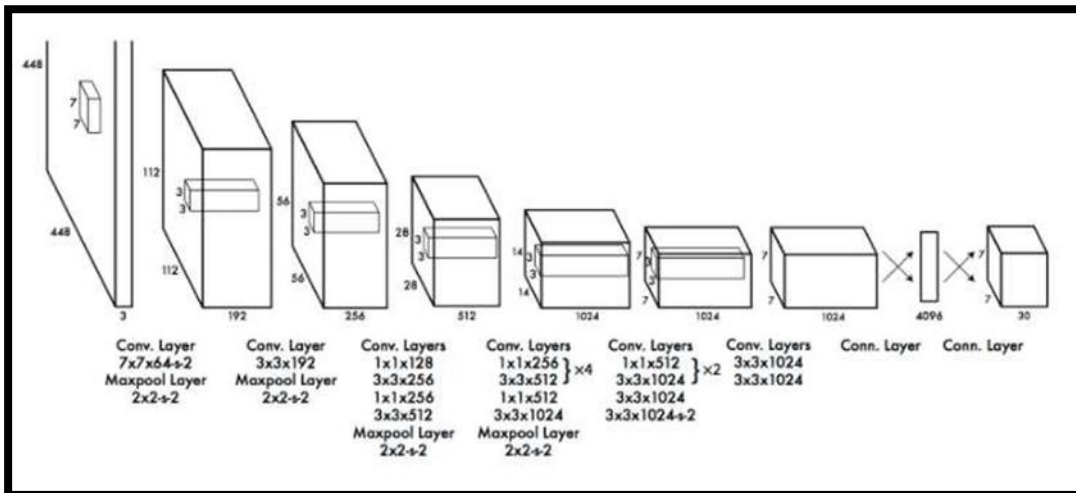


Figure 3.2 Yolo Architecture.

3.2.2 DeepSort architecture

Deep Sort[13] is a new tracking method that enhances Simple Online and Real-time Tracking, and it has shown excellent outcomes in the Multiple Object Tracking (MOT) issue. Each frame contains more than one item to track in the MOT issue setup. The technique necessitates a total score from all detections and tracks. The IOU metric is computed across the video frames of the detection and the bounding boxes of the tracks, and Deep Sort[13] uses it to generate this score. The Kalman filter assigned to each track predicts the latter. It also employs appearance data to follow objects through occlusions, minimising the number of identification shifts. Deep Sort[13] enables a programmer to add this functionality by computing deep features for each bounding box and factoring in the tracking algorithm based on deep feature similarity.

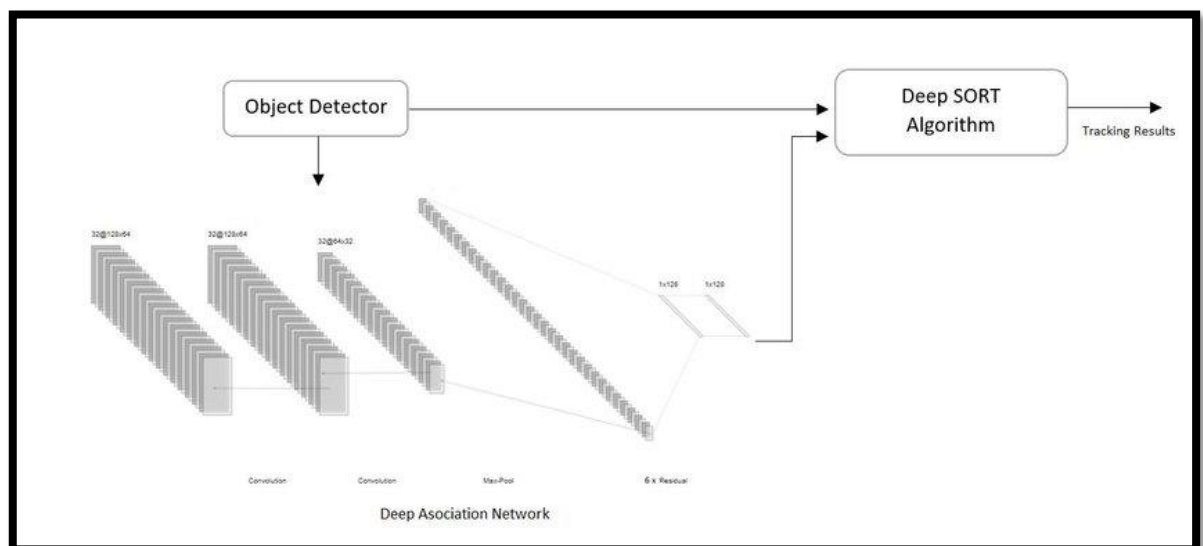


Figure 3.3 Deep Sort Architecture

Figure 3.3 above shows the network's CNN design of Deep Sort[13] architecture which consist of a residual network, two convolutional layers, and six residual blocks make up the deep association clock. The dense layer's calculated features map has a dimensionality of 128. To be consistent with the cosine appearance metric, a batch layer and 12 normalisation projects are added to the unit hypersphere.

3.3 Visual Masking

Visual masking[14] is a visual perception process. It occurs when the presence of another image, called a mask, reduces the appearance of one image, called a target. It's possible that the target isn't visible, or that it has a low contrast or brightness. Forward masking, reverse masking, and simultaneous masking are the three possible time configurations for masking. The mask comes before the target in forward masking. Backward masking is when the mask moves in the opposite direction of the target. Simultaneous masking shows both the mask and the target at the same time. The graphic below shows an example of how the *Open CV*[14] initialises a visual masking, which was the major architecture employed in this project.

```
vid_fps = int(vid.get(cv2.CAP_PROP_FPS))
mask = cv2.imread('C:/Thesis/yolov4-deepsort/data/video/mask1.jpg')
```

Figure 3.4 Initializing Masking Technique in 'OpenCV'.

3.3.1 Target-to-mask spatial separation

This masking pattern[14] utilize a layer of filter, where the filter layer was designed in black in colour and the targeted/masking point marked in white in colour. When there is pattern masking, suppression can be noticed in both forward and backward masking, but not when there is metacontrast. Simultaneous masking, on the other hand, will make it easier to see the target during pattern masking. When metacontrast is paired with either contemporaneous or forward masking, facilitation occurs. This is due to the fact that lateral propagation requires time for the mask to achieve the target's position. The time needed for lateral propagation rises as the target grows further away from the

mask. As the mask moves closer to the target, the masking effect will get stronger. The visual masking technique used in the system is shown Figure 3.5.

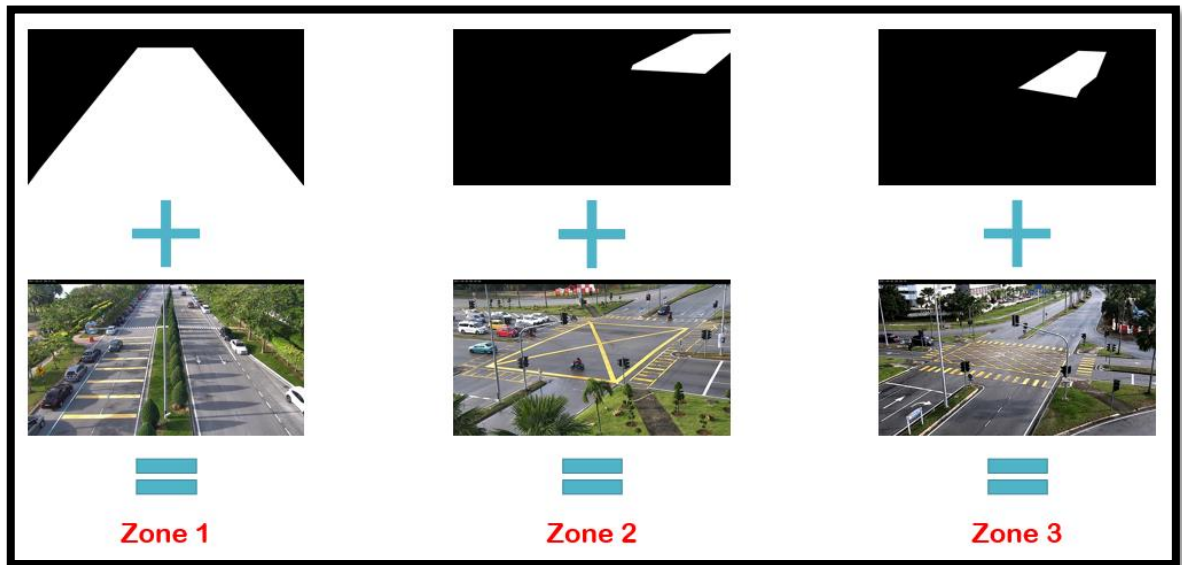


Figure 3.5 Masking Technique used in this system.

Figure 3.5 above shows the visual masking technique used to identify the detection zone for system to count the number of vehicles and to classify the status of congestion. There are totally three types of zone to be considered. The first zone, has been designed to detect two different traffic lane, which are left lane and right lane. Whereas the second and third zone, has the design to detect the focussed traffic lane. This is due to clarified that the system has multi features in detecting a zone, which are the system itself can focus a specified targeted traffic lane and it also can focus multi traffic lanes.

3.4 Project Block diagram

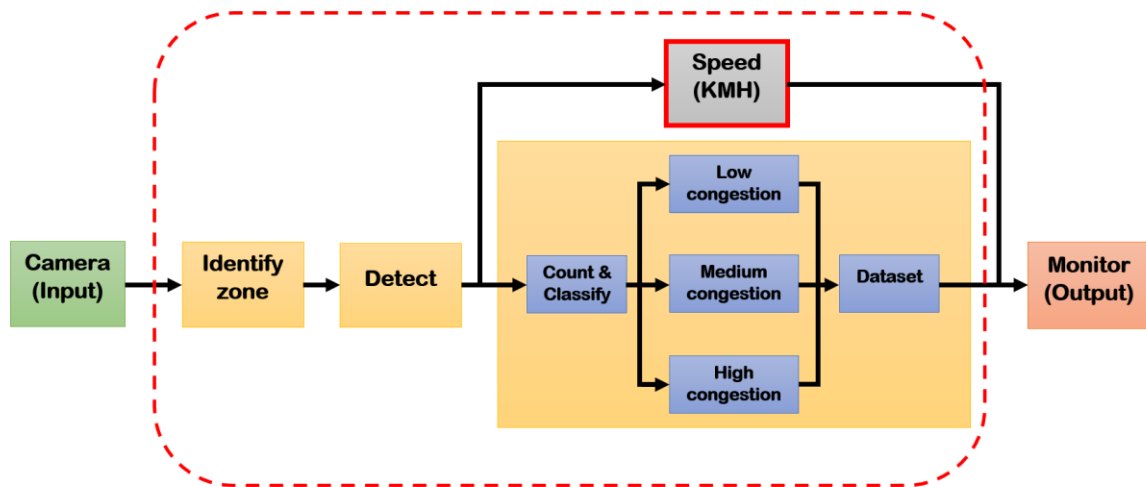
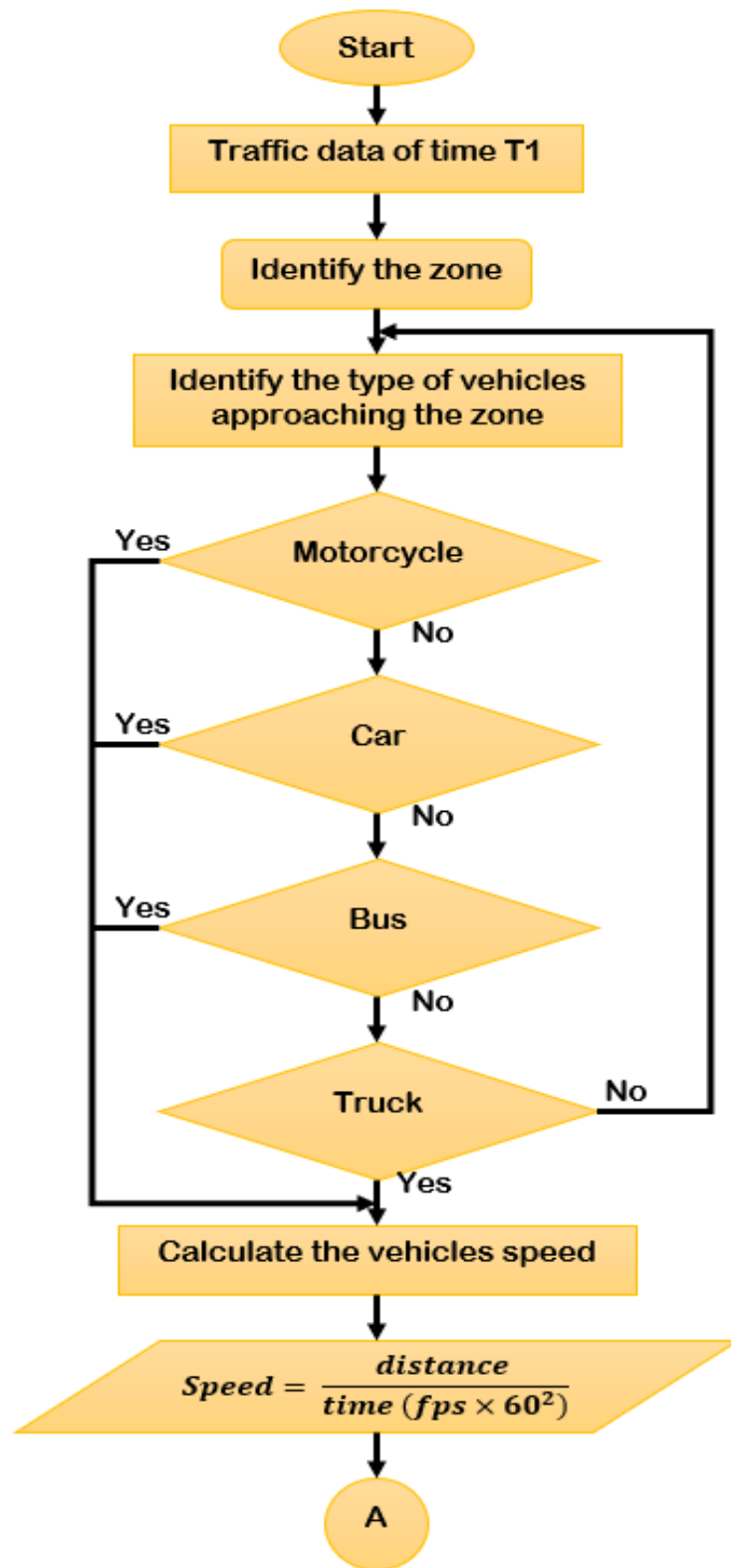


Figure 3.6 Project Block diagram.

The block diagram seen above depicts the project's complete life cycle. First, the project's input, which is the camera's live captured image/video. Then it moves on to the identifying zone, where it uses the YOLO model[6] architecture to detect things (cars, motorbikes, trucks, and so on). The algorithm will then compute the number of items identified in the identification zone and categorise them into three groups: “Low Congestion,” “Medium Congestion,” and “High Congestion.” The flowchart graphic will illustrate how the categorization will be depending on the number of items observed. Following that, all of the data will be collected and saved in order to evaluate the recorded footages. Finally, all identified items and congestion classifications, as well as the live acquired image/video from the roadside CCTV camera, will be presented on the monitor.

3.5 Project Flowchart diagram



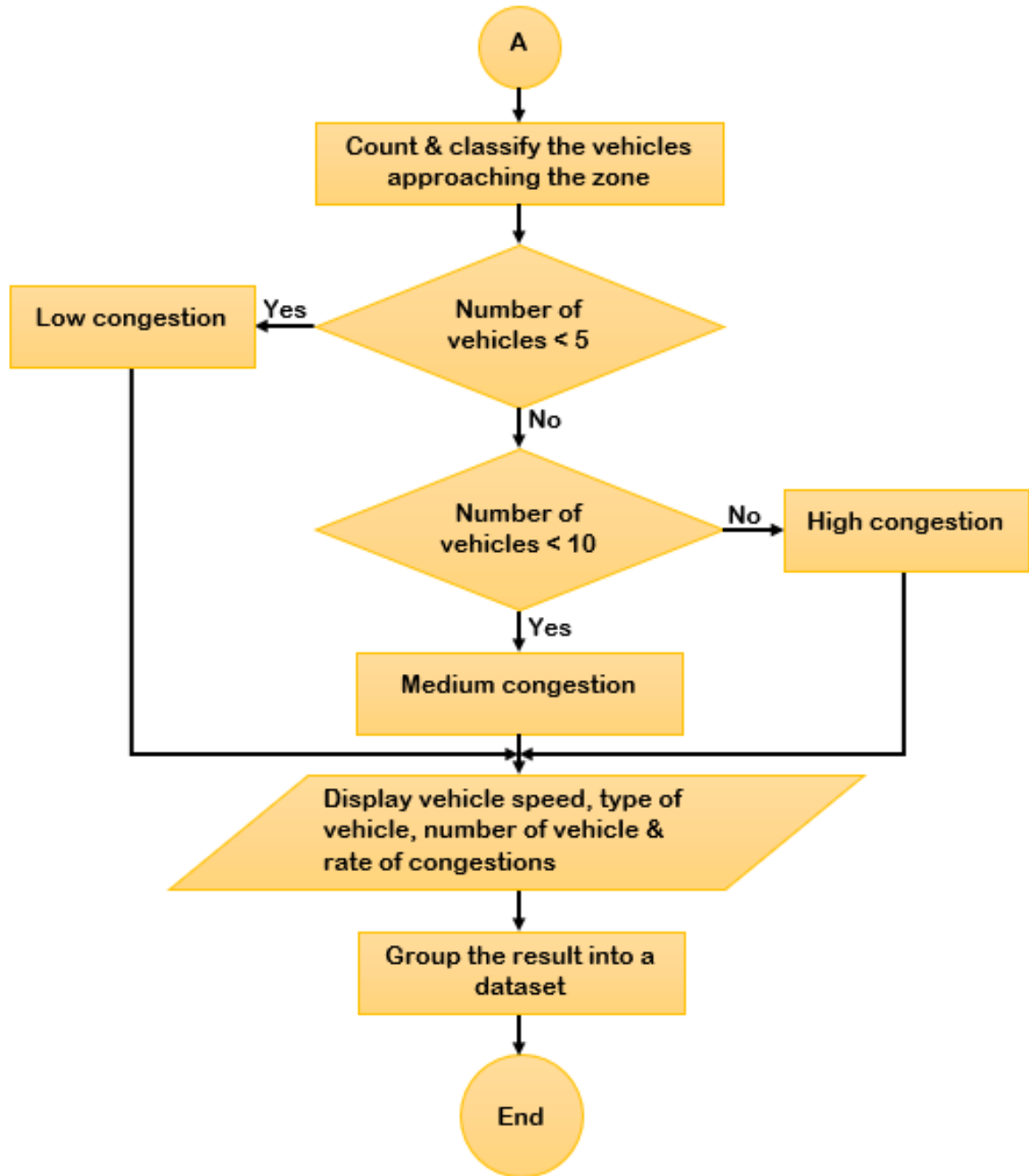


Figure 3.7 Project Flowchart diagram.

The flowchart diagram shown above depicts how the system is able to classify traffic congestion circumstances. To begin, a new traffic data set based on real-time and date will be developed for the purpose of preserving and examining recorded footage. Then, using the YOLO model[6] architecture, it travels to the identifying zone and detects the number of vehicles (cars, motorbikes, bus, and trucks) approaching the zone. In addition, the system also calculates the speed of the identified vehicle in KM/H. The system will then make a choice based on the programming. According to the project's

requirements, the system must categorise traffic congestion into three categories: “Low Congestion,” “Medium Congestion,” and “High Congestion.” As a result, the choice is divided into three categories. For case 1, if the number of vehicles counted is less than five, the algorithm will classify it as “Low Congestion.” For case 2, if the number of vehicles counted is less than 10, the algorithm will classify it as “Medium Congestion.” For case 3, if the number of vehicles counted exceeds 10, the algorithm will classify it as “High Congestion.” Finally, all of the data will be aggregated and saved so that the captured footages may be reviewed.

CHAPTER 4

RESULTS AND DISCUSSION

4.1 Introduction

This chapter highlights the findings and analysis of the architecture employed to meet the project's goal throughout training, as well as its significance.

4.2 Dataset

A single phase of training was done in this experiment, with the goal of classifying road objects into each of four different categories: motorcycle, car (car, taxi, mpv, and van), bus, and truck. The fact that I am inexperienced to Deep Learning[2], the major goal was to create a system for categorising traffic congestion. Identifying the observed automobiles based on their categories was a terrific experience towards the conclusion of this project, since the dataset received from '*coco.names*' is quite unique and perfect for this picture classification problem. For training purposes, over 80 types of categorization of objects illustrated in Figure 4.1 that correspond to the following classes were employed, although just four classes were needed for this system to meet the project's aim.

person	bird	suitcase	fork	chair	toaster
bicycle	cat	frisbee	knife	sofa	sink
car	dog	skis	spoon	pottedplant	refrigerator
motorbike	horse	snowboard	bowl	bed	book
aeroplane	sheep	sports ball	banana	diningtable	clock
bus	cow	kite	apple	toilet	vase
train	elephant	baseball bat	sandwich	tvmonitor	scissors
truck	bear	baseball glove	orange	laptop	teddy bear
boat	zebra	skateboard	broccoli	mouse	hair drier
traffic light	giraffe	surfboard	carrot	remote	toothbrush
fire hydrant	backpack	tennis racket	hot dog	keyboard	
stop sign	umbrella	bottle	pizza	cell phone	
parking meter	handbag	wine glass	donut	microwave	
bench	tie	cup	cake	oven	

Figure 4.1 Classes in 'coco.names'.

The classifications in this system are used to identify the type of vehicle that the system has identified. To produce the best results, the coco.names dataset is trained many times. The figure 4.2 below gives an example of how the classes were specified in the software, which was one of the project's main framework.

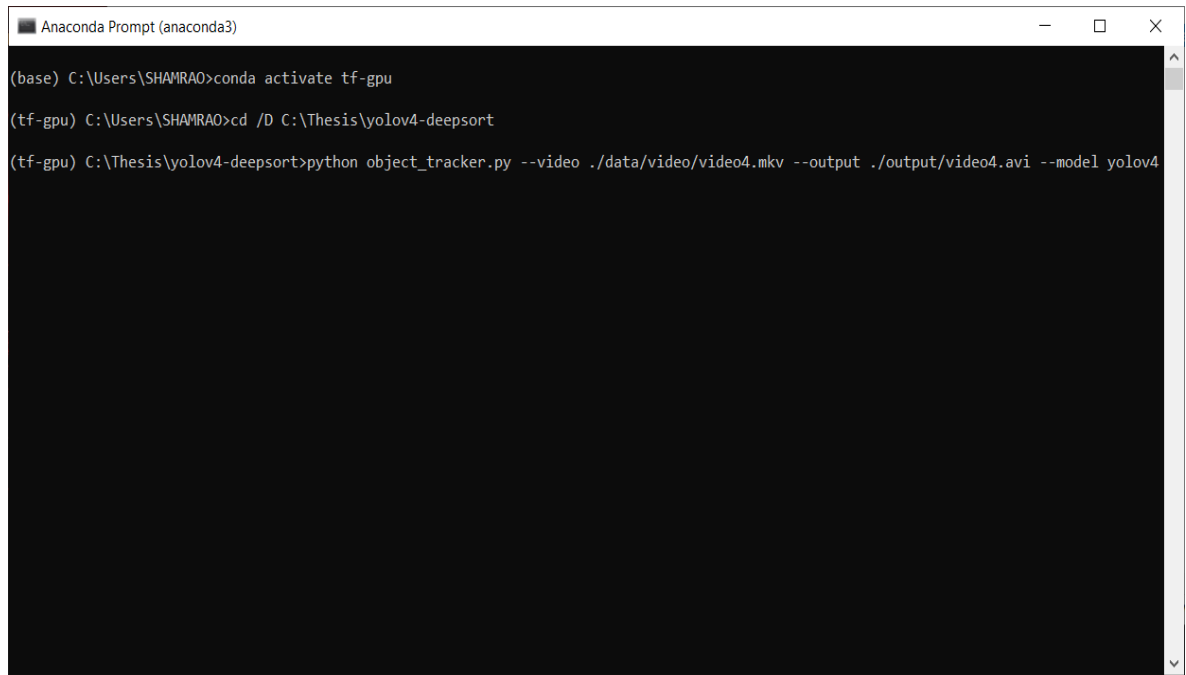
```
class_names = utils.read_class_names(coco.names)
# by default allow all classes in .names file
# allowed_classes = list(class_names.values())
# custom allowed classes (uncomment line below to customize tracker for only people)
allowed_classes = ['motorcycle', 'car', 'bus', 'truck']
```

Figure 4.2 Initializing Allowed Classes.

4.3 Anaconda prompt

Anaconda prompt is a Python 3.3+ package for prompting input on the command line. There are no dependencies in the Python[7] code. A few recommendations should be followed in order to process the output from the system: Tap Start, then search for or

pick Anaconda Prompt out from menu. Then, using the enter key on the PC, input the code shown in Figure 4.3 line by line.

A screenshot of an Anaconda Prompt window. The window title is "Anaconda Prompt (anaconda3)". The terminal shows the following commands and their prompts:

```
(base) C:\Users\SHAMRAO>conda activate tf-gpu
(tf-gpu) C:\Users\SHAMRAO>cd /D C:\Thesis\yolov4-deepsort
(tf-gpu) C:\Thesis\yolov4-deepsort>python object_tracker.py --video ./data/video/video4.mkv --output ./output/video4.avi --model yolov4
```

Figure 4.3 Anaconda Prompt window.

4.4 Results and Analysis

As this software was run on the GPU, it took less time to accomplish the training than if it had been run on the CPU. The findings were optimal, indicating that the detection and categorization of traffic congestion is quite accurate. Furthermore, the identifications of the different types of vehicles are exact. The accuracy of determining the speed of the identified vehicles is less than ideal. This is owing to a miscalculation in the initial road distance estimate and an inconvenient location of the roadside CCTV camera. If the aforementioned flaws can be addressed, the results produced may be more accurate and superior than this.

4.4.1 Vehicle Detection



Figure 4.4 Vehicle detection result.

As shown in Figure 4.4 above a pink square emerged around the car, this indicates that the system has detected an object at the detection zone. The colour and the thickness of the square can be adjusted in the system program as shown in the Figure 4.5 below.

```
Boxes = tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),  
color = [i * 255 for i in color]
```

Figure 4.5 Altering the colour and the thickness of the square.

4.4.2 Vehicle Identification and Counting



Figure 4.6 Vehicle Identification and Counting result.

Figure 4.6 shows two images on the right shows the classes of vehicle (Car, Truck, Bus, Motorbike) along with the number of vehicles that has been detected. The image on the left shows two cars waiting for the traffic lights to turn green. From the Figure 4.6, it is seen that the system has detected two vehicles on the detection zone and has identified

the vehicle is cars. Furthermore, the system displayed the digit '2' beside the class name labelled car, indicating that this system has a high level of accuracy in counting vehicles that has been detected on the detection zone.

4.4.3 Average Speed Counting



Figure 4.7 Average Speed Counting result.

Figure 4.7 above shows that a type of vehicle was detected in the detection zone with the average speed of 40.7 km/h. The average speed can be calculated by using the programming formula shown in Figure 4.8 below.

```
# dist = (pix_diff * mpp) / 1000
# time_hours = 1 / (60 * 60)
# speed = dist/time_hours

dist = pix_diff * mpp
time_sec = 1 / vid_fps
speed = dist / time_sec
```

Figure 4.8 Formula for Average Speed.

Mathematical formula to calculate speed:

$$Speed = \frac{distance (KM)}{time (H)}$$

4.4.4 Classifying Congestion Rate

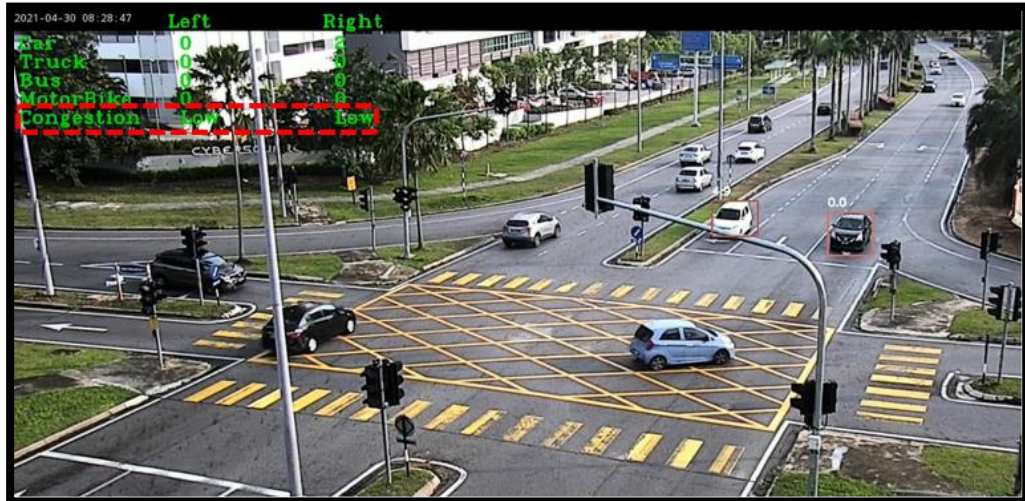


Figure 4.9 Result classifying Low Congestion Rate.

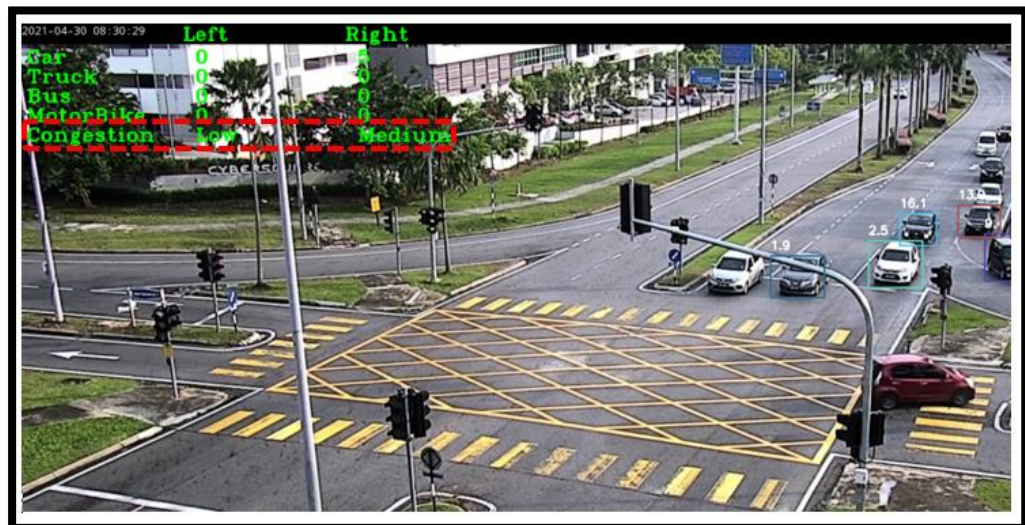


Figure 4.10 Result classifying Medium Congestion Rate.

Figure 4.9 & 4.10 shows that the result of classify the traffic condition of the targeted area. The system classifies the traffic congestion rate based on the number of vehicles detected in the detection zone. As mention in chapter 3, the number of vehicles detected has three conditions to state the traffic congestion rate. If the number of vehicles is less than 5 the congestion rate be shown as 'Low', if the number of vehicles is less than 10 the congestion rate be shown as 'Medium', and if the number of vehicles is more than

10 the congestion rate be shown as '*High*'. In Figure 4.9, the congestion rate shown as '*Low*' because there are only two vehicles has been detected by the system. In Figure 4.10, the congestion rate shown as '*Medium*' because there are more than 5 vehicles has been detected by the system.

CHAPTER 5

CONCLUSION

5.1 Introduction

The project's objectives had been met. The Vision Based Traffic Congestion Classifying Monitoring System (VBTCMS) has the capacity of identifying and classifying all vehicles that pass through its detection zone. Most crucially, the system can detect traffic congestion rates in real time, which is necessary to meet the goals. To summary, the VBTCMS has a significant influence on Artificial Intelligence 4.0, which sees computer vision systems replace manual systems. The concept of a VBTCMS may be used to a wide range of applications and research areas. Some computer vision project researchers include SenseTime, MegVii, viso.ai, and NAUTO. As a result, the VBTCMS project is current and has a broad variety of possible applications in engineering, scientific research, and design. For forthcoming engineers, the experience and talents of this project will bring a great deal of value and opportunities.

5.2 Future Recommendation

Deep Learning is a fascinating field in which we may immerse ourselves. As a future suggestion for this research, we may use several types of architectures to increase the detection speed and accuracy derived from the system dataset. Varied designs give different detection speeds and accuracy, which might make this traffic congestion monitoring system easier to use.

Aside from that, the YOLO architecture might be fine-tuned and trained using additional dataset as desired by the user. YOLO is a simple and fascinating design to learn, with excellent results.

Furthermore, the arrangement of the roadside CCTV cameras site hampered the monitoring potential. This is a crucial limiting factor in monitoring capability that might be addressed if authorities properly site the roadside CCTV cameras. As a result, future highways may be completely occupied by roadside CCTV cameras.

Adding new lessons and a longer duration of CCTV video footages might also make the training more complicated and demanding. This method, on the other hand, might be used to assess the accuracy of the forecast and determine whether Deep Learning is the way of the future. Last but not least, because data training can take a long time and requires a lot of patience, it's better to do it with a powerful GPU. As a result, an optimal GPU is recommended in order to take this research to the next level.

REFERENCES

- [1] “Traffic Congestion - an overview | ScienceDirect Topics.” <https://www.sciencedirect.com/topics/earth-and-planetary-sciences/traffic-congestion> (accessed Feb. 12, 2022).
- [2] B. Liu and Z. Ding, “A distributed deep reinforcement learning method for traffic light control,” *Neurocomputing*, Dec. 2021, doi: 10.1016/J.NEUCOM.2021.11.106.
- [3] I. O. Olayode, L. K. Tartibu, M. O. Okwu, and U. F. Uchechi, “Intelligent transportation systems, un-signalized road intersections and traffic congestion in Johannesburg: a systematic review,” *Procedia CIRP*, vol. 91, pp. 844–850, Jan. 2020, doi: 10.1016/J.PROCIR.2020.04.137.
- [4] J. Zhao, H. Xu, Y. Tian, and H. Liu, “Towards application of light detection and ranging sensor to traffic detection: an investigation of its built-in features and installation techniques,” *Journal of Intelligent Transportation Systems*, Feb. 2022, doi: 10.1080/15472450.2020.1807346.
- [5] L. Parisi, R. Ma, N. RaviChandran, and M. Lanzillotta, “hyper-sinh: An accurate and reliable function from shallow to deep learning in TensorFlow and Keras,” *Machine Learning with Applications*, vol. 6, p. 100112, Dec. 2021, doi: 10.1016/J.MLWA.2021.100112.
- [6] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, “A Review of Yolo Algorithm Developments,” *Procedia Computer Science*, vol. 199, pp. 1066–1073, Jan. 2022, doi: 10.1016/J.PROCS.2022.01.135.
- [7] H. Izadkhah, “An introduction of Python ecosystem for deep learning,” *Deep Learning in Bioinformatics*, pp. 31–66, Jan. 2022, doi: 10.1016/B978-0-12-823822-6.00010-X.

- [8] P. Mccaffrey, "Packages, interactive computing, and analytical documents," *An Introduction to Healthcare Informatics*, pp. 159–174, Jan. 2020, doi: 10.1016/B978-0-12-814915-7.00012-0.
- [9] B. Berisha, "Alleviating Traffic Congestion in Prishtina." [Online]. Available: <http://scholarworks.rit.edu/theses>
- [10] S. Jan-Evert Nilsson Eric Markus Author Wenjie Zhang, "Managing Traffic Congestion-Case study of Hangzhou," 2010.
- [11] J. Wang and J. Sun, "Research on Traffic Congestion of Urban Main Road under Accident Conditions."
- [12] A. Saha, B. Ahmed, M. Rahman, T. Tasnim Nahar, A. K. Saha, and T. T. Nahar, "Analysis of Traffic Congestion and Remedial Measures at Traffic Mor in Pabna City, Bangladesh International Journal of Recent Development in Engineering and Technology Website: www.ijrdet.com Analysis of Traffic Congestion and Remedial Measures at Traffic Mor in Pabna City, Bangladesh," 2013. [Online]. Available: www.ijrdet.com
- [13] M. Oplenskedal, P. Herrmann, and A. Taherkordi, "DeepMatch2: A comprehensive deep learning-based approach for in-vehicle presence detection," *Information Systems*, p. 101927, Oct. 2021, doi: 10.1016/J.IS.2021.101927.
- [14] S. Paul, A. Norkin, and A. C. Bovik, "On visual masking estimation for adaptive quantization using steerable filters," *Signal Processing: Image Communication*, vol. 96, p. 116290, Aug. 2021, doi: 10.1016/J.IMAGE.2021.116290.

APPENDIX

Project source code.

```
import os

# comment out below line to enable tensorflow logging outputs

os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'

import time

import tensorflow as tf

physical_devices = tf.config.experimental.list_physical_devices('GPU')

if len(physical_devices) > 0:

    tf.config.experimental.set_memory_growth(physical_devices[0], True)

from absl import app, flags, logging

from absl.flags import FLAGS

import core.utils as utils

from core.yolov4 import filter_boxes

from tensorflow.python.saved_model import tag_constants

from core.config import cfg
```

```
from PIL import Image

import cv2

import numpy as np

import matplotlib.pyplot as plt

from tensorflow.compat.v1 import ConfigProto

from tensorflow.compat.v1 import InteractiveSession

# deep sort imports

from deep_sort import preprocessing, nn_matching

from deep_sort.detection import Detection

from deep_sort.tracker import Tracker

from tools import generate_detections as gdet

flags.DEFINE_string('framework', 'tf', '(tf, tflite, trt)')

flags.DEFINE_string('weights', './checkpoints/yolov4-416',

                    'path to weights file')

flags.DEFINE_integer('size', 416, 'resize images to')

flags.DEFINE_boolean('tiny', False, 'yolo or yolo-tiny')
```

```
flags.DEFINE_string('model', 'yolov4', 'yolov3 or yolov4')
```

```
flags.DEFINE_string('video', './data/video/test.mp4', 'path to input video or set to 0 for webcam')
```

```
flags.DEFINE_string('output', None, 'path to output video')
```

```
flags.DEFINE_string('output_format', 'XVID', 'codec used in VideoWriter when saving video to file')
```

```
flags.DEFINE_float('iou', 0.45, 'iou threshold')
```

```
flags.DEFINE_float('score', 0.50, 'score threshold')
```

```
flags.DEFINE_boolean('dont_show', False, 'dont show video output')
```

```
flags.DEFINE_boolean('info', False, 'show detailed info of tracked objects')
```

```
flags.DEFINE_boolean('count', False, 'count objects being tracked on screen')
```

```
def main(_argv):
```

```
    # Definition of the parameters
```

```
    max_cosine_distance = 0.4
```

```
    nn_budget = None
```

```
    nms_max_overlap = 1.0
```

```
    # initialize deep sort
```

```
model_filename = 'model_data/mars-small128.pb'

encoder = gdet.create_box_encoder(model_filename, batch_size=1)

# calculate cosine distance metric

metric = nn_matching.NearestNeighborDistanceMetric("cosine",
max_cosine_distance, nn_budget)

# initialize tracker

tracker = Tracker(metric)

# load configuration for object detector

config = ConfigProto()

config.gpu_options.allow_growth = True

session = InteractiveSession(config=config)

STRIDES, ANCHORS, NUM_CLASS, XYSCALE = utils.load_config(FLAGS)

input_size = FLAGS.size

video_path = FLAGS.video

# load tfLite model if flag is set

if FLAGS.framework == 'tflite':
```

```
interpreter = tf.lite.Interpreter(model_path=FLAGS.weights)

interpreter.allocate_tensors()

input_details = interpreter.get_input_details()

output_details = interpreter.get_output_details()

print(input_details)

print(output_details)

# otherwise load standard tensorflow saved model

else:

    saved_model_loaded = tf.saved_model.load(FLAGS.weights,
tags=[tag_constants.SERVING])

    infer = saved_model_loaded.signatures['serving_default']

# begin video capture

try:

    vid = cv2.VideoCapture(int(video_path))

except:

    vid = cv2.VideoCapture(video_path)
```



```

out = None

# get video ready to save locally if flag is set

if FLAGS.output:

    # by default VideoCapture returns float instead of int

    width = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH))

    height = int(vid.get(cv2.CAP_PROP_FRAME_HEIGHT))

    fps = int(vid.get(cv2.CAP_PROP_FPS))

    codec = cv2.VideoWriter_fourcc(*FLAGS.output_format)

    out = cv2.VideoWriter(FLAGS.output, codec, fps, (width, height))

#####

vid_fps = int(vid.get(cv2.CAP_PROP_FPS))

mask = cv2.imread('C:/Thesis/yolov4-deepsort/data/video/mask1.jpg')

frame_num = 0

ref_bbox = np.array([], dtype = np.int32)

#####

# while video is running

```

```

while True:

    return_value, frame = vid.read()

    center = int(vid.get(cv2.CAP_PROP_FRAME_WIDTH)/2)

    if return_value:

        pframe = cv2.bitwise_and(src1 = frame, src2 = mask)

        pframe = cv2.cvtColor(pframe, cv2.COLOR_BGR2RGB)

        image = Image.fromarray(frame)

    else:

        print('Video has ended or failed, try a different video format!')

        break

    frame_num +=1

    print('Frame #: ', frame_num)

    frame_size = pframe.shape[:2]

    image_data = cv2.resize(frame, (input_size, input_size))

    image_data = image_data / 255.

    image_data = image_data[np.newaxis, ...].astype(np.float32)

```

```

start_time = time.time()

# run detections on tflite if flag is set

if FLAGS.framework == 'tflite':

    interpreter.set_tensor(input_details[0]['index'], image_data)

    interpreter.invoke()

    pred = [interpreter.get_tensor(output_details[i]['index']) for i in
range(len(output_details))]

    # run detections using yolov3 if flag is set

    if FLAGS.model == 'yolov3' and FLAGS.tiny == True:

        boxes, pred_conf = filter_boxes(pred[1], pred[0], score_threshold=0.25,

                                        input_shape=tf.constant([input_size, input_size]))

    else:

        boxes, pred_conf = filter_boxes(pred[0], pred[1], score_threshold=0.25,

                                        input_shape=tf.constant([input_size, input_size]))

    else:

        batch_data = tf.constant(image_data)

```

```
pred_bbox = infer(batch_data)
```

```
for key, value in pred_bbox.items():
```

```
    boxes = value[:, :, 0:4]
```

```
    pred_conf = value[:, :, 4:]
```

```
boxes, scores, classes, valid_detections =  
tf.image.combined_non_max_suppression(  

```

```
    boxes=tf.reshape(boxes, (tf.shape(boxes)[0], -1, 1, 4)),
```

```
    scores=tf.reshape(  

```

```
        pred_conf, (tf.shape(pred_conf)[0], -1, tf.shape(pred_conf)[-1])),
```

```
    max_output_size_per_class=50,
```

```
    max_total_size=50,
```

```
    iou_threshold=FLAGS.iou,
```

```
    score_threshold=FLAGS.score
```

```
)
```

```
# convert data to numpy arrays and slice out unused elements
```

```

num_objects = valid_detections.numpy()[0]

bboxes = boxes.numpy()[0]

bboxes = bboxes[0:int(num_objects)]

scores = scores.numpy()[0]

scores = scores[0:int(num_objects)]

classes = classes.numpy()[0]

classes = classes[0:int(num_objects)]

# format bounding boxes from normalized ymin, xmin, ymax, xmax ---> xmin,
ymin, width, height

original_h, original_w, _ = pframe.shape

bboxes = utils.format_boxes(bboxes, original_h, original_w)

# store all predictions in one parameter for simplicity when calling functions

pred_bbox = [bboxes, scores, classes, num_objects]

# read in all class names from config

class_names = utils.read_class_names(cfg.YOLO.CLASSES)

# by default allow all classes in .names file

```

```

# allowed_classes = list(class_names.values())

# custom allowed classes (uncomment line below to customize tracker for only
people)

allowed_classes = ['motorcycle', 'car', 'bus', 'truck']

# loop through objects and use class index to get class name, allow only classes in
allowed_classes list

names = []

deleted_indx = []

for i in range(num_objects):

    class_indx = int(classes[i])

    class_name = class_names[class_indx]

#####

# if np.all(mask[int((bboxes[i][1] + bboxes[i][3])/2), int((bboxes[i][0] +
bboxes[i][2])/2)] == 0):

    if np.all(mask[int(bboxes[i][1]), int(bboxes[i][0])] == 0):

        deleted_indx.append(i)

```

```
#####
```

```
    if class_name not in allowed_classes:

        deleted_indx.append(i)

    else:

        names.append(class_name)

names = np.array(names)

count = len(names)

bboxes = np.delete(bboxes, deleted_indx, axis=0)

scores = np.delete(scores, deleted_indx, axis=0)

# print(bboxes[1])

# encode yolo detections and feed to tracker

features = encoder(frame, bboxes)

detections = [Detection(bbox, score, class_name, feature) for bbox, score,
class_name, feature in zip(bboxes, scores, names, features)]

#initialize color map

cmap = plt.get_cmap('tab20b')
```

```

colors = [cmap(i)[:3] for i in np.linspace(0, 1, 20)]

# run non-maxima supression

boxes = np.array([d.tlwh for d in detections])

scores = np.array([d.confidence for d in detections])

classes = np.array([d.class_name for d in detections])

indices = preprocessing.non_max_suppression(boxes, classes, nms_max_overlap,
scores)

detections = [detections[i] for i in indices]

cv2.putText(frame, "Left", (250, 25),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)

cv2.putText(frame, "Right", (500, 25),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)

cv2.putText(frame, "Car", (10, 60), cv2.FONT_HERSHEY_COMPLEX_SMALL,
1.5, (0, 255, 0), 2)

cv2.putText(frame, "Truck", (10, 90),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)

cv2.putText(frame, "Bus", (10, 120),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)

cv2.putText(frame, "MotorBike", (10, 150),
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)

```



```
cv2.putText(frame, "Congestion", (10, 180),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
#####
```

```
unique_classes = np.unique(classes).tolist()
```

```
# print(unique_classes, type(unique_classes))
```

```
classes_list = classes.tolist()
```

```
classes_count = {i : classes_list.count(i) for i in unique_classes}
```

```
#####
```

```
# Call the tracker
```

```
tracker.predict()
```

```
tracker.update(detections)
```

```
#####
```

```
left_veh = 0
```

```
right_veh = 0
```

```
car = [0, 0]
```

```
truck = [0, 0]
```

```
bus = [0, 0]

motorbike = [0, 0]

congestion = ['', '']

speed = 0.0

dist = 0

mpp = 1/10    # meter per pixel

#####

# update tracks

for track in tracker.tracks:

    if not track.is_confirmed() or track.time_since_update > 1:

        continue

    bbox = track.to_tlbr()

    class_name = track.get_class()

    if bbox[0] < center:

        left_veh += 1

        if class_name == "car":
```

```
        car[0] += 1

    elif class_name == "truck":

        truck[0] += 1

    elif class_name == "bus":

        bus[0] += 1

    elif class_name == "motorbike":

        motorbike[0] += 1

else:

    right_veh += 1

    if class_name == "car":

        car[1] += 1

    elif class_name == "truck":

        truck[1] += 1

    elif class_name == "bus":

        bus[1] += 1

    elif class_name == "motorbike":
```

```

        motorbike[1] += 1

#####

#####

    if ref_bbox.size == 0 :

        temp = np.concatenate((np.array(int(track.track_id), dtype = np.int32), bbox),
axis = None).reshape(1, -1)

        # temp = int(temp)

        ref_bbox = np.append(ref_bbox, temp.astype(int)).reshape(-1, 5)

        # print('Ref_BBox size was zero. ', ref_bbox, ref_bbox.shape)

    elif int(track.track_id) not in ref_bbox[:, 0]:

        temp = np.concatenate((np.array(int(track.track_id), dtype = np.int32), bbox),
axis = None).reshape(1, -1)

        ref_bbox = np.concatenate((ref_bbox, temp.astype(int)), axis = 0)

        # print('Ref_BBox size was greater. ', ref_bbox, ref_bbox.shape,
type(ref_bbox[0,0]))

    ind = np.where(int(track.track_id) == ref_bbox[:, 0])[0]

    ind = int(ind)

```

```

pix_diff = abs(bbox[1] - ref_bbox[ind, 2]) + abs(bbox[3] - ref_bbox[ind, 4])

# dist = (pix_diff * mpp) / 1000

# time_hours = 1 / (60 * 60)

# speed = dist/time_hours

dist = pix_diff * mpp

time_sec = 1 / vid_fps

speed = dist / time_sec

# Draw Box on Screen and print("Speed for tracking ID {} is {}
KPH.".format(str(int(track.track_id)), str(speed)))

color = colors[int(track.track_id) % len(colors)]

color = [i * 255 for i in color]

cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])),
color, 2)

cv2.putText(frame, "{:.1f}".format(speed), (int(bbox[0]), int(bbox[1] - 10)), 0,
0.6, (255,255,255), 2)

# Updating the bounding box co-ordinates

ref_bbox[ind, 1 : ] = bbox

```

```
#####
```

```
#####
```

```
# if enable info flag then print details about each track

if FLAGS.info:

    print("Tracker ID: {}, Class: {}, BBox Coords (xmin, ymin, xmax, ymax):
    {} Pixel Difference: {}".format(str(track.track_id), class_name, (int(bbox[0]),
int(bbox[1]), int(bbox[2]), int(bbox[3])), str(pix_diff)))

    if (car[0] + truck[0] + bus[0] + motorbike[0]) < 5:

        congestion[0] = "Low"

    elif (car[0] + truck[0] + bus[0] + motorbike[0]) >= 5 and (car[0] + truck[0] +
bus[0] + motorbike[0]) < 10:

        congestion[0] = "Medium"

    else:

        congestion[0] = "High"

    if (car[1] + truck[1] + bus[1] + motorbike[1]) < 5:

        congestion[1] = "Low"

    elif (car[1] + truck[1] + bus[1] + motorbike[1]) >= 5 and (car[1] + truck[1] +
bus[1] + motorbike[1]) < 10:
```

```
congestion[1] = "Medium"
```

```
else:
```

```
congestion[1] = "High"
```

```
cv2.putText(frame, str(car[0]), (270, 60),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(truck[0]), (270, 90),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(bus[0]), (270, 120),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(motorbike[0]), (270, 150),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(congestion[0]), (270, 180),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(car[1]), (520, 60),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(truck[1]), (520, 90),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(bus[1]), (520, 120),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(motorbike[1]), (520, 150),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
cv2.putText(frame, str(congestion[1]), (520, 180),  
cv2.FONT_HERSHEY_COMPLEX_SMALL, 1.5, (0, 255, 0), 2)
```

```
# calculate frames per second of running detections
```

```
fps = 1.0 / (time.time() - start_time)
```

```
print("FPS: %.2f" % fps)
```

```
result = np.asarray(frame)
```

```
# result = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

```
if not FLAGS.dont_show:
```

```
    cv2.imshow("Output Video", result)
```

```
# if output flag is set, save video file
```

```
if FLAGS.output:
```

```
    out.write(result)
```

```
if cv2.waitKey(1) & 0xFF == ord('q'): break
```

```
cv2.destroyAllWindows()
```

```
if __name__ == '__main__':
```

```
    try:
```



```
app.run(main)
```

```
except SystemExit:
```

```
pass
```