AUTOMATED TRAFFIC COUNTING DATA COLLECTION AND ANALYSIS

ANAND LOW HONG REN

A project report submitted in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology (Hons) Electronic Systems

Faculty of Engineering and Green Technology
Universiti Tunku Abdul Rahman

May 2021

DECLARATION

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

Signature	:	[illegible]
Name	:	ANAND LOW HONG REN
ID No.	:	18AGB01371
Date	:	3rd SEPTEMBER 2021

APPROVAL FOR SUBMISSION

I certify that this project report entitled **"AUTOMATED TRAFFIC COUNTING DATA COLLECTION AND ANALYSIS"** was prepared by **ANAND LOW HONG REN** has met the required standard for submission in partial fulfilment of the requirements for the award of Bachelor of Technology (Hons) Electronic Systems at Universiti Tunku Abdul Rahman.

Approved by,

Signature	:	[illegible]
Supervisor	:	Dr. Lee Han Kee
Date	:	25/4/2022

Specially dedicated to

my beloved grandmother, grandfather, brother, mother and all my friends.

ACKNOWLEDGEMENTS

I would like to thank everyone who had contributed to the successful completion of this project. I would like to express my gratitude to my research supervisor, Dr. Lee Han Kee for his invaluable advice, guidance and his enormous patience throughout the development of the research.

In addition, I would also like to express my gratitude to my loving parent and friends who had helped and given me encouragement to face every challenge and come out better and wiser than before.

AUTOMATED TRAFFIC COUNTING DATA COLLECTION AND ANALYSIS

ABSTRACT

The increase in the number of vehicles purchased over the years cause a high volume of vehicles on the road. This leads to traffic congestion especially in urban areas. This problem disrupts the daily life of many people. It is important to conduct traffic analysis and surveys to extract traffic information which would be useful for solving and evaluating the quality of transportation. Optimal traffic arrangements that reduce traffic congestion can be designed by engineers using the collected traffic data. Traffic data collection is also useful for other issues such as vehicle accidents, managing parking areas, speeding, vehicle theft detection and others. There have been many methods of traffic data collection proposed and implemented over the years, each with their own pros and cons. This project proposed an automated traffic counting data collection and analysis algorithm that is able to use computer vision to count and measure the speed of vehicles, while also able to classify vehicles into different categories using the power of deep learning and AI. The performance of the algorithm is determined by the counting, classification, and speed measuring accuracy. The factors affecting the performance of the algorithm is discussed. The system is able to performance the tasks when it is in the bright condition with the accuracy of more than 95%. However, the accuracy is dropped to 50% when the condition is dark. This is due to the system is unable to detect the vehicle in such condition.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS / ABBREVIATIONS

RADAR	Radio Detection and Ranging
LIDAR	Light Detection and Ranging
IR	Infrared
VCR	videocassette recorder
AI	Artificial Intelligence
CNN	Convolution Neural Network
CT	Computerized Tomography
MRI	Magnetic Resonance Imaging
CV	Computer Vision
MOG	Mixture of Gaussians
YOLO	You Only Look Once
VNC	Virtual Network Computing
fps	frames per second

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background

Traffic congestion especially in urban areas is an avoidable problem in Malaysia. This problem is especially evident daily during the morning when people are going to work, and in the evening, when people are coming home from work. This problem is especially serious during the festive seasons in Malaysia such as Chinese New Year, Hari Raya Aidil Fitri, Thaipusam and many others. Other than that, traffic congestions also happen during unprecedented phenomena such as floods, accidents and others.

Malaysia as a developing country going through urbanization and economic modernization activities. With development in different aspects such as economic, physical, social, politics, comes the increased demand of transportation for work, leisure, and other purposes. According to a market research agency called Nielsen (2014), it is found in 2014 that Malaysia has the third highest rate of private car ownership in the world, with 93 percent of households owning a car. This shows the heavy dependency of Malaysians on private vehicles for transport for daily activities, which results in a high number of vehicles on the road.

Traffic congestion happens when there are too many cars on the road and the traffic flow is disrupted. Traffic congestion creates havoc on the road and disrupts drivers' everyday routines. Time spent on the road has a different kind of negative consequences for productivity, social behaviour, the environment and the economy.

Therefore, it is important to conduct traffic analysis and surveys to extract traffic information which would be useful for solving and evaluating the quality of transportation. With this traffic information, engineers are able to analyse the traffic situation and be able to solve and design an optimal traffic arrangement which minimizes traffic congestions. Traffic data collection and traffic surveys are designed to collect statistics that properly represent the area's real-world traffic condition. For example, traffic surveys may count the number of cars on a road or gather data on travel times, travel frequency, origin/destination for the trips and others.

Other than that, vehicle counting, and traffic data collection can be useful for other uses such as vehicle accidents, managing parking areas, vehicle theft detection and others.

Traffic counting and surveys are generally categorized into two different types, manual counting and automatic counting. There are two ways of manual counting. Firstly, an operator can stand by the side of the road and record the number of vehicles seen and category of car in a paper pad. Another way of manual counting is a camera is placed at an eagle’s eye point of view over the road, and a video of the road will be recorded. After that, an operator would review the video and count the vehicles passing by analysing the video. For automatic counting, it includes various ways such as radar, piezoelectric sensors (e.g., RADAR vehicle counter Sierzega SR4), pneumatic road tubes, and induction loops. Recently, machine learning and AI has started to become a hot topic in the technology industry, and one of the real-world problems people have been trying to solve using AI and machine learning is traffic counting.

1.2 Problem Statements

For many years, traffic counting has been conducted in various ways, such as manual counting and automatic counting.

For manual counting, an operator has to be at the side of the road, constantly observing vehicles passing by the road, then write down the vehicle category and the time of the vehicle passing the road. Another way of manual counting is by recording the road that require traffic flow analysis, and then the operator will review that video and count the vehicles. The operator can use tally sheets or mechanical counters to aid their counting of vehicles. However, this method is not very effective because the quality or accuracy of the manual counts are affected by human error and visibility of the vehicles on road.

For automatic counting, there a several methods such as piezo, pressure sensors, inductive and magneto-metric sensors, acoustic sensors, Lidar and Radar, photo/video and IR sensors (Yatskiv, et al., 2013). These sensors have been used to replace manual counting, however each of the method has its advantages and disadvantages. For example, pressure, piezoelectric, inductive sensors have to be installed into the road surface which is cumbersome and requires special permission from the road owner. RADAR and LIDAR sensors have a major disadvantage where they are only able to detect the speed of one moving vehicle at a time. Acoustic sensors are prone to disruption by bad weather conditions and probabilistic noise according to Yatskiv, et al. (2013). It also does not measure several vehicles simultaneously.

All traffic collection data methods reflect a trade-off between the method’s objectives, available resources, possible coverage, and the quantity of data to gather. Depending on the objectives, the associated expenses, and the required level of quality, one of the available traffic data collecting methods may be chosen. Overall, the best method so far has been using video and IR sensors to collect traffic data, because this method is relatively inexpensive compared to others due to low-cost electronic devices. By combining the overall best sensor to collect traffic data with the power of computer vision machine learning, an evolved and better version of traffic data collection method can be created.

1.3 Aims and Objectives

The objectives of this thesis are shown below:

i. To construct a low-cost hardware system that is able to capture good quality video recording of vehicles on the road

ii. To determine the best computer vision algorithm to be implemented in the traffic data collection system.

iii. To construct a software program that is able to detect, track and count the vehicles in the video.

iv. To construct a software program that is able to calculate the speed of every vehicle passing in the video

v. To construct a software program that is able to classify the vehicles in the video.

CHAPTER 2

LITERATURE REVIEW

2.1 Traffic Data Collection Systems

2.1.1 Manual Traffic Data Collection

Manual traffic data collection refers to the collection of traffic data through manual ways, which requires a human operator to examine vehicles on the road. Some examples of manual traffic data collection include counting vehicles at junctions, estimating average daily traffic, and calculating yearly average daily traffic (Adebisi, 1987; Baker et al., 1982; Findley et al., 2011). Manual traffic data collection is generally done in two ways, on site (inspecting vehicles directly beside the road), or inspection through video recordings. The data is usually recorded by operators using tally sheets or mechanical counters (Zheng & Mike, 2012). The data are recorded for a time interval (e.g., 10 min), then the total is calculated and can be saved to the computer for later processing (Schumann, 2001; Wylie, 2010; Jalihal, et al., 2005). Manual counting is generally not able to capture complicated traffic patterns, and advance data like travel time. Zheng and Mike (2012) investigated the accuracy of manual traffic data collection by examining a video recording of the traffic in Southampton taken by Transportation Research Group (TRG) of University of Southampton. In the authors' research, they used a computer-controlled VCR which can modify the playback speed of the video recording, ranging from 1 fps to 9 times the normal playback speed. The operator would press certain keys (L for long vehicle, S for short vehicle), then the time stamps and the vehicle category will be saved in a text file. The result obtained from this method is highly accurate, but this method is more time consuming. The authors took the result and compared to the manual counting done by students from TRG of University of Southampton on site. The research found that counting errors (counting number of vehicles on the road) is generally less than 1%, and classification errors (classifying the vehicle into two categories: long (more or equal than 5.2m in length) and short (less than 5.2m in length)) are more significant, at an average of 4% to 5%. This result is due to the difficulties of judging the vehicles' length through the video recordings.

The price for manual traffic collection is dependent on a few factors, such as number of traffic movements, number of vehicle classes or categories, and traffic density. In general, manual traffic collection provides a cheaper hourly rate for simple traffic scenes such as highways, but it gets much more expensive when locations with high volume of traffic and movements are required to be analysed (Stofan, 2020).

Overall, manual traffic data collection is slow, prone to human errors, and it is only providing static traffic volume reports in Excel spreadsheets at best (Stofan 2020).

2.1.2 Automatic Traffic Data Collection

Automatic traffic data collection refers to all methods of traffic data collection that does not require a human operator to examine and manually count the vehicles on the road. Automatic traffic data collection includes LIDARs, RADARs, inductive loop sensors, acoustic sensors, pressure (piezoelectric) sensors and others. Different sensor technology comes with its own advantages and disadvantages.

LIDAR (Light Detection and Ranging) sensors uses active optical systems that project light in the form of a pulsed laser onto the vehicle, and the laser will be reflected back into the system for information processing. The accuracy of the LIDAR sensor is highly dependent on the weather conditions (snow, rain and dust) according to Yatskiv et al. (2013). RADAR sensors are similar but uses radio waves instead of pulsed lasers to determine the distance, velocity of the vehicles. Both LIDAR and RADAR sensors have a major disadvantage which is they are only able to detect the velocity of a single moving vehicle at a time. It is usually used for traffic enforcement camera to monitor compliance with speed limits by the police.

Inductive loop sensors are a wire placed under the road surface and connected to a controller. When a vehicle passes the induction loop or stops on it, the vehicle's ferrous body material increases the loop's inductance, but the peripheral metal of the vehicle decreases inductance because of eddy currents produced. The decrease in inductance due to eddy currents offsets the increase in inductance due to ferrous metal, thus, an overall decrease in inductance is observed. Decrease in inductance results in the decrease of impedance in the loop, which will actuate the electronics unit output relay, which sends a signal to the controller to indicate the passing or presence of a vehicle. Induction looks can classify the type of vehicle depending on its iron mass, while it may also detect the velocity of the vehicle with an accuracy of ±1%. The main disadvantage is that it requires the destruction of the road surface to install the inductive loops.

Pressure such as piezo electric sensors consist of two switch elements, which are both apart with a certain distance. The first switch element is triggered when the front wheel of the vehicle crosses it, which signals the timer to start. When the back wheel crosses the second element, it stops the timer. This sensor is able to record the velocity, weight and direction of the vehicles passing by (Batenko et al. 2011). The accuracy of the velocity measurement is less than 1%. The sensor also has to be installed into the road surface, and the measurements can be disrupted by weather conditions, such as temperature and ice.

Table 2. 1 Comparisons between different traffic counting methods (Yatskiv, et al., 2013)

Technology	Vehicle Counter	Velocity measurement Accuracy (%)	Real time data	Additional data	Affected by weather conditions
Manual human counters	Yes	No	No	Vehicle Type	Yes
Pressure (piezoelectric) sensors	Yes	Yes, <1%	Yes	Weight, Vehicle Type	No
Inductive loop sensors	Yes	Yes, <1%	Yes	Vehicle Type	No
Ultrasonic sensors	Yes	No	Yes	Vehicle Type	Yes
RADAR and LIDAR sensors	Yes	Yes, < 3km/h	Yes	Vehicle Type	No

2.2 Computer Vision

Computer vision is a field of artificial intelligence (AI) that allows computers and other systems to derive useful information for the end user from digital images, videos and other visual inputs. Computer vision can be said to be the eyes for a machine system, will the AI algorithm is the brain. Computer vision trains the computer to identify and distinguish objects, how far away the object is, whether the object is moving, or if something is wrong. Computer vision requires a lot of data, much like machine learning. It requires repeated analysis of the different and unique datasets until it is able to recognise the unique features of each object and finally identify the object in the photo or video.

Currently, the most used technologies for computer vision is deep learning (a type of machine learning) and a convolution neural network (CNN). Machine learning uses different statistical algorithmic models to allow the computer to “learn” the patterns and unique features of an object from visual data. These algorithms enable the machine to run through the data and learn the features of the objects by itself, without the need for programmers to specifically program it to recognize the object (IBM, 2020).

There are a few tasks that can successfully be done by computer vision, which includes image classification, object detection, object tracking and content-based image retrieval.

Image classification means the system is able to analyse an image and is able to classify the image based on the subject in the photo (human, animals, vehicles). The system capable of correctly predicting which class the image belongs to. This is especially useful in many cases where classifying the images into different groups is crucial. For example, a group of images of different animals can be classified into groups based on their species, such as dog, cat, mouse and others.

Object detection refers to the system able to identify the object inside the image by finding similarities of the object with the available datasets. If enough similarities are found, the system is able to correctly identify the object in different images or videos. For example, quality inspection machines can use computer vision systems to identify defects or scratches on the products the system is inspecting, which is very useful for quality control.

Object tracking refers to the computer vision system able to track the object after detecting it. This task requires a sequence of pictures or video streams as input data. For example, autonomous vehicles have to use object tracking to track people, other vehicles, road infrastructure and other things in motion to prevent accidents and comply with traffic regulations (Le, 2018).

2.3 Digital Image Processing

Digital Image Processing is the "manipulation of images using digital computers according to Eduardo and Gelson (2005). If human vision is important to us humans, then digital image processing is one significant part of computer vision that allows the image to transform into data that is significant. It's use is undoubtedly increasing exponentially over the years, following the rise of Artificial Intelligence and Machine Learning. It's applications range from the medical sector, such as the 3D modelling in CT and MRI, to entertainment in the form of the Nintendo Wii remote which is able to use computer vision and image processing to track the player's movements. Multimedia systems such as televisions and displays, rely heavily on digital image processing.

The discipline of digital image processing is vast, which includes various different algorithms and techniques to manipulate images for different purposes. An image can be regarded as a function $f(x,y)$ of two continuous variables x and y. A digital image consists of a matrix of numbers representing every single pixel in the image. Digital image processing consists of the manipulation of those finite precision numbers.

Digital image processing can be divided into several classes: image enhancement, image restoration, image analysis, and image compression. In image enhancement, images are modified through heuristic methods, which allows useful data to be extracted from the images. Image restoration techniques is used for processing bad or corrupted images to remove degradation so that the clarity or features of the original image can be restored. Image analysis techniques are used to process the image in such a way that information can be automatically extracted from it and used for other purposes, such as quality inspection of a product. Examples of image analysis are image segmentation, edge extraction, and texture and motion analysis (Eduardo & Gelson, 2005). A digital image can contain huge amounts of information depending on it's resolution, format and encoding. For example, a gray-scale image of moderate resolution, say 512×512, needs $512 \times 512 \times 8 \approx 2 \times 106$ bits for its representation. Therefore, image compression is essential as a way to reduce the file size of an image for better storage and sharing of digital images.

2.3.1 Background Subtraction

The task of marking foreground entities or objects plays an important role in the video pre-processing process as the initial phase of computer vision applications (Murzova, 2021). These applications include vehicle detection, people tracking, animal tracking and others that require tracking, monitoring, recognition of objects. Background subtraction allows CV programs to obtain rough but rapid identifications (outlines) of objects that appear in the video stream (Murzowa, 2021). Figure 2.1 shows the input and output of passing an image through the background subtraction algorithm while Figure 2.2 shows the processes involved in background subtraction.

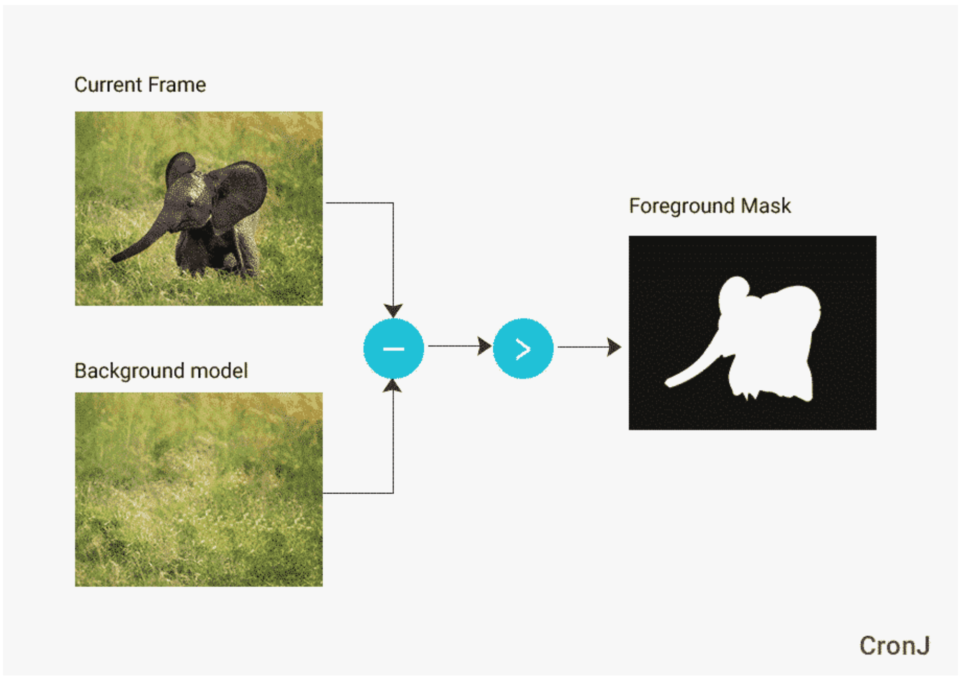


Figure 2. 1 Input and Output of Background Subtraction

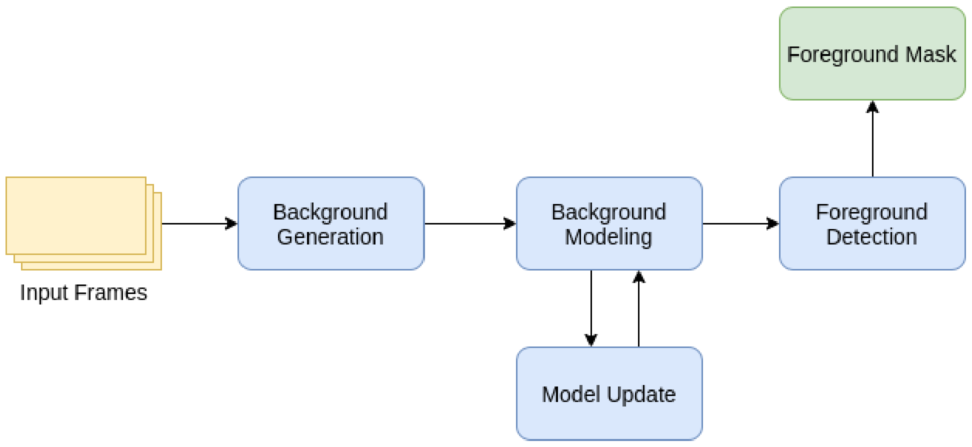


Figure 2. 2 Basic Background Subtraction process

Background subtraction methods creates a background model to separate foreground from the background. The background subtraction process contains these phases:

i. Background Generation: Processes multiple frames from video to obtain the background image

ii. Background Modelling: Defines the model for background representation

iii. Background Model Update: Any changes that occurs in the video will be processed here to update the Background Model

iv. Foreground Detection: Divides the pixel into two groups, background, or foreground.

The background subtraction algorithm then outputs an image which is a binary mask, with the foreground objects in white pixels while the background objects in black pixels.

2.3.1.1 Descriptors

One important concept in background subtraction is descriptors, which is also known as features. Descriptors define the regions in an image that are marked according to the background model. This comparison allows the categorisation of region into the background or the foreground. Descriptors can be marked based on colour, texture or even edges. Popular pixel domain descriptors:

i. Colour: Colour features are sensitive to illumination, shadows and anything which affects the appearance of moving objects. Therefore, algorithms usually combine this with other descriptors to achieve a more robust background subtraction model.

ii. Edge: Edge features are great because they are unaffected to light variations and great for moving object detection. However, they are sensitive to both high and low textured objects.

iii. Texture: Texture features provide spatial information. Texture descriptors are unaffected by different illumination and shadows.

2.3.1.2 Background Modelling Algorithms

The three available background modelling algorithms present in OpenCV are GMG, MOG and MOG2.

The GMG algorithm is proposed by Godbehere, Matsukawa, and Goldberg in 2012. GMG models the background with a combination of Bayesian Inference and Kalman Filters. Bayesian inference is a method in which the Bayes’ theorem is used to update the probability for a hypothesis as more information is obtained. This means newer observations are given more weight than older observations to compensate for variable or changing illumination. The algorithm consists of two stages. In the first stage, the method accumulates for each pixel, weighted values based on how long a colour (pixel RGB value) stays at that position. When new frames are inserted to the algorithm, new observations are added into the model, thus effectively updating these weighted values. Colours that stay static for some amounts of time are considered background by the model. The second stage filters pixels in the foreground to reduce noise (Marcomini & Cunha, 2018).

Mixture of Gaussians, also known as MOG, was first proposed by KaewTraKulPong and Bowden in 2002. A combination of k Gaussian distributions models each background pixel using this technique, with k values between 3 and 5. The authors make the assumption that distinct distributions correspond to distinct background and foreground hues. Each of the distributions utilised in the model has a weight proportionate to the amount of time each colour spends on that pixel. Therefore, when a pixel's weight distribution is small, it is classed as foreground. The MOG2 technique was developed to address one of MOG's limitations: the fixed number of used distributions. MOG2 provides a more accurate depiction of the complexity of colours in each frame by using a configurable number of Gaussians distributions that are mapped pixel by pixel.

2.4 Machine Learning

Machine learning is an implementation of artificial intelligence (AI) that gives systems the ability to automatically learn and develop from experience without being specifically programmed. Machine learning focuses on the creation of computer systems that can access data and use it to learn about themselves. Traditional programming methods rely on hardcoded rules, which step-by-step set out how to solve a problem. In comparison, a task is set for machine learning programmes, and a vast volume of data is provided to use as examples of how this task can be performed or from which patterns can be found. The machine then discovers how the intended output will better be obtained. It can be viewed as narrow AI: provided a particular collection of data to learn from, machine learning helps smart systems that are able to learn a specific purpose.

While still not approaching the human-level knowledge that is typically synonymous with the term AI, opposed to conventional programming approaches, the ability to learn from data increases the amount and complexity of tasks that machine learning systems can perform. Machine learning can execute complex functions such that the desired outputs cannot be generated based on human programmed step-by-step procedures. The learning dimension also produces applications that can be flexible and increase the accuracy of their outcomes once they are implemented (Markoff, 2015).

The three key methods of machine learning are supervised, unsupervised and reinforcement learning.

Supervised learning is by using labelled datasets to train algorithms that either classify the data or predict outcomes based on the learning the patterns in the data. When the input data is inserted into the machine learning model, the weights in the machine learning algorithm will adjust until the model has fitted appropriately. This process ensures that the machine learning model does not overfit or underfit to the data.

Unsupervised learning is different from supervised learning in the datasets used. For unsupervised learning, the datasets are not labelled, thus the algorithms have to analyse and cluster the data based on similarities on certain features or discover hidden patterns or groupings within the data without any supervision from humans. It’s able to discover similarities and differences which makes it great for data analysis, customer segmentation, image and pattern recognition. It can also be used to reduce dimensions in datasets for better results in the machine learning model. Algorithms used in unsupervised learning include principal component analysis (PCA) and singular value decomposition (SVD), neural networks, k-means clustering, probabilistic clustering methods and others (IBM, 2020).

Reinforcement machine learning is similar to supervised learning, but the algorithm is not trained using sample data. Instead, the model learns through trial and error. A series of successful results will be reinforced in order to create the most appropriate solution to a particular problem (IBM, 2020).

2.5 Deep Learning

Deep Learning is a subfield of machine learning and artificial intelligence concerned with algorithms inspired by the structure and function of the brain called artificial neural networks. These artificial neural networks are algorithms inspired by the human brain and learn from large amounts of data. Just like how humans learn from past experiences, deep learning algorithms would perform a task repeatedly, each time tweaking parameters in the algorithm a little to improve the outcome. These algorithms are referred as deep learning because the neural networks have various (deep) layers that enable learning. Any problem that requires thinking to solve is a problem that deep learning can learn to solve. However, to successfully create a deep learning neural network which produces good results, huge amounts of data have to be collected.

The amount of data humans generates every day is astronomical - currently estimated at 2.6 quintillion bytes- and it's the resource that makes deep learning possible. Since deep learning heavily relies on a ton of data to learn from, the increases in data creation in recent years is one of the reasons why deep learning capabilities have grown considerably. In addition to increase in data creation and collection, stronger computing power and also the growth of Artificial Intelligence as a Service benefitted deep learning too (Marr, 2018).

In general, Deep learning allows computers to solve complex problems even when using a data set that is very diverse, unstructured and interconnected. The more deep learning algorithms learn, the better they perform. Figure 2.3 compares the performance and amount of training data needed for deep learning to other learning algorithms.

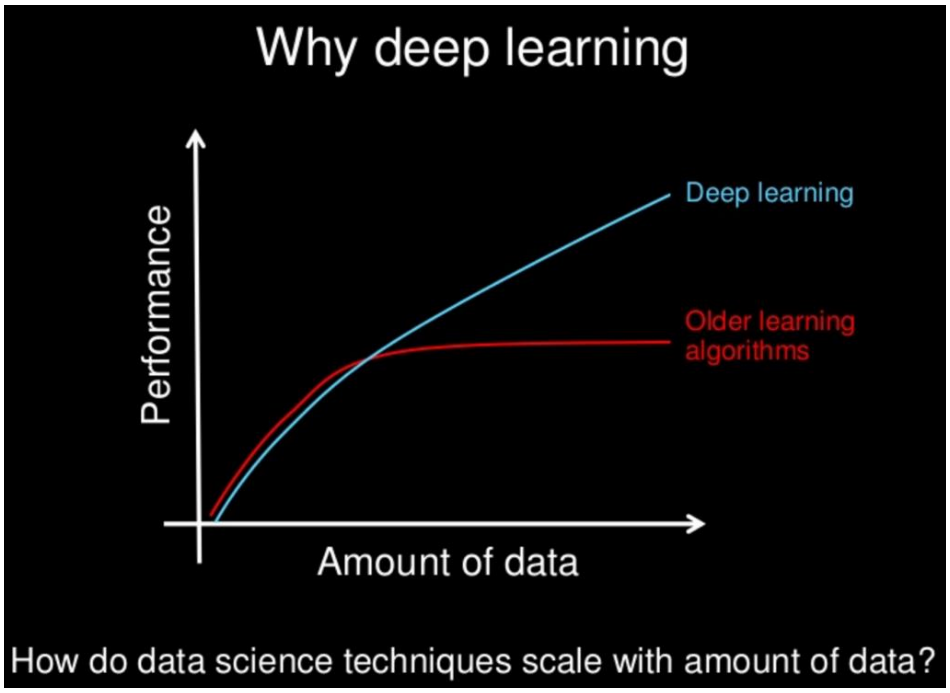


Figure 2. 3: Deep Learning vs other older machine learning algorithms

2.5.1 YOLOv3

YOLOv3 (You Only Look Once, Version 3) is a real-time object recognition algorithm that recognises particular items in videos, live streams, or images. YOLO detects objects using characteristics learnt by a deep convolutional neural network. Joseph Redmon and Ali Farhadi developed YOLO versions 1-3.YOLO's initial version was released in 2016, while the third version, was released two years later in 2018. YOLOv3 is a variant of YOLO and YOLOv2 that has been enhanced. YOLO is implemented using the deep learning packages Keras or OpenCV (Meel, 2021). Figure 2.4 shows the processes involved in YOLO object detector.

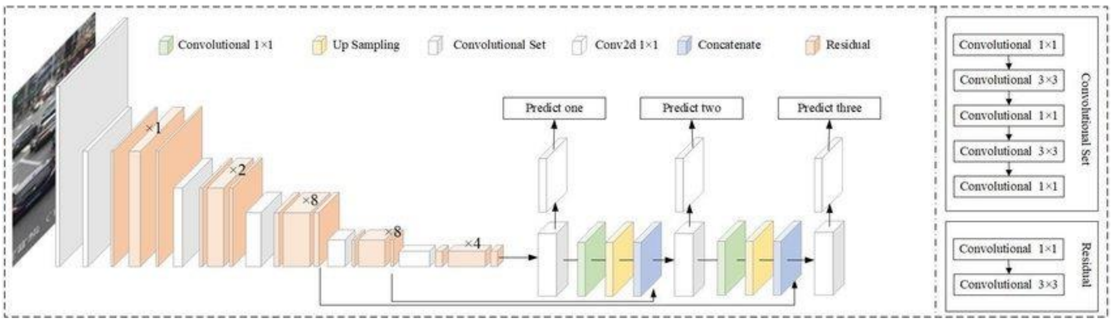


Figure 2. 4: Overview of YOLO object detector

As is customary for object detectors, the convolutional layers' learnt features are given to a classifier, which makes the detection prediction. The prediction in YOLO is built on a convolutional layer with 1×1 convolutions. YOLO stands for "you only look once" due to the fact that its prediction utilises 1×1 convolutions. the prediction map size is same to the feature map size before it.

YOLO is a Convolutional Neural Network (CNN) that is capable of real-time object identification. CNNs are classifier-based systems capable of processing incoming pictures as organised arrays of data and finding correlations between them. YOLO has the benefit of being much quicker than other networks while maintaining the same level of accuracy. It enables the model to consider the whole picture at test time, ensuring that its predictions are influenced by the image's global context. YOLO and other convolutional neural network methods provide a numerical value to areas based on their similarity to preset classifications. High-scoring areas are labelled as positive detections of the class to which they most closely correspond. For instance, in a live traffic stream, YOLO may be used to distinguish between various types of cars based on which parts of the video score well in contrast to preset vehicle classifications (Meel, 2021).

2.5.2 Faster R-CNN

Faster R-CNN was first published in 2015 at NIPS. Following publication, it underwent many modifications. Faster R-CNN is the third iteration of the R-CNN papers, which were co-authored by Ross Girshick. Everything began with the 2014 publication of "Rich feature hierarchies for accurate object detection and semantic segmentation" (R-CNN), which utilised a technique called Selective Search to suggest areas of interest and a conventional Convolutional Neural Network (CNN) to categorise and modify them (Rey, 2018). It rapidly developed into Fast R-CNN, released in early 2015, where a method called Region of Interest Pooling enabled the model to run considerably faster by pooling costly calculations. Finally, there was

Faster R-CNN, which introduced the first completely differentiable model. Figure 2.5 shows the process involved in Faster R-CNN.

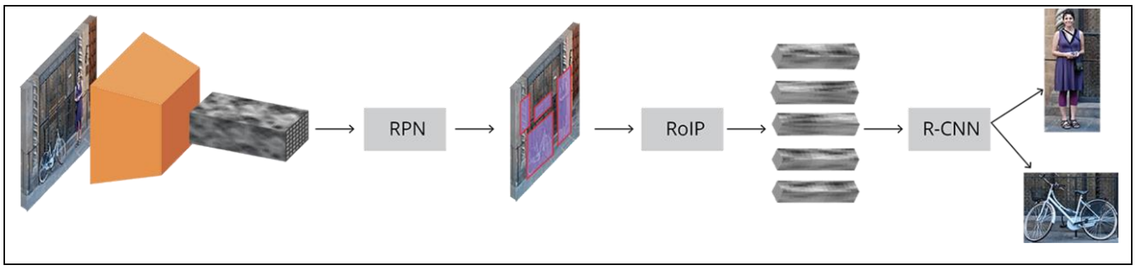


Figure 2. 5: Overview of Faster R-CNN

Faster R-design CNN's is complex due to the presence of many moving components. It all begins with an image, the following information will be extracted: a list of bounding boxes, a label for each bounding box, and a probability for each label and bounding box.

The input images are expressed as *Height* × *Width* × *Depth* tensors (multidimensional arrays), which are processed through a pre-trained CNN until reaching an intermediate layer, where they are transformed into a convolutional feature map. This is used as a feature extractor in the next section. Following that, is what is known as a Region Proposal Network (RPN, for short). It is used to discover up to a specified number of regions (bounding boxes) that may contain objects using the characteristics calculated by the CNN. The most challenging aspect of utilising Deep Learning (DL) for object identification is probably creating a variable-length list of bounding boxes. When deep neural networks are modelled, the final block is often a fixed-sized tensor output. For instance, in image classification, the output is a *(N,)* shaped tensor, where *N* is the number of classes, and each scalar at location I denotes the probability of the image being label_i (Rey 2018).

The RPN solves the variable-length issue by using anchors: fixed-size reference bounding boxes that are evenly distributed across the source picture. Rather of attempting to determine the location of objects, the issue was divided into two components. Each anchor is examined to see whether it contains a relevant item and how the anchor could be modified to better suit the relevant object. Once a list of potentially relevant items and their positions in the original picture is created, the issue becomes simpler to solve. Region of Interest (RoI) Pooling can be used to the CNN features and the bounding boxes of relevant items to extract the features that relate to the relevant objects into a new tensor. Finally, there is the R-CNN module, which utilises this information to: Classify the material included inside the bounding box (or reject it, using the label "background"), and update the bounding box coordinates to better fit the objects (Rey 2018).

2.5.3 TensorFlow

TensorFlow is an open-source machine learning platform that runs from start to finish. It features a large, flexible ecosystem of tools, libraries, and community resources that allow researchers to advance the state-of-the-art in machine learning and developers to quickly construct and deploy ML applications (TensorFlow, 2022).

Google research scientists and software developers often construct cutting-edge models and make them publicly accessible rather than keeping them private in order to improve the research community's capabilities. The COCO detection challenge, which focuses on recognising objects in pictures (estimating the possibility that an item is at this position) and their bounding boxes, was won by Google's in-house object detection system in October 2016. The Google approach has not only been featured in a number of publications and used in a number of Google products (Nest Cam, Image Search, and Street View), but it has also been made available to the general public as an open-source framework based on TensorFlow.

The TensorFlow object detection API provides a framework for building a deep learning network that can identify objects. The framework includes various important features, as well as five pre-trained models: SSD (Single Shot Detector) with MobileNets, Region-Based Fully Convolutional Networks (R-FCN), and Faster R-CNN, as well as five pre-trained models. The quickest model is SSD, followed by R-FCN, and lastly the faster R-CNN. The model's precision, on the other hand, is the inverse.

The regional proposal methods are used in all of these models. To construct a provisional enumeration of feasible bounding boxes in a picture, such algorithms employ image segmentation (that is, partitioning the image into regions based on the primary colour differences within areas themselves). The idea is that region proposal algorithms propose a small number of boxes to examine, far fewer than an exhaustive sliding windows method would propose. This enabled them to be used in the initial R-CNNs, or region-based convolutional neural networks, which operated by using a region proposal technique to locate a few hundreds or thousands of areas of interest in an image. Each zone of interest is processed by a CNN to build features for each area, which are then used to categorise the region using a support vector machine and a linear regression to calculate more exact bounding boxes (Luca Massaron, 2018).

Fast R-CNN used CNN to analyse the whole picture at once, transform it, then apply the region suggestion to the transformation. This reduced the number of calls processed by CNN from a few thousand to just one. Another aspect that made it quicker was that it employed a soft-max layer and a linear classifier instead of an SVM for classification, thereby extending the CNN rather than moving the input to a new model (Luca Massaron, 2018).

R-FCN, on the other hand, are quicker than Faster R-CNN since they are fully convolutional networks with no fully connected layers following the convolutional layers. They're end-to-end networks, meaning they go from convolutional input to output. They become even quicker as a result of this (they have a much lesser number of weights than CNN with a fully connect layer at their end). However, their speed comes at a cost; they are no longer defined by image invariance (CNN can figure out the class of an object, no matter how the object is rotated). A position-sensitive score map, which is a means to assess whether sections of the original image processed by the FCN correspond to parts of the class to be categorised, is supplemented by a faster R-CNN (Luca Massaron, 2018).

Finally, we have solid-state drives (SSD) (Single Shot Detector). Because the network anticipates the bounding box position and class as it analyses the picture, the performance is significantly faster here. By bypassing the region proposal step, SSD can calculate a large number of bounding boxes quickly. It only decreases heavily overlapping boxes, but it still processes the most bounding boxes of any of the models we've discussed so far. Its speed is due to the fact that it classifies each bounding box as it delimits it: by performing everything in one shot, it has the quickest speed, yet it performs comparably (Luca Massaron, 2018).

2.5.4 Performance metrics for deep learning

The more popular performance metrics for deep learning is mAP (mean average precision) and average recall. However, to understand these two metrics, the concept of precision and recall must be established first.

Precision measures the accuracy of the predictions. Recall measures how well the model find these positives. The value of precision and recall is always between 0 and 1. The calculation of precision and recall is shown in equation 2.1 and 2.2.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Eq. (2.1)

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Eq. (2.2)

The precision of every prediction is calculated and averaged to get the mAP of the model. The recall calculated for every prediction and averaged which results in the average recall. To build a good object detection model the value should be as close to 1 as possible for mAP and average recall.

2.6 Journal Reviews

Marcomini and Cunha (2018) compared the performance between three different background modelling methods by using vehicle segmentation in highway traffic videos. The authors analysed seven videos of highway traffic videos, with a total video time of 2 hours. The authors measured how well each algorithm performs in detection and segmentation based on accuracy rate, processing time and precision rate. The three background modelling methods are GMG, Mixture of Gaussians (MOG) and Mixture of Gaussians 2 (MOG2). All three methods have comparable precision rate at above 90%, however MOG and MOG2 have precision rates around 100%, while GMG has a precision rate between 60% and 80%. Comparing processing times, MOG2 is on average 3 times faster than MOG, while being 10 times faster than GMG. In conclusion, MOG2 is the best performer among the three background subtraction methods (Marcomini & Cunha, 2018).

Li et al. (2014) proposed two different methods of real-time moving vehicle detection, tracking and counting system. One uses the pixel changes in histogram when a vehicle passes to count the vehicle, while the second method uses adaptive background subtraction in tandem with blob tracking. The first method is only able to count vehicles based on their presence. The second method is able to detect and count the vehicle, t it is able to track the path of the moving vehicle. The authors test this model by inserting a video of cars travelling in one single direction on the highway. It is found that the first method achieves an accuracy rate of 96%, while the second method have a higher accuracy rate of 98.4%. The results show that background subtraction with blob tracking is a more robust and accurate way of counting vehicles.

Sorwar et al. (2017) proposed a real-time vehicle monitoring which is able to count the number of vehicles by using extended maxima and minima to detect cars. The authors crop the everything out of the video frame except the vehicle lane and uses extended minima and maxima to extract the area of cars, then uses the area of the vehicles as weights in a formula which estimates the number of cars in the frame. In broad daylight, the model achieves an accuracy rate of 91%, while during the night, the model is able to achieve an accuracy rate of 88%. The positive is this model is able to work in various different angles compared to other models. However, the model is affected by streetlights, roadside objects, which will decrease the accuracy of the model if longer duration of datasets are tested.

Tangkocharoen and Srisuphab (2017) uses Haar cascade classifier for their vehicle detection. This model uses AdaBoost in tandem with a cascade classifier to recognise the vehicles in frame. Haar feature-based cascade classifiers were first introduced by Paul Viola and Michael Jones in 2001, and it became an effective object detection method. The classifier is first trained with both positive and negative examples, which is a machine learning based approach (OpenCV 2016). This allows the classifier to construct a function that describes the relationship of the input (image) and output (binary mask). This training step requires a substantial amount of computational power, but once trained, the objects can be detected by the classifier rather instantaneously. The authors were able to detect vehicles under various lighting and illumination in images obtained from highway traffic videos in Bangkok. However, the author does not apply this model onto videos, and their model was only able to detect vehicles, but not count them.

Faruque et al. (2019) proposed a program which is able to classify vehicles into 6 different categories: bike, truck, car, van, bus and trailer. These six types of vehicles are used by the Federal Highway Association (FHWA) to categorize different types of vehicles. The authors used Faster R-CNN and YOLOv3 deep learning methods to classify vehicles in several videos. There were several challenges that reduces the accuracy of vehicle classification, such as changes in environmental conditions, the background of the object, environment illumination, blur, motion and video resolution. The training sets were created manually from different traffic videos to train the deep learning classifiers, which is time consuming. The authors found that YOLO has faster training and testing rate compared to Faster R-CNN. YOLOv3 also has a higher accuracy rate at around 96.78% to 99.76%, while Faster R-CNN has an accuracy rate of around 95.91% to 97.93%. Deep learning methods are feasible for vehicle classification, however the training a deep learning model requires large datasets, which means the large computational operations in terms of memory that requires some amount of time and a powerful GPU to compute. Figure 2.6 shows the six categories of vehicles classified by Faruque et al. (2019) in his research.

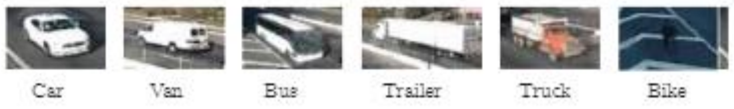


Figure 2. 6: Six categories of vehicles based on FHWA

CHAPTER 3

METHODOLOGY

3.1 Proposed System Design

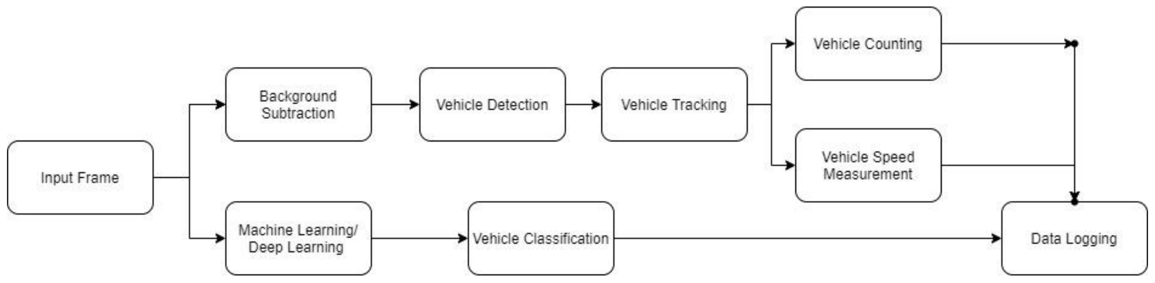


Figure 3. 1: Proposed System Design

The proposed system is able to complete three main functions, vehicle counting, vehicle speed measurement, and vehicle classification (shown in Figure 3.1). These important data from the vehicles will be stored in a file for further traffic data collection and analysis.

The Raspberry Pi and Raspberry Pi Cam V2 is setup at an outdoor position where the video of the traffic is recorded. This makes up the hardware side of the proposed system.

After that, the video is processed through software to extract useful traffic vehicle data from the video.

For vehicle counting and vehicle speed measurement, the system implemented background subtraction algorithm onto the input frames from the video which helps to separate and track moving objects from the background based on the particle filter algorithm. Tracked vehicles are counted based on a threshold that considers the area of the detected vehicles. The vehicle's speed is counted by using the equations 3.1, 3.2 and 3.3. The distance is calculated by finding the distance in pixels of two points P, dividing by c, which is constant which shows how many pixels per meter. To find c, the physical distance is measured between two points in the image and use Equation 3.2. To calculate time, the frame rate of video and frames travelled when car travels between two points in the video is required.

$$Speed = \frac{Distance}{Time}$$

Eq. (3.1)

$$Distance = \frac{|P_{end} - P_{start|}}{c}$$

Eq. (3.2)

$$Time\ = \frac{total\ frames\ travelled}{frame\ rate}$$

Eq. (3.3)

For vehicle classification, a machine learning classifier to be built and trained following the process in Figure 3.2.

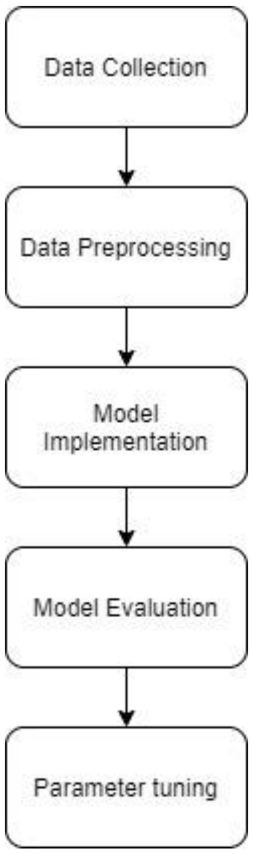


Figure 3. 2: Process of developing a deep learning classifier model

Figure 3.2 shows the whole process of developing a machine learning classifier model. There are 5 steps in this process which is: Data Collection, Data Pre-processing, Model Implementation, Model Evaluation and Parameter Tuning.

For the data collection, the data to train the deep learning classifier model have to be collected, and in this project's application, the images of different classes of vehicles from the side view required. A lot of images are required to create a large dataset which would be needed to train the deep learning classifier model. The data can be collected manually through videos of vehicles, or they can be obtained from websites like Google Image Search.

Data pre-processing is where the datasets is processed in a way so that it is able to be understood by the deep learning classifier model. First, the objects in the images must be labelled by drawing bounding boxes over them and link them with an object class or category. This is a labour-intensive part of the process. This step is required to allow the deep learning model to recognise each object and linking them to their respective classes.

Model Implementation is training the algorithm in deep learning classifier models such as YOLO and R-CNN. These models consist of neural networks that learns a mapping function from inputs to outputs. This is accomplished by changing the network's weights in accordance to the model's errors on the training dataset. Adjustments are made to continuously decrease this error until a suitable model is discovered or the learning process becomes stuck and terminates. The process of training neural networks is by far the most time-consuming aspect of utilising the method in general, both in terms of configuration effort and computing complexity needed to perform the process (Brownlee 2021).

While training a model is critical, how the model generalises to unseen data is as critical and should be included into any deep learning process. One commonly used method of evaluation is cross-validation. Cross-validation is a statistical method that divides the original observation dataset into two sets, a training set for training the model and an independent set for evaluating the model. Metrics for model evaluation are needed in order to measure model performance. The evaluation metrics used are determined by the deep learning job at hand (such as classification, regression and clustering). Certain measures, such as precision-recall, are applicable to a variety of applications. The majority of deep learning applications use supervised learning problems such as classification and regression. Examples for classification metrics are classification accuracy, confusion matrix, logarithmic loss, f-measure and others.

When our model is having errors, such as bias or low accuracy, hyperparameters can be tuned to achieve better training for the deep learning classifier model which results in better prediction accuracy. There are quite a few hyperparameters and variables that can be tuned:

i. Tune Learning Rate

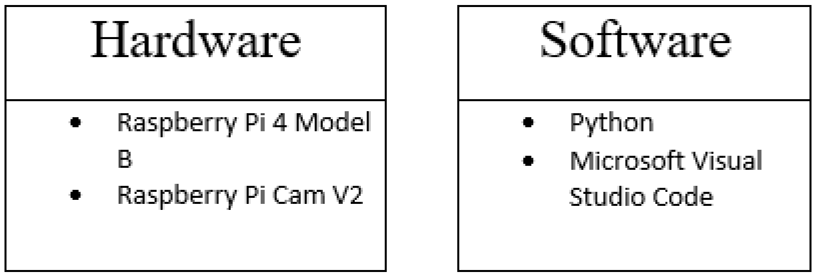
ii. Tune Regularization Parameter

iii. Tune Training Epoch

iv. Use Different Cost functions

v. Initialize weights differently

3.2 Equipment



Hardware	Software
• Raspberry Pi 4 Model B • Raspberry Pi Cam V2	• Python • Microsoft Visual Studio Code

Figure 3. 3: Equipment List for the Development of Proposed Automatic Traffic Counting System

Figure 3.3 shows the list of equipment required for the development of the proposed automatic traffic counting system for this project. The equipment consists of both hardware and software. The hardware's main purpose is to capture a video recording of the vehicles on the road, which is the data collection part of this project. The software part is the analysis and processing of the video data, such as object detection, tracking and classification.

3.3 Hardware Configuration

Figure 3.4 shows the hardware configuration for the proposed automatic traffic counting system. It consists of the Raspberry Pi Cam V2 and the Raspberry Pi 4 Model B. The Raspberry Pi Cam V2 is an 8MP camera with a Sony IMX219 sensor that is able to capture high-resolution photos, and videos with resolution up to 1080p. The Pi Cam V2 is connected to the Raspberry Pi by inserting the Pi Cam's flex cable into the CAMERA connector.

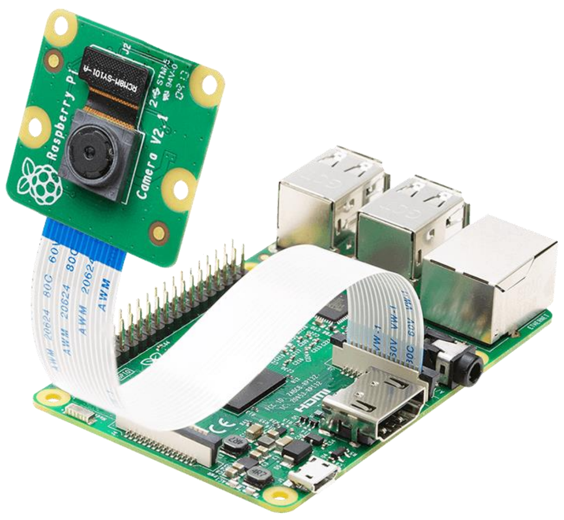


Figure 3. 4: Hardware Configuration for the proposed Automatic Traffic Counting System

3.4 Programming Language and Environment

Programming language and programming environment forms an important part of the software of the proposed automatic traffic counting data collection and analysis system. Various image processing techniques and machine learning algorithms will be used in the proposed system. Thus, the best programming language and environment must be chosen to provide the best performance and codability for this project.

Python is the preferred programming language for coding by many data scientists. Python is easy to learn and use because of its simplified syntax, which places a higher focus on natural language. Python can be easily written even by newcomers and executed faster than other programming languages. Python also has many open computer science related libraries such as Computer Vision, Machine Learning, Image Processing and others which makes it a great coding tool for computer science related projects and tasks.

Visual Studio Code is a code editor that combines a source code editor with powerful developer tools that eases the coding and debugging process, such as IntelliSense code completion and debugging. It’s simple to use interface allows for more time and effort spent on implementing ideas in the code, rather than struggling to setup the environment.

3.5 Data Collection

Data collection is to collect data that will be used for the vehicle detecting, tracking, counting, classification model. In this project, the data being collected is the video of the vehicles on the road. A video is captured when the Python program in Figure 3.6 runs. The frame captured is shown in Figure 3.5. The video shot for this project is in H.264 format with a resolution of 640 x 480 and 25 frames per second (fps). H.264 is one of the most widely used codec in the world, with it being used in optical disc, broadcasts, and other video medias. H.264 cannot be directly viewed by most media players. Thus, it has to be incorporated into different container formats such as MEPG-4, QuickTime, Flash and others. The resolution is set at 640 x 480 to save storage space as multiple hours of video have to be recorded and stored inside the 12GB microSD card. 25 fps is the standard for movies and TV shows, and it is the minimum speed needed to capture video that have realistic motion (Ozer, 2011).

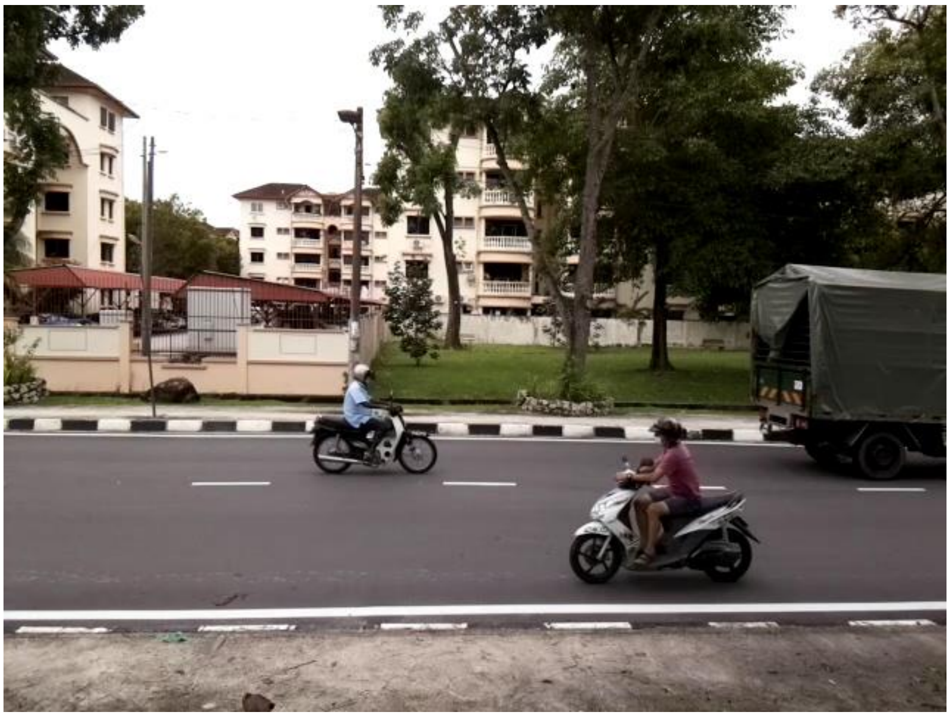
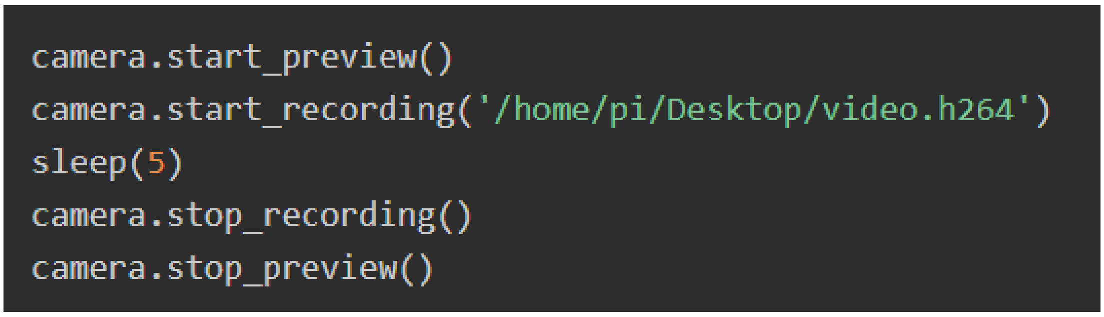


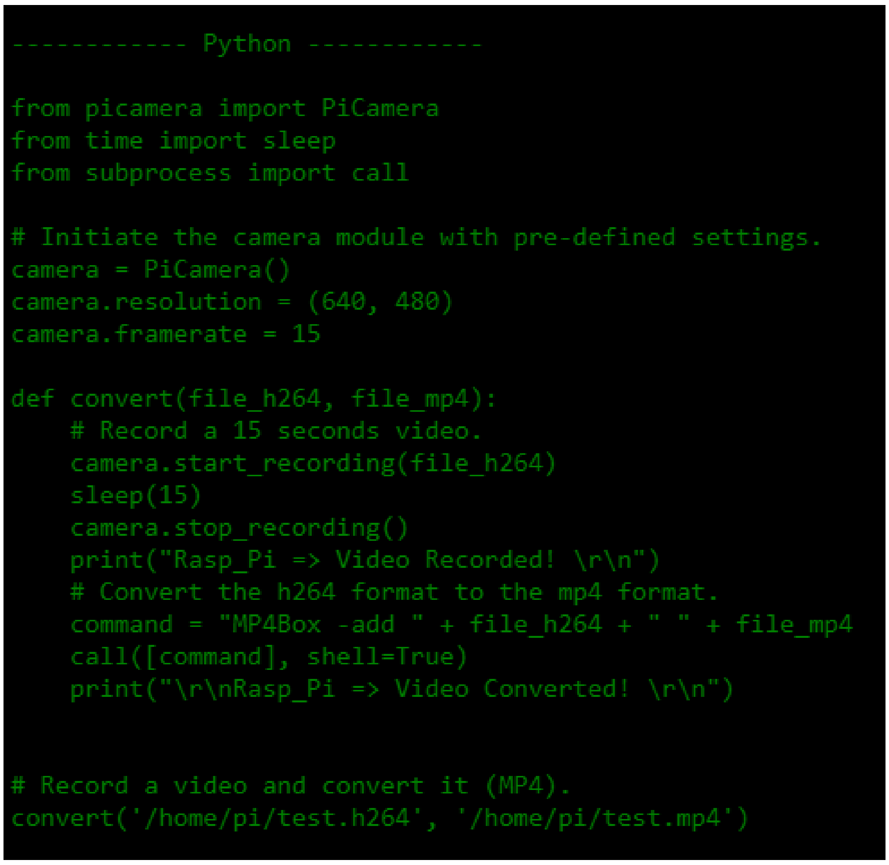
Figure 3. 5: A frame from the traffic video



```
camera.start_preview()
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(5)
camera.stop_recording()
camera.stop_preview()
```

Figure 3. 6: Python program to capture video on the Raspberry Pi

After collecting the data (video), the data might not be in a form that is needed to be fitted into the vehicle detecting tracking, counting and classification model. First, the video shot by the Raspberry Pi is in H.264 which is a video codec. The video has to be processed to be contained in a container such as MP4. To accomplish this, the video is converted from H.264 to MP4 using the Python program in Figure 3.7.



```
------------ Python ------------

from picamera import PiCamera
from time import sleep
from subprocess import call

# Initiate the camera module with pre-defined settings.
camera = PiCamera()
camera.resolution = (640, 480)
camera.framerate = 15

def convert(file_h264, file_mp4):
    # Record a 15 seconds video.
    camera.start_recording(file_h264)
    sleep(15)
    camera.stop_recording()
    print("Rasp_Pi => Video Recorded! \r\n")
    # Convert the h264 format to the mp4 format.
    command = "MP4Box -add " + file_h264 + " " + file_mp4
    call([command], shell=True)
    print("\r\nRasp_Pi => Video Converted! \r\n")


# Record a video and convert it (MP4).
convert('/home/pi/test.h264', '/home/pi/test.mp4')
```

Figure 3. 7: Python program to convert video from H.264 to MP4 format

3.6 Bill of Materials

The materials used, quantity, unit price, and total price are listed in Table 3.1. The total cost of the materials used for this project is RM415.

Table 3. 1: Bill of Materials

Material	Quantity	Unit	Unit price (RM)	Total price (RM)
Raspberry Pi 4 Model B	1	1	174	174
Official Case for Raspberry Pi 4B (red/White)	1	1	25	25
16GB Micro SD Card with NOOBS for RPI	1	1	36	36
Raspberry Pi 8MP Camera Module V2	1	1	115	115
Power bank	1	1	50	50
USB-C cable	1	1	15	15
			Total Cost (RM)	415

3.7 Project Management

The project timeline for FYP 1 is shown in Table 3.1 while the project timeline for FYP 2 is shown in Table 3.2.

Table 3. 2: Gantt Chart for FYP 1

Week / Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Project Selection	■	■												
Literature Review			■	■	■	■	■	■	■					
Methodology Research					■	■	■	■	■	■	■			
Hardware selection							■	■	■	■				
Configure Hardware								■	■	■				
Record footage for data									■	■	■	■		
Process data and build software										■	■	■	■	■

Table 3. 3: Gantt Chart for FYP 2

Week / Task	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Build Speed Measuring Algorithm	▨	▨												
Debug and test Speed Measuring Algorithm		▨	▨											
Prepare training dataset for Deep Learning Classification	▨	▨	▨											
Train Object Detection Model				▨										
Finetune Object Detection Model				▨	▨									
Integrate all modules together						▨	▨	▨	▨	▨				
Run tests and record results and discussion										▨	▨			

CHAPTER 4

RESULTS AND DISCUSSIONS

4.1 Hardware setup

The Raspberry Pi is connected to a display to view the viewfinder for the camera. Indoors, the Raspberry Pi is interfaced with a monitor through HDMI, however this is not to viable outdoors. To solve this issue, the Raspberry Pi can display its content and can be remote controlled on a smart phone when being placed outdoors. This is done by connecting the Raspberry Pi to an Android smart phone through VNC Server, which is completely free and is pre-installed on the Raspberry Pi by default. A direct connection can be established by having both devices connecting over the same private local network. Mobile data hotspot sharing is used to connect the smartphone with the Raspberry Pi.

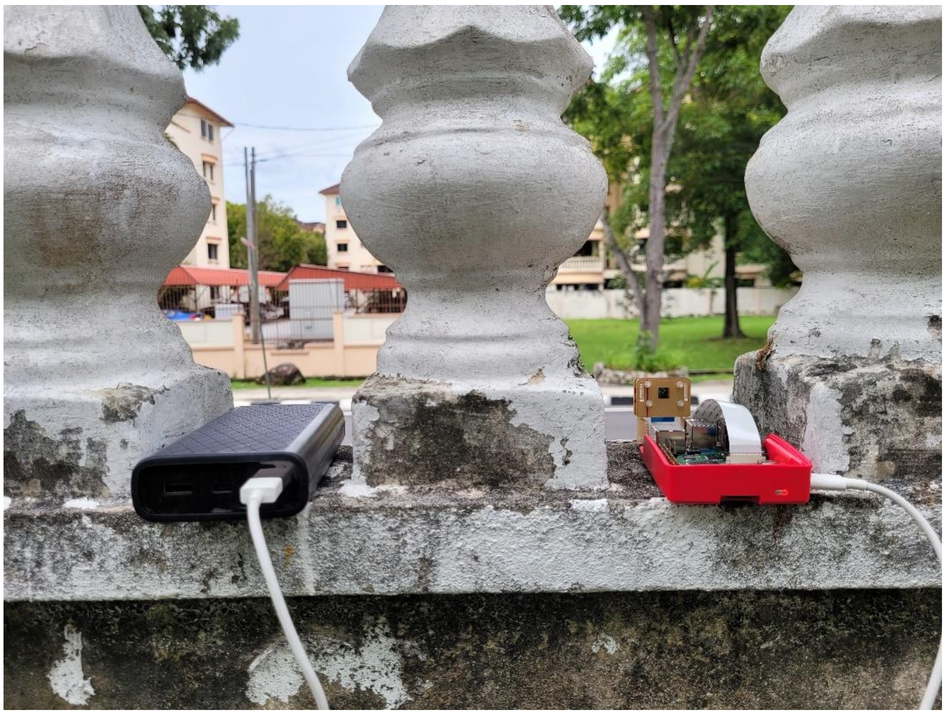


Figure 4. 1: Setup of hardware to collect traffic data outdoors

Once the Raspberry Pi is able to be remotely controlled, the Raspberry Pi is placed at a spot outdoor where the camera is able to get a good view of the road outside as seen in Figure 4.1. The camera is perpendicular to the road. The Raspberry Pi only requires power at this point, which is provided by the 20,000 mAh power bank.

In Figure 4.2, the viewfinder for the camera is displayed on the smartphone after the Python program starts. The camera is placed perpendicular to the road, and it is able to get a clear side view of vehicles that pass by. Vehicles will move to the left on the nearby lane, while vehicles will move to the right on the further end lane.

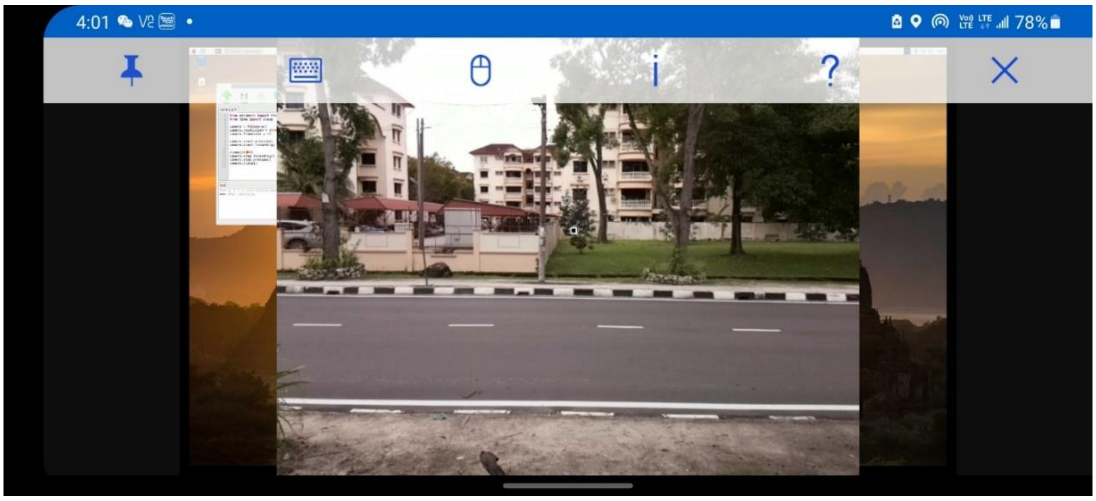


Figure 4. 2: Remote view of the Raspberry Pi Interface

4.2 Software Setup

4.2.1 Vehicle Counting

The video recorded in .mp4 format consists of 25 frames per second. The frames are extracted one by one to be applied with image processing to be able to detect the vehicles. Figure 4.3 to figure 4.8 shows the step by step processing done to the image by the algorithm.

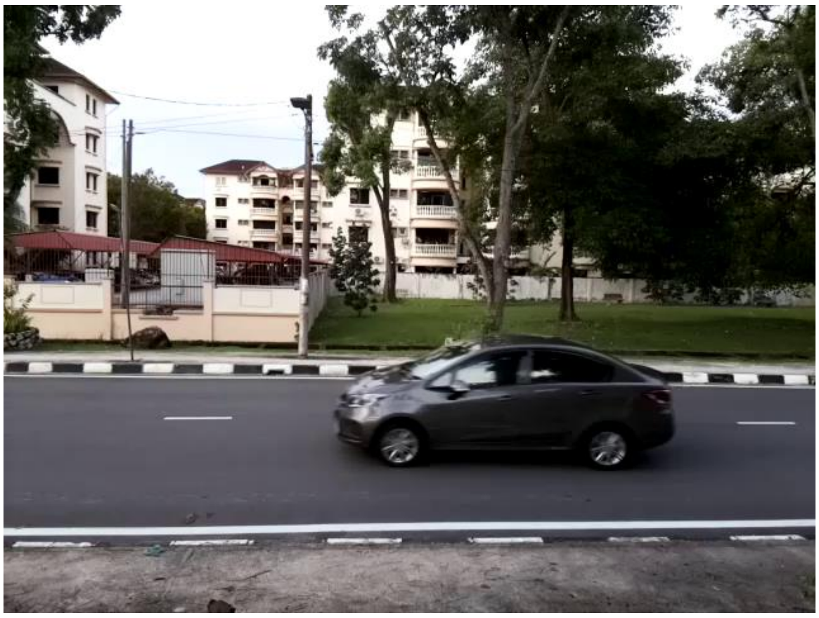


Figure 4.3: Original Frame

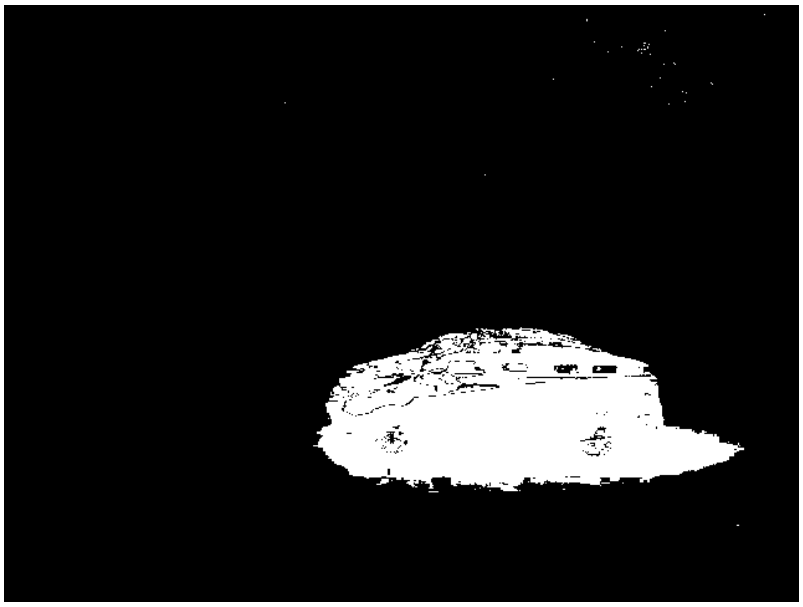


Figure 4. 4: Frame after applying MOG2BackgroundSubtractor

After the frame is extracted, the background is subtracted by the frame using OpenCV's MOG2 Background Subtractor function. The output image is the subject in white while the background is in black.

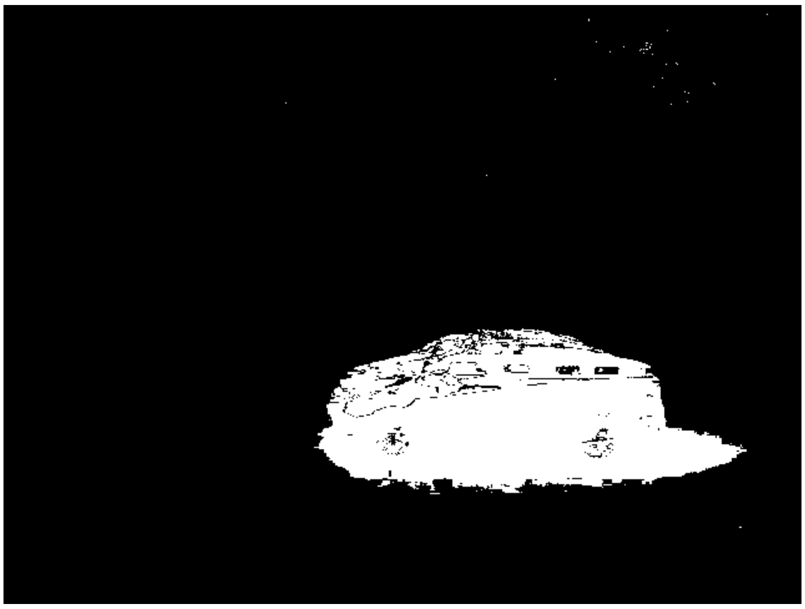


Figure 4. 5: Frame after applying binary image thresholding

The resulting image is then applied with binary image thresholding to ensure that the image is entirely greyscale for further processing. There are white specks in the black background and black specks in the white foreground which will have to remove to prevent OpenCV's findContours function to mistake the white specks as our Region of Interest (ROI). To accomplish this, morphology opening, and closing is used in conjunction.

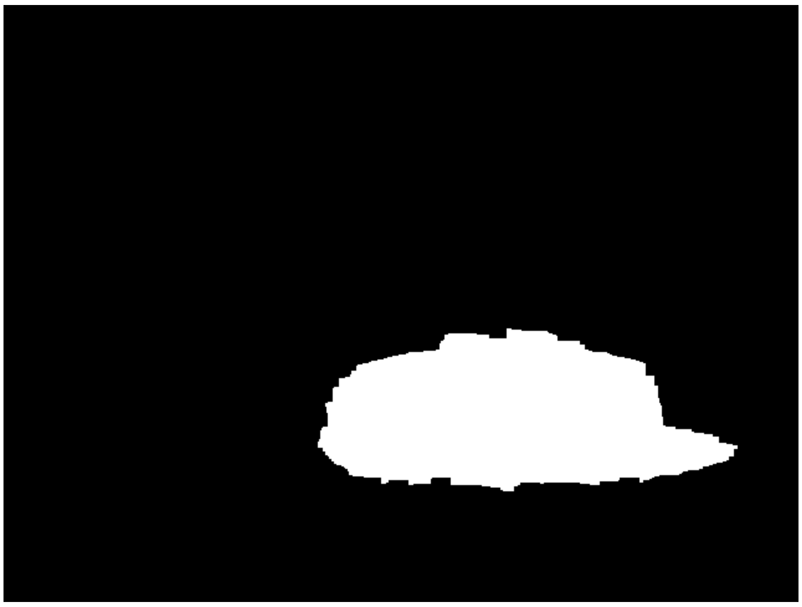


Figure 4. 6: Frame after applying Morphology Opening

After morphology opening is applied, all the white and black specs are smoothened into the foreground and background. The rough edge of the white foreground is also smoothened out. The resulting image only has one foreground which is the vehicle to be detected.

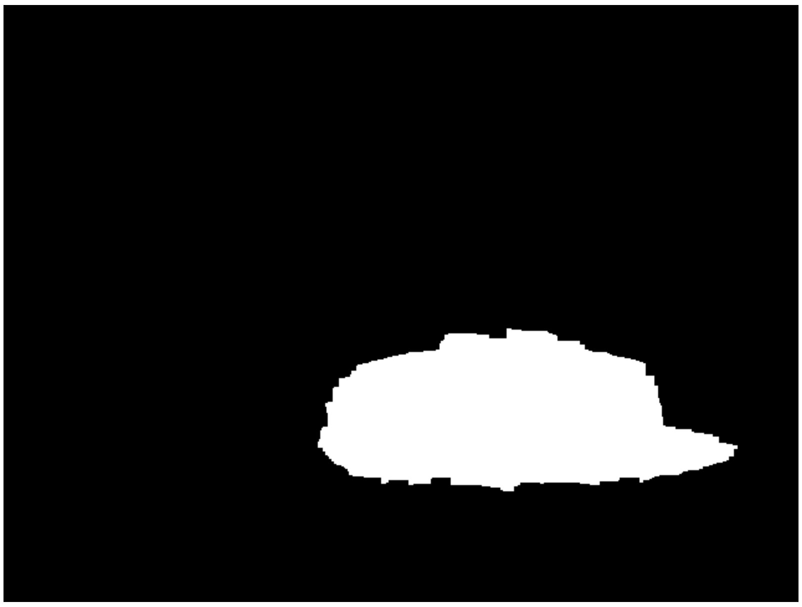


Figure 4. 7: Frame after applying Morphology Closing

Applying morphology closing smoothens the edges further of the white foreground. The image is ready to be used to find contours by OpenCV’s findContours function.

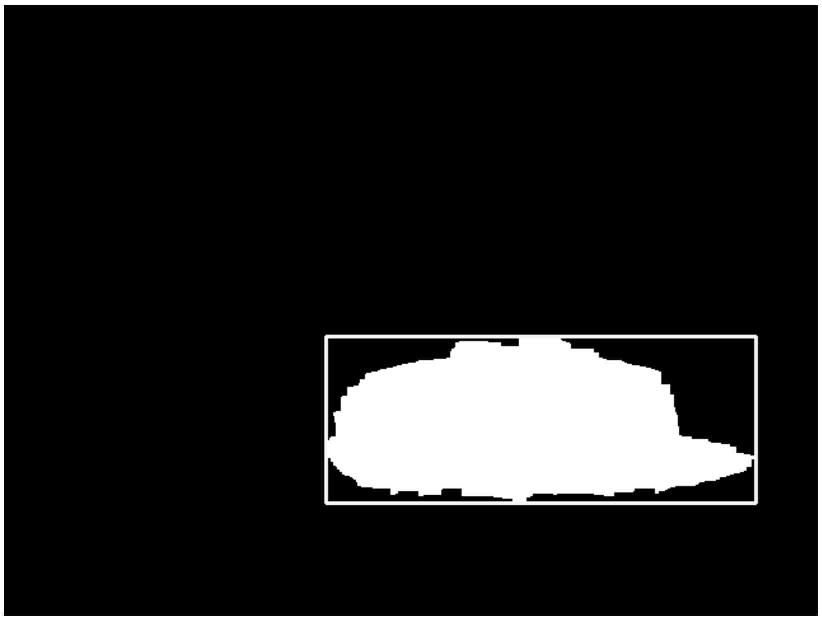


Figure 4. 8: Contour detected by OpenCV findContours function

OpenCV's findContours function detects changes in image colour and marks it as a contour. The external retrieval mode of the function is used, which only stores the extreme outer contours, while inner contours are ignored (Shaikh, 2020). This is shown in Figure 4.9(Shaikh, 2020).

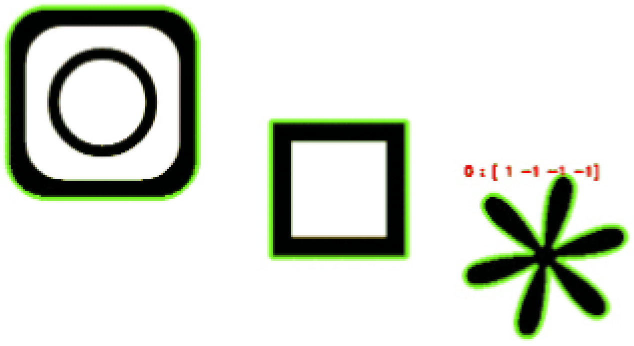


Figure 4. 9: Example of external retrieval mode

After the contours are found, contours with area between 2000 and 60000 px^2 are assumed to be vehicles. Since the parameters of the bounding rectangles are known, the centroid of the vehicle is found using the formula.

When the centroid is enters the range of left and right limit, the contour is registered as a new vehicle and the centroid coordinate are recorded. The next frame is extracted, and the same image processing happens. If it is a new car, the centroid will be registered as a new car. Else, it updates the coordinate of the old car. When the coordinate passes the middle lines, the code counts the vehicle. This is to ensure the object is travelling a certain distance before being counted as a vehicle. The code can determine whether the vehicle is heading to the right or to the left based on the direction the centroid is heading and count accordingly.

4.2.2 Vehicle Speed Measuring

The vehicles is detected and tracked using the algorithm in vehicle counting. The output of the algorithm is the bounding rectangle and centroid of each vehicle in the current frame. For speed measuring, the vehicle centroid is used to calculate the speed of the vehicle.

However, due to the camera's field of view (Figure 4.10), the L increases as the distance of the object from the lens increases. Thus, *c* for vehicles going to the right is larger than the *c* for vehicles going to the left. To calculate time, the frame rate of video and frames travelled when car travels between two points in the video is required.

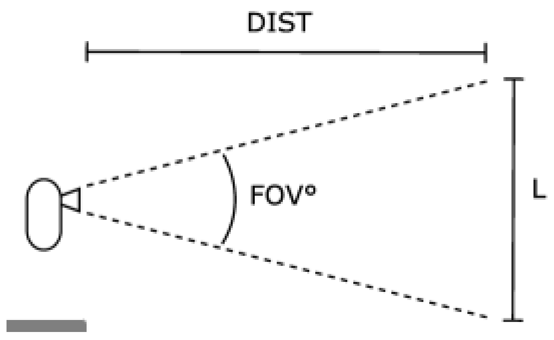


Figure 4. 10: Camera's field of view

The calculations for the speed measurements are using the formulas of Eq. (4.1), Eq. (4.2), Eq. (4.3) and Eq. (4.4).

$$meters\ per\ pixel = mpp = \frac{distance\ constant}{frame\ width}$$

Eq. (4.1)

$$distance\ in\ pixels = \ p_{AB} = |\ col_B - col_A|$$

Eq. (4.2)

$$distance\ in\ meters\ \ between\ point\ AB = \ d_{AB} = P_{AB} * mpp$$

Eq. (4.3)

$$Average\ speed = \frac{\frac{d_{AB}}{\Delta t_{AB}} + \frac{d_{BC}}{\Delta t_{BC}} + \frac{d_{CD}}{\Delta t_{CD}}}{3}$$

Eq. (4.4)

In equation Eq. (4.1), the distance constant is the physical distance of the frame width. This can be measured by standing at the left edge of the frame and measure the distance until the right edge of the frame. To do this measurement safely, the measurement is done one the near end pavement and far end pavement. The near end pavement measures in at 6.12m, while the far end pavement measures at 19.33m. The distance constant of vehicles going to the left (near end) is set at 13m while for vehicles going to the right (far end) is set at 15.1m.

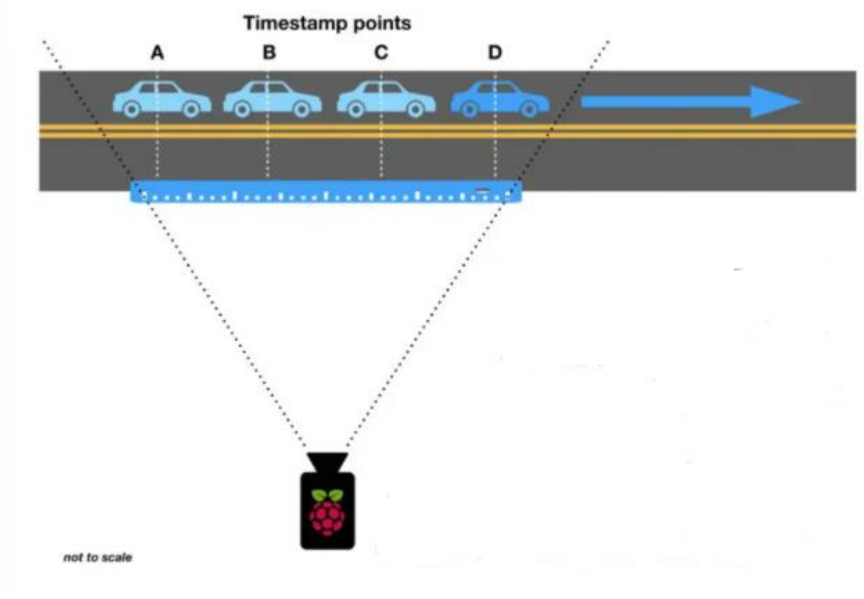


Figure 4. 11: Method of speed estimation

First, the vehicle's centroid location is recorded at four points in the frame which are A, B, C and D respectively as shown in Figure 4.11 (Rosebrock, 2019). For vehicles going to the right, the location is collected sequentially from A, B, C to D. For vehicles going to the left, the location is collected sequentially from D, C, B to A. The algorithm will also record the time when the vehicle centroid is at that point. After that, the centroid location is grouped into (A, B), (B, C) and (C, D). The distance in pixels between the two points in these three groups are calculated using equation Eq. (4.2). After that, the distance in meters between the two points are calculated using equation Eq. (4.3). After that, the average speed of the vehicle is calculated by using equation Eq. (4.4).

4.2.3 Vehicle Classification Model

4.2.3.1 Annotate images

After the data collection, data preparation is needed to convert image information into values that the deep learning algorithm can interpret. The objects in the video is being classified into 3 classes: car, motorcycle, and bicycle.

First, the video is needed to be converted into images or frames. This can be done by using a simple Python program made by Patel in 2019 using the OpenCV library. The program allows the user to set the frame capture rate of the program. For example, if the frame capture rate is set to 0.5, the program will capture one frame every 0.5 seconds, which means 2 frames will be captured each second. The frames are saved in a folder.

After the frames are extracted from the video, images without objects inside is deleted as it is not needed to train the deep learning model. This process took around 1 week.

Labels are used to help the training model identify unlabelled objects in the data (Nelson, 2020). Data that has been accurately labelled is essential to successful deep learning. LabelImg is a free, open-source program for labelling pictures visually. It's developed in Python and has a graphical user interface built using PyQT. It's a quick and painless approach to identify a few hundred photographs for your next object detection project. Once LabelImg is opened, the user needs to draw a rectangle box over the object and enter the label for the object. The label is saved in a separate .xml file for each image. This process has to repeated for every image; thus, this process is time consuming since training the deep learning requires few thousands to tens of thousands of images.

Around 4000 images extracted from videos captured is annotated for the project. The interface of LabelImg to label the images is shown in Figure 4.12.

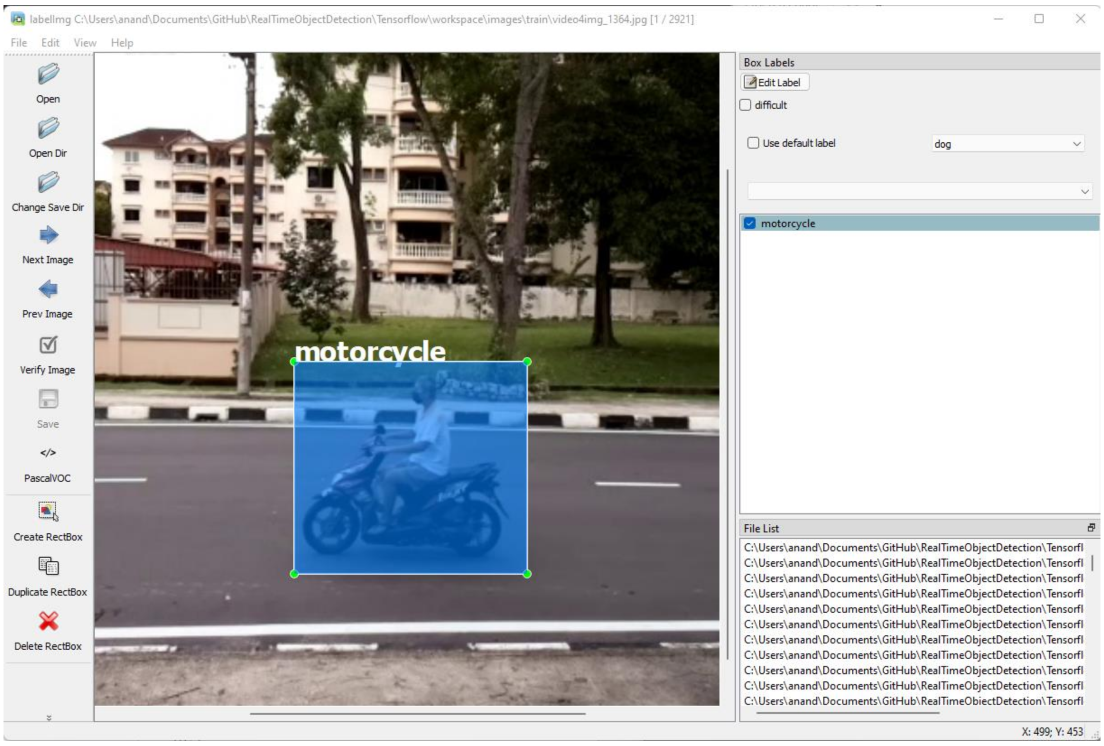


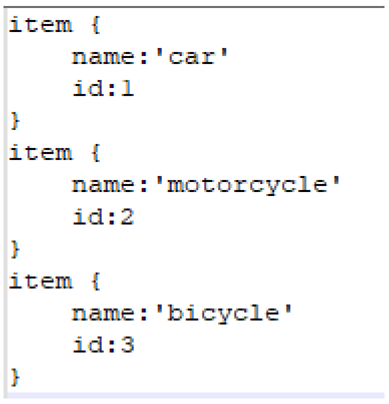
Figure 4. 12: Annotating images using labelImg

4.2.3.2 Partition the image dataset

After annotating every image, the image dataset is partitioned into two folders, test and training. The ratio is 9:1, which means 90% of the images are placed into the training folder while 10% of the images are used for testing. The images are placed into the two folders with the .xml files.

4.2.3.3 Create Label Map

Tensorflow requires a label map, which maps the class labels to integer values. The label map is saved as a .pbtxt (protobuf text) file. The contents of the label map is shown in Figure 4.13.



```
item {
    name:'car'
    id:1
}
item {
    name:'motorcycle'
    id:2
}
item {
    name:'bicycle'
    id:3
}
```

Figure 4. 13: Label Map for the current project

4.2.3.4 Create Tensorflow Records

Tensorflow object detection API is not able to directly read each .xml file. The annotation data in the .xml files must be compiled into a TFRecord format. This can be accomplished by running a pre-written script that iterates through all .xml files in the train and test folder and outputs a .record file for each.

4.2.3.5 Download Pre-Trained Model

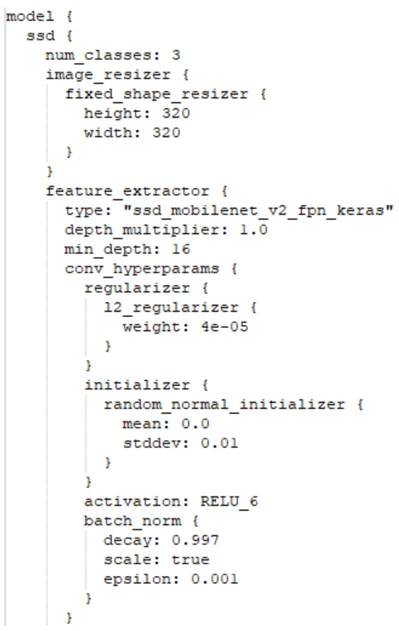
The main aim for training a neural network is using several forward and backward iterations to get the right weights for the neural network.

For this project, transfer learning is used, which means using a pre-trained model and training the model with the self-collected data. By using pre-trained models that have been trained on large datasets, the weights and architecture previously learnt by the model can be used immediately to apply the learning to the car classification project’s problem statement. The learning from the model is “transferred” which saves a huge amount of time and effort (Analytics Vidhya, 2017). For example, ImageNet has 1.2 million images that are used to create a generalized object detection model. The model can classify the images into 1000 separate object classes. Through transfer learning, these pre-trained networks based on the model is able to generalize object classes from images rather accurately.

First, a pre-trained model is downloaded from the TensorFlow 2 Detection Model Zoo. The Model Zoo consists of various pre-trained models which differ in speed, mAP (mean average precision) and output. SSD MobileNet v2 320x320 is chosen as it is quick with a speed of 19ms and a respectable mAP of 20.2.

4.2.3.6 Configure Training Pipeline

After the pre-trained model is downloaded, the pipeline.config file consists of the parameters of the neural network training that can be changed. For this project, the number of classes is set to 3 since the objects are categorized into 3 classes. The batch size, which is the number of training examples that passed through the neural network in one iteration, is set to 4. This means 4 samples from the training dataset are taken to train the network each iteration until every sample is propagated through the network. Part of the pipeline.config is shown in Figure 4.14.

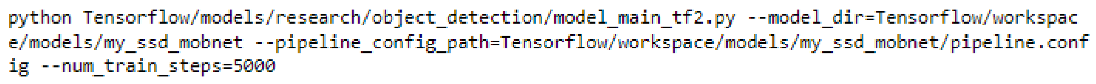


```
model {
  ssd {
    num_classes: 3
    image_resizer {
      fixed_shape_resizer {
        height: 320
        width: 320
      }
    }
    feature_extractor {
      type: "ssd_mobilenet_v2_fpn_keras"
      depth_multiplier: 1.0
      min_depth: 16
      conv_hyperparams {
        regularizer {
          l2_regularizer {
            weight: 4e-05
          }
        }
        initializer {
          random_normal_initializer {
            mean: 0.0
            stddev: 0.01
          }
        }
        activation: RELU_6
        batch_norm {
          decay: 0.997
          scale: true
          epsilon: 0.001
        }
      }
```

Figure 4. 14: Part of the pipeline.config

4.2.3.7 Training the model

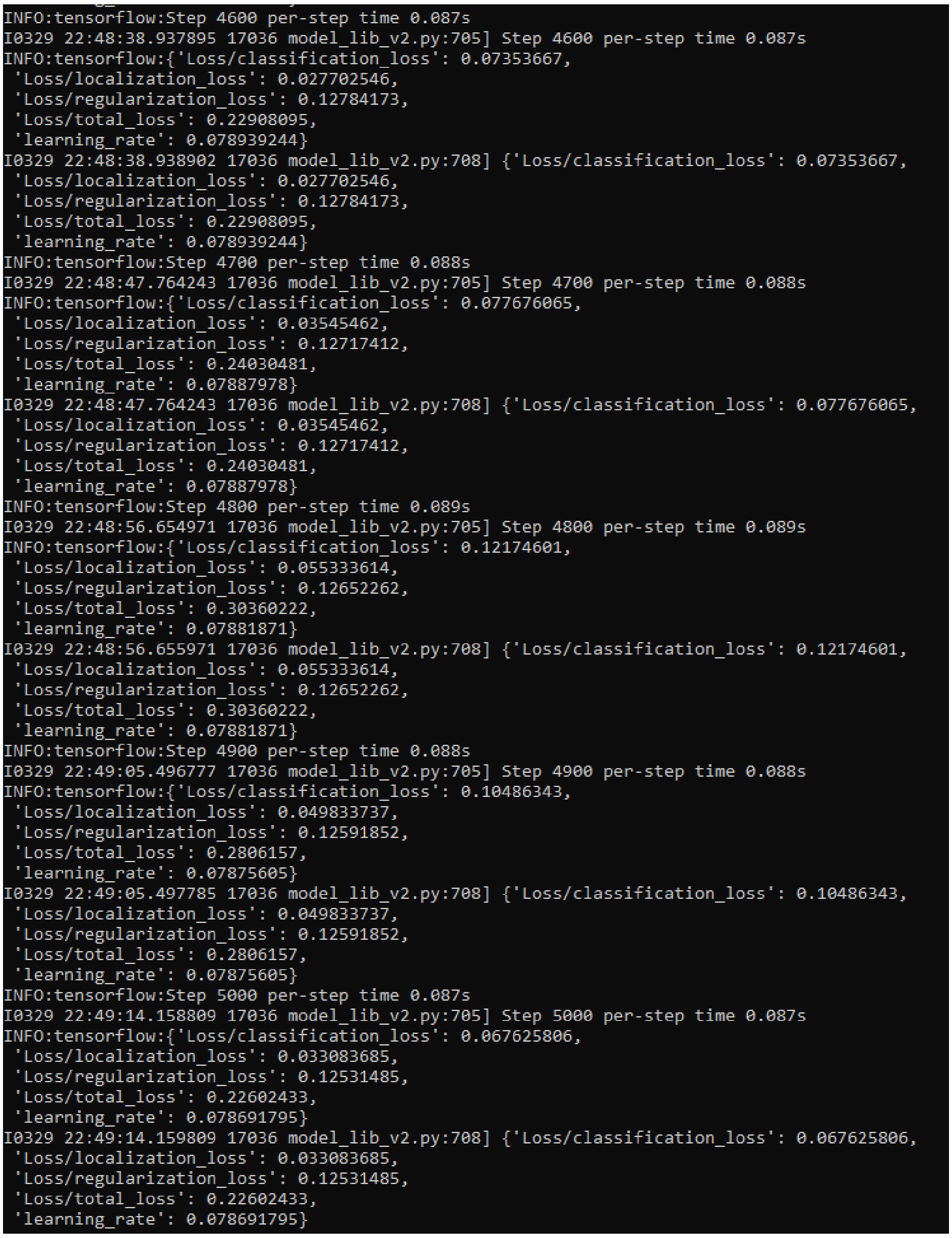
To initiate the training process, the model_main_tf2.py script is run through the terminal with the command in Figure 4.15. The model directory, pipeline config path and number of training steps are parsed into the script as arguments. A training step is one gradient update, where one batch size of examples is processed (Tolotra Samuel, 2018).



```
python Tensorflow/models/research/object_detection/model_main_tf2.py --model_dir=Tensorflow/workspac
e/models/my_ssd_mobnet --pipeline_config_path=Tensorflow/workspace/models/my_ssd_mobnet/pipeline.conf
ig --num_train_steps=5000
```

Figure 4. 15: Command to initiate training

Once training is initiated, the output is printed as in Figure 4.16. During the training, the current step, step time, classification loss, regularization loss, total loss and learning rate is shown.



```
INFO:tensorflow:Step 4600 per-step time 0.087s
I0329 22:48:38.937895 17036 model_lib_v2.py:705] Step 4600 per-step time 0.087s
INFO:tensorflow:{'Loss/classification_loss': 0.07353667,
 'Loss/localization_loss': 0.027702546,
 'Loss/regularization_loss': 0.12784173,
 'Loss/total_loss': 0.22908095,
 'learning_rate': 0.078939244}
I0329 22:48:38.938902 17036 model_lib_v2.py:708] {'Loss/classification_loss': 0.07353667,
 'Loss/localization_loss': 0.027702546,
 'Loss/regularization_loss': 0.12784173,
 'Loss/total_loss': 0.22908095,
 'learning_rate': 0.078939244}
INFO:tensorflow:Step 4700 per-step time 0.088s
I0329 22:48:47.764243 17036 model_lib_v2.py:705] Step 4700 per-step time 0.088s
INFO:tensorflow:{'Loss/classification_loss': 0.077676065,
 'Loss/localization_loss': 0.03545462,
 'Loss/regularization_loss': 0.12717412,
 'Loss/total_loss': 0.24030481,
 'learning_rate': 0.07887978}
I0329 22:48:47.764243 17036 model_lib_v2.py:708] {'Loss/classification_loss': 0.077676065,
 'Loss/localization_loss': 0.03545462,
 'Loss/regularization_loss': 0.12717412,
 'Loss/total_loss': 0.24030481,
 'learning_rate': 0.07887978}
INFO:tensorflow:Step 4800 per-step time 0.089s
I0329 22:48:56.654971 17036 model_lib_v2.py:705] Step 4800 per-step time 0.089s
INFO:tensorflow:{'Loss/classification_loss': 0.12174601,
 'Loss/localization_loss': 0.055333614,
 'Loss/regularization_loss': 0.12652262,
 'Loss/total_loss': 0.30360222,
 'learning_rate': 0.07881871}
I0329 22:48:56.655971 17036 model_lib_v2.py:708] {'Loss/classification_loss': 0.12174601,
 'Loss/localization_loss': 0.055333614,
 'Loss/regularization_loss': 0.12652262,
 'Loss/total_loss': 0.30360222,
 'learning_rate': 0.07881871}
INFO:tensorflow:Step 4900 per-step time 0.088s
I0329 22:49:05.496777 17036 model_lib_v2.py:705] Step 4900 per-step time 0.088s
INFO:tensorflow:{'Loss/classification_loss': 0.10486343,
 'Loss/localization_loss': 0.049833737,
 'Loss/regularization_loss': 0.12591852,
 'Loss/total_loss': 0.2806157,
 'learning_rate': 0.07875605}
I0329 22:49:05.497785 17036 model_lib_v2.py:708] {'Loss/classification_loss': 0.10486343,
 'Loss/localization_loss': 0.049833737,
 'Loss/regularization_loss': 0.12591852,
 'Loss/total_loss': 0.2806157,
 'learning_rate': 0.07875605}
INFO:tensorflow:Step 5000 per-step time 0.087s
I0329 22:49:14.158809 17036 model_lib_v2.py:705] Step 5000 per-step time 0.087s
INFO:tensorflow:{'Loss/classification_loss': 0.067625806,
 'Loss/localization_loss': 0.033083685,
 'Loss/regularization_loss': 0.12531485,
 'Loss/total_loss': 0.22602433,
 'learning_rate': 0.078691795}
I0329 22:49:14.159809 17036 model_lib_v2.py:708] {'Loss/classification_loss': 0.067625806,
 'Loss/localization_loss': 0.033083685,
 'Loss/regularization_loss': 0.12531485,
 'Loss/total_loss': 0.22602433,
 'learning_rate': 0.078691795}
```

Figure 4. 16: Printout during training

4.2.3.8 Evaluating the model

Figure 4.17 to Figure 4.21 shows the graph of evaluation metrics used to evaluate how well the deep learning model is trained and performing.

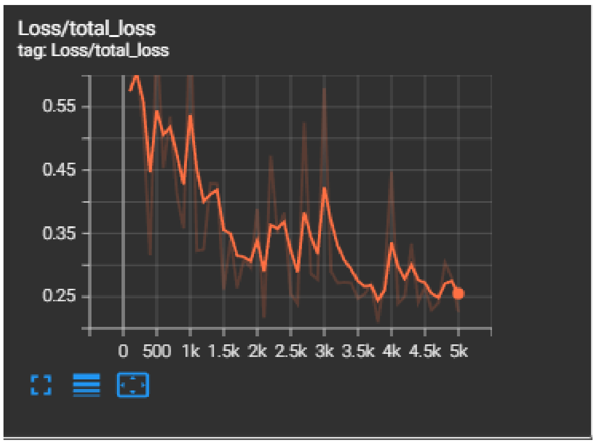


Figure 4. 17: Loss metric during the training of the model

In deep learning, the main objective of the training of the neural network is to minimize error. This objective function is referred to as "loss". From Figure 4.17, the loss starts out at 0.56 during step 0, and it flactuates but the overall loss is decreasing. The loss reaches a decent 0.25 during the final step.

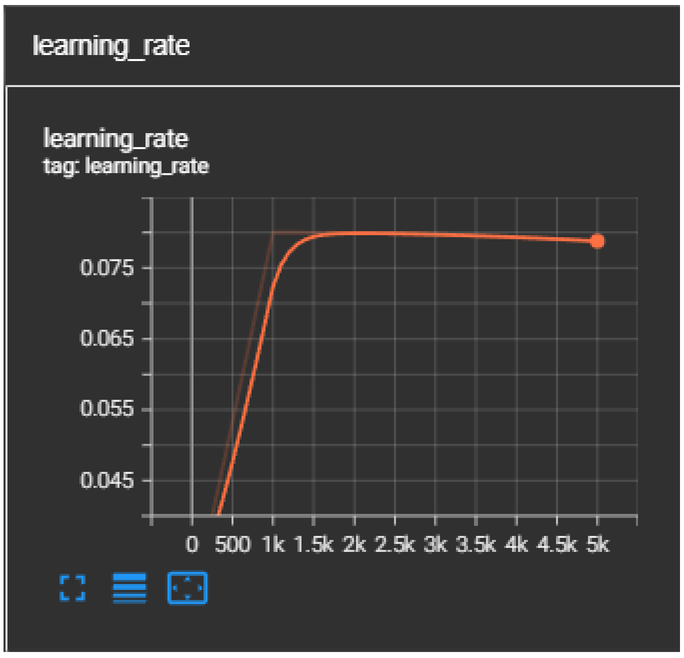


Figure 4. 18: Learning rate of the model

Learning rate is defined as the number of weights that is updated during each step size when the model is training. Figure 4.18 shows the learning rate of the model against the number of steps during training.

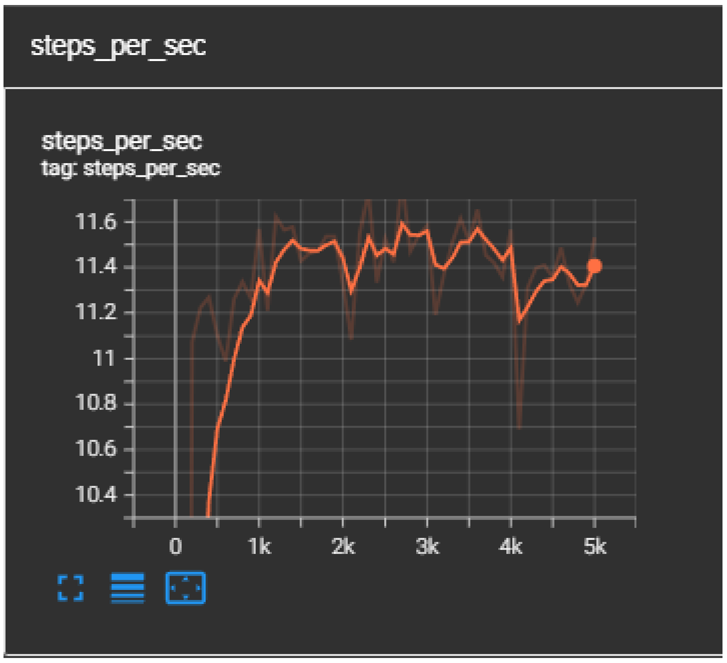


Figure 4. 19: Steps per second during training

Figure 4.19 shows the steps per second is directly correlated to the computational power of the GPU or CPU hardware used. The higher the computational power, the higher the steps per second.

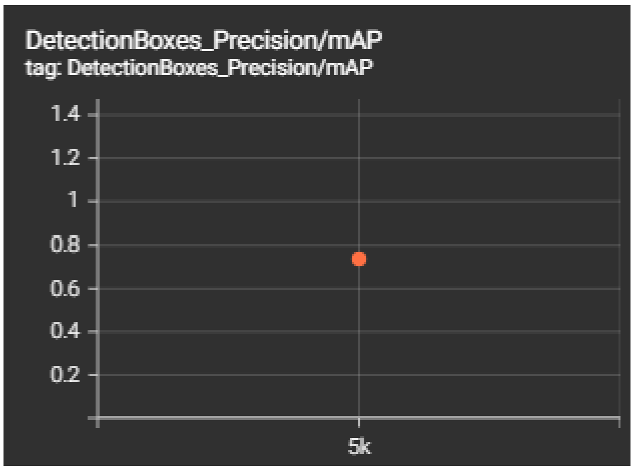


Figure 4. 20: mAP of the object detection model

From Figure 4.20, the mAP of the model trained is 0.7372. This means that the model can accurately classify vehicles of various classes 73.72 times out of 100 predictions. From Figure 4.21, the average recall of the model is 0.7795. The performance of the trained deep learning model is decent.

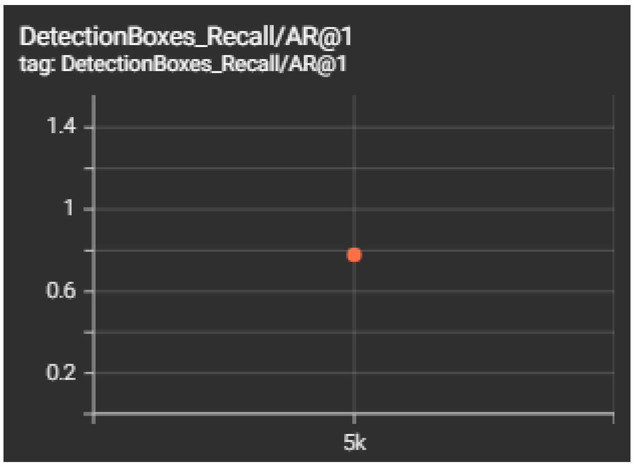
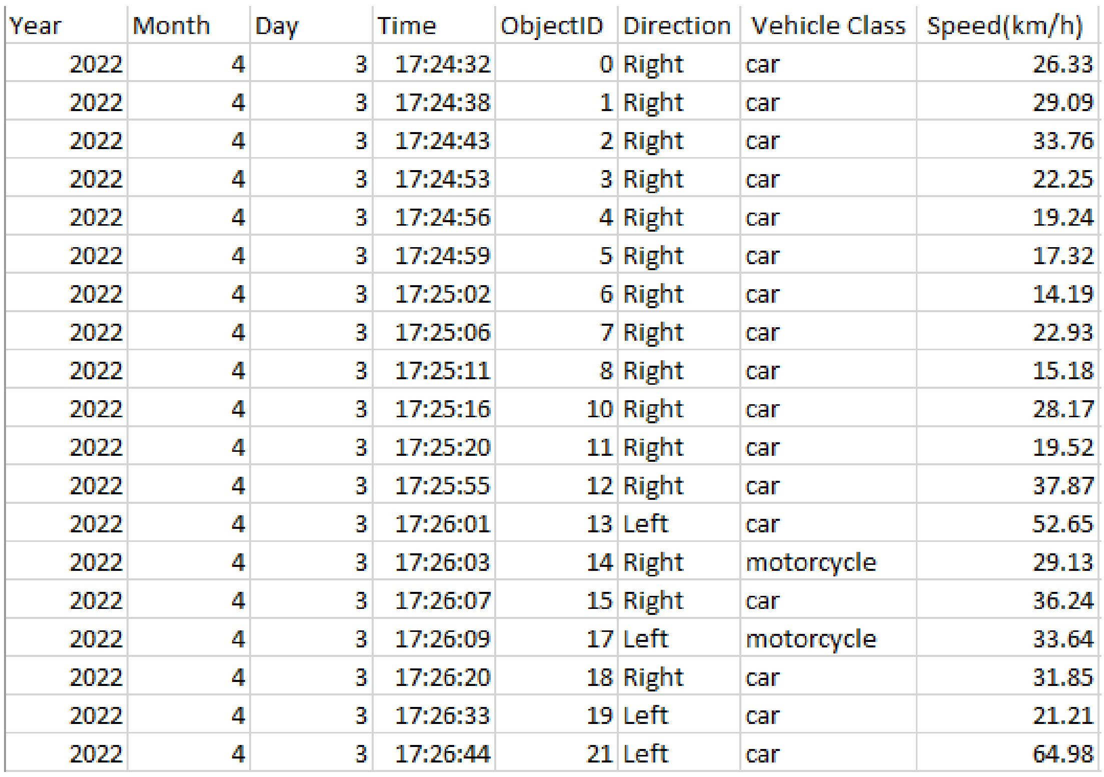


Figure 4. 21: Average recall of the model

4.2.4 Data Logging



Year	Month	Day	Time	ObjectID	Direction	Vehicle Class	Speed(km/h)
2022	4	3	17:24:32	0	Right	car	26.33
2022	4	3	17:24:38	1	Right	car	29.09
2022	4	3	17:24:43	2	Right	car	33.76
2022	4	3	17:24:53	3	Right	car	22.25
2022	4	3	17:24:56	4	Right	car	19.24
2022	4	3	17:24:59	5	Right	car	17.32
2022	4	3	17:25:02	6	Right	car	14.19
2022	4	3	17:25:06	7	Right	car	22.93
2022	4	3	17:25:11	8	Right	car	15.18
2022	4	3	17:25:16	10	Right	car	28.17
2022	4	3	17:25:20	11	Right	car	19.52
2022	4	3	17:25:55	12	Right	car	37.87
2022	4	3	17:26:01	13	Left	car	52.65
2022	4	3	17:26:03	14	Right	motorcycle	29.13
2022	4	3	17:26:07	15	Right	car	36.24
2022	4	3	17:26:09	17	Left	motorcycle	33.64
2022	4	3	17:26:20	18	Right	car	31.85
2022	4	3	17:26:33	19	Left	car	21.21
2022	4	3	17:26:44	21	Left	car	64.98

Figure 4. 22: Example of Data Logged into XML file

The data log file is saved in xml format. The date, time, vehicle ID, direction, class and speed is all recorded into the data log file. The date, time recording is especially useful when the algorithm is processing a live feed of the traffic.

4.3 Vehicle Counting Evaluation

4.3.1 Test samples

To evaluate the algorithm, six 15 minutes videos of vehicles is recorded using the hardware setup in Chapter 4.1. The six videos are recorded on two days, with 3 videos each day. On the first day, the 3 videos are recorded in the morning, afternoon and evening. This step is repeated for the second day. The first day is a weekend day while the second day is weekday. Figure 4.23 to Figure 4.26 shows the scenes recorded during the 6 videos. The scenes have varied lighting conditions.

After the video is recorded, manual counting is used to accurately determine the total number of vehicles, total vehicles going to the right, total vehicles going to the left, and finally the class type of each vehicle.

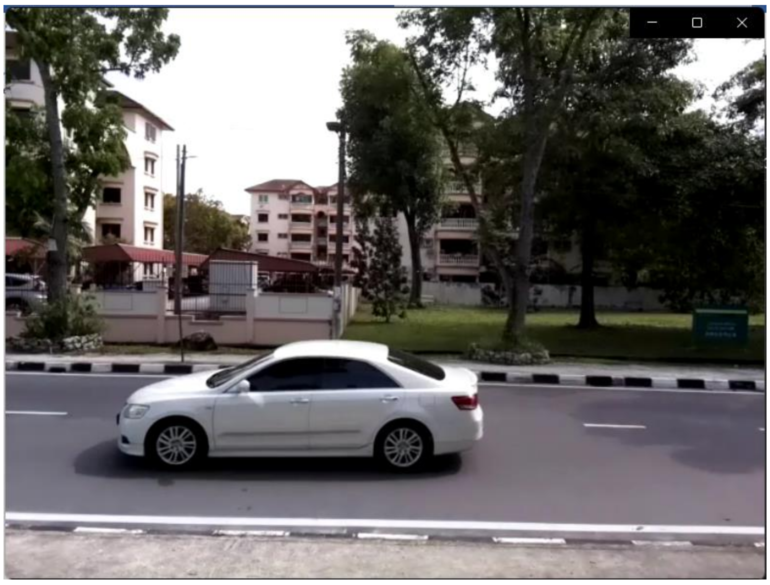


Figure 4. 23: Scene recorded in the morning

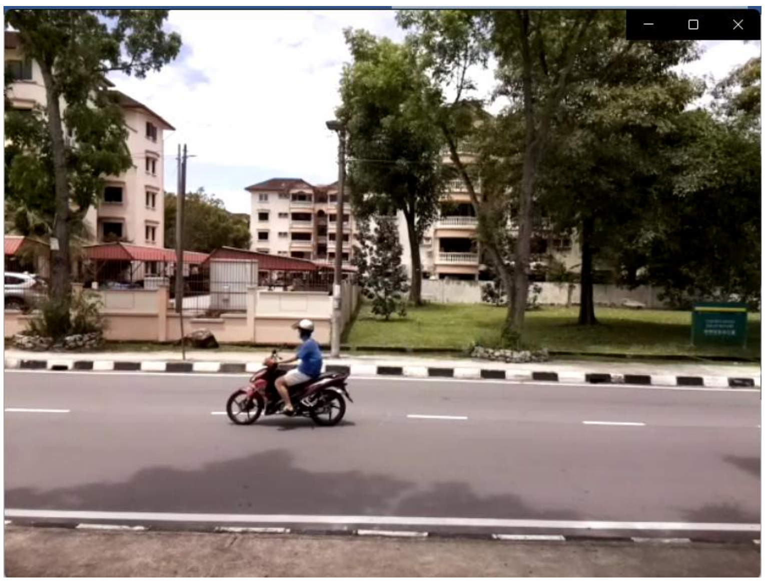


Figure 4. 24: Scene recorded in the afternoon

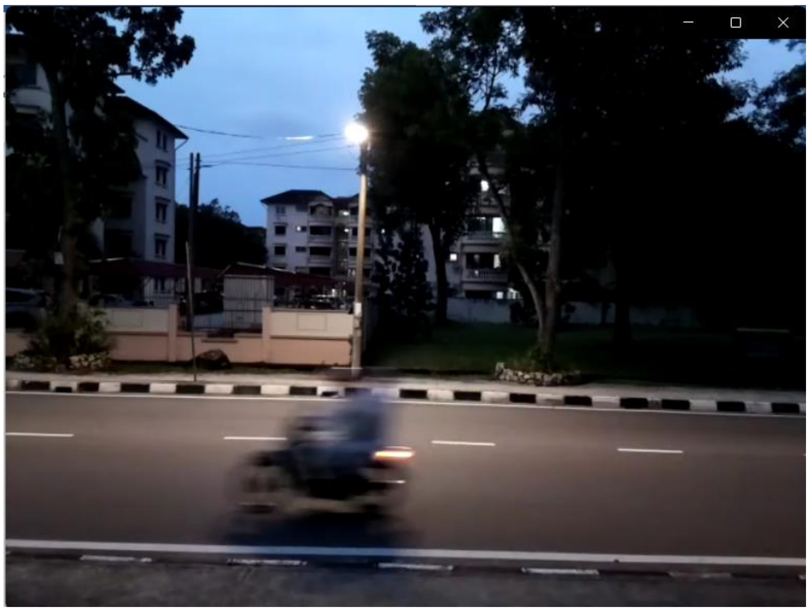


Figure 4. 25: Scene recorded in the first night

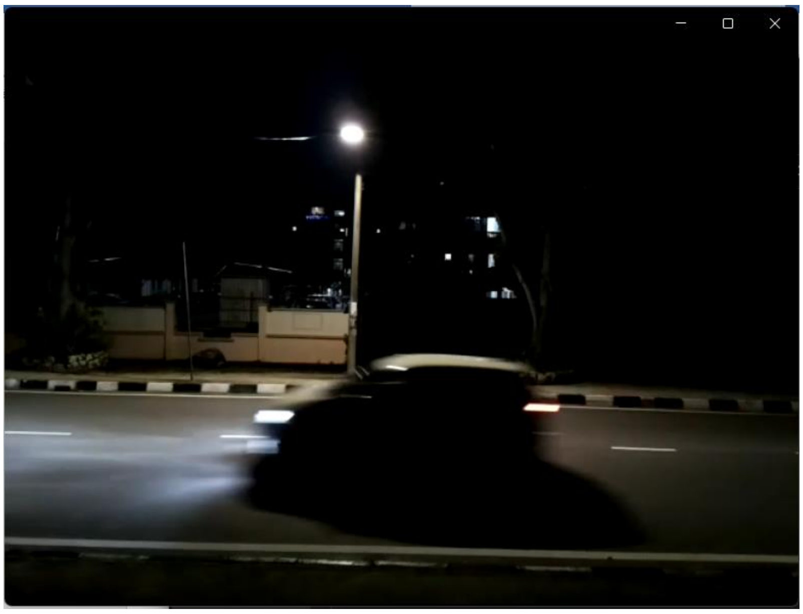


Figure 4. 26: Scene recorded in the second night

The scene recorded in the morning and afternoon is in bright condition since the sun is still out. The vehicles are very visible even in motion.

For the first night, the scene is recorded from 7pm to 7.15pm. The sun has set but there is still some light from the sky. The vehicles have lower visibility compared to the scenes recorded in the morning and afternoon. The vehicles have motion blur when moving, which would affect the classification of vehicles.

For the second night, the scene is recorded from 7.45pm to 8.00pm. The sky is totally dark. The only light source in the scene is the road lamp. The vehicles body is dark, with the headlamps and backlights visible.

4.3.2 Manual Counting results

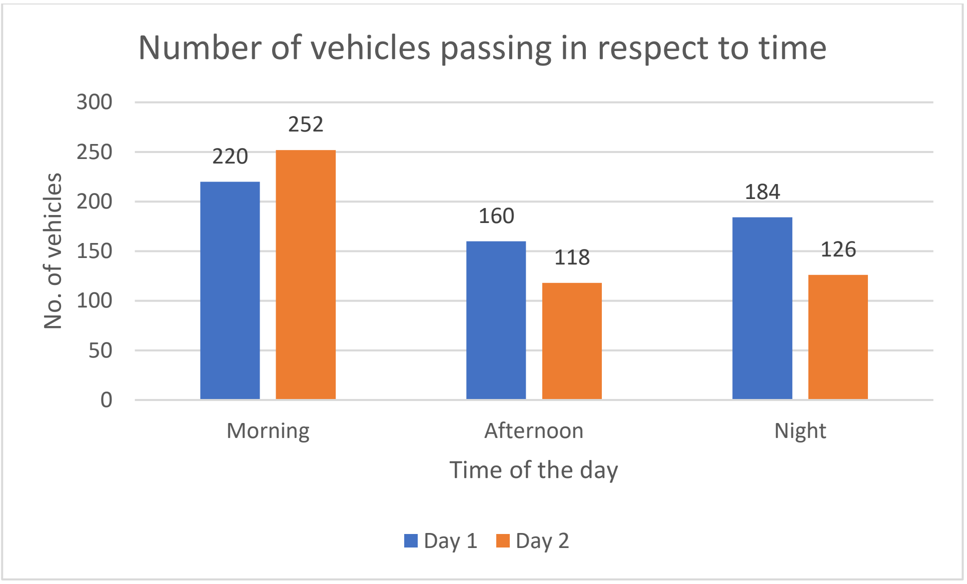


Figure 4. 27: Graph of total number of vehicles passing in respect to time (Manual Counting)

From Figure 4.27, the peak time for vehicles passing the road is during the morning. The number of vehicles at night is slightly higher than number of vehicles in the afternoon. Weekend day has less vehicles than the weekday in the morning but has more vehicles in the afternoon and at night.

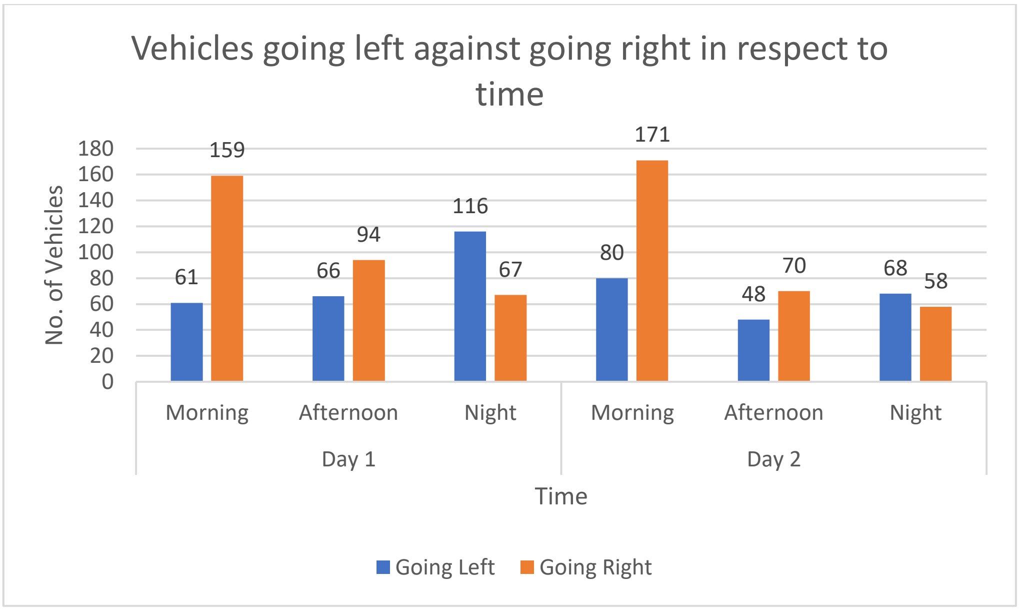


Figure 4. 28: Graph of number of vehicles going left vs going right in respect to time (Manual Counting)

The total number of vehicles passing by the road is further broken down into two categories: vehicles going to the left and vehicles going to the right. From figure 4.28, the number of vehicles going to the right is higher in the morning and afternoon on both days, while there are more vehicles going to the left during the night on both days.

4.3.3 Manual Counting vs Algorithm Counting

Table 4. 1: Total number of vehicles (manual vs algorithm)

Day	Time	Manual Counting	Algorithm counting	Accuracy (%)
1	Morning	220	235	93.18181818
1	Afternoon	252	261	96.42857
1	Night	160	161	99.375
2	Morning	118	118	100
2	Afternoon	184	194	94.56521739
2	Night	126	162	71.42857143

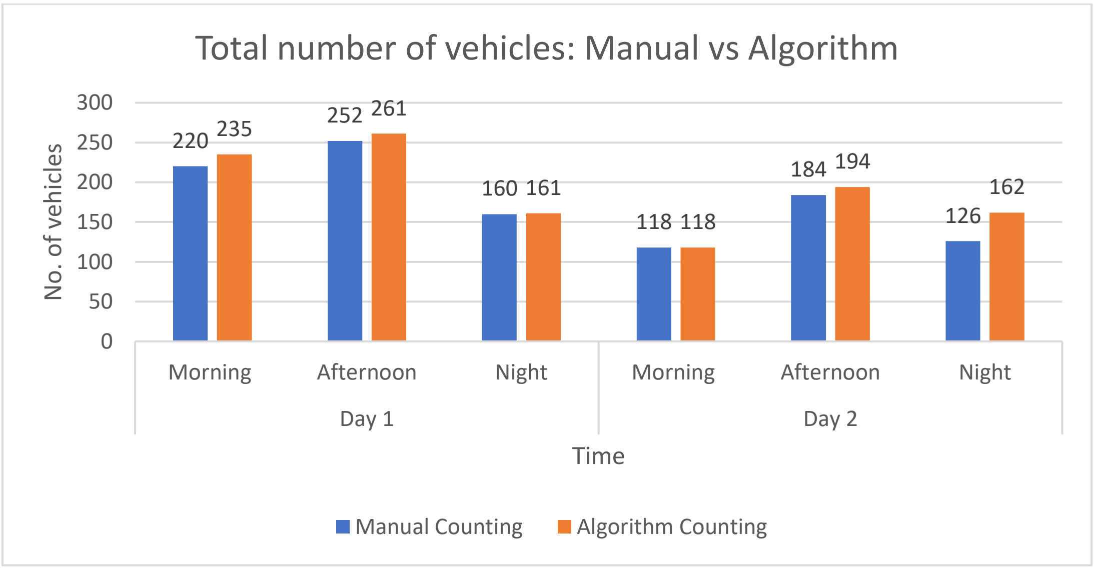


Figure 4. 29: Graph of manual counting vs algorithm counting for total number of vehicles

Table 4.1 compares the number of vehicles counted manually and counted by the algorithm and the accuracy of the algorithm. Figure 4.29 compares the number of vehicles counted manually and counted by the algorithm using a bar chart. From table 4.1, the accuracy during morning and afternoon is high from a range of 93.18% to 100%. The accuracy of the detection decreases to 71.43% during the night. The algorithm counting overcounts the total number of vehicles in most cases.

Table 4. 2: Number of vehicles going to the left (manual vs algorithm)

Day	Time	Manual Counting	Algorithm Counting	Accuracy (%)
1	Morning	61	66	91.80327869
1	Afternoon	80	81	98.75
1	Night	66	65	98.48484848
2	Morning	48	47	97.91666667
2	Afternoon	116	122	94.82758621
2	Night	68	84	76.47058824

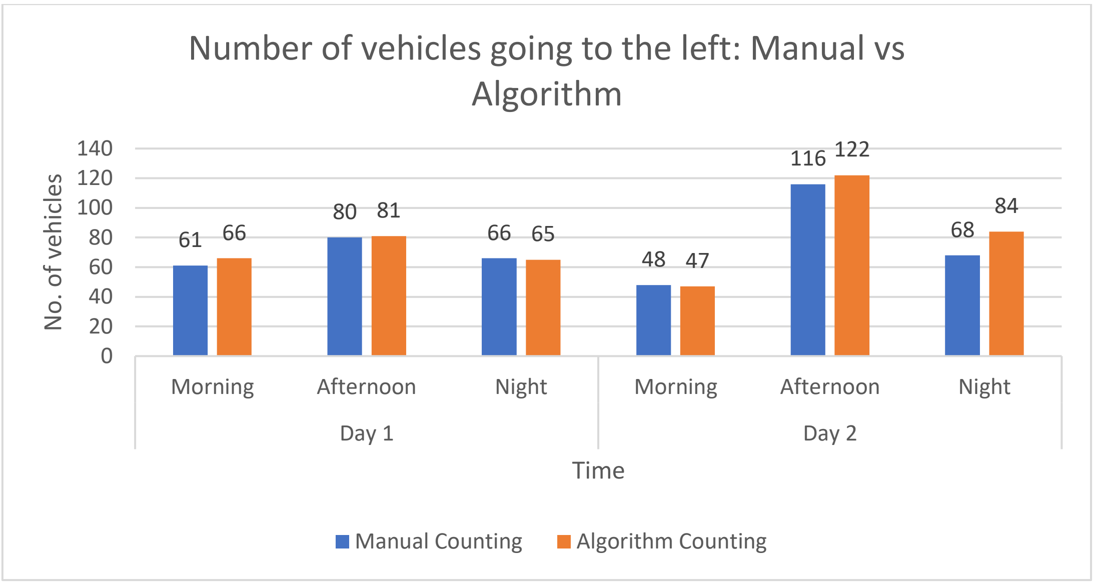


Figure 4. 30: Graph of number of vehicles going to the left (manual vs algorithm)

Table 4.2 compares the number of vehicles going to the left counted manually and counted by the algorithm and the accuracy of the algorithm. Figure 4.30 compares the number of vehicles going to the left counted manually and counted by the algorithm using a bar chart. From Figure 4.30, the algorithm overcounts 4 times and undercounts 2 times for vehicles going to the left. The accuracy for counting vehicles going to the left is at a minimum of 76.47% during the second night, while at a maximum of 98.48% on the first night.

Table 4. 3: Number of vehicles going to the right (manual vs algorithm)

Day	Time	Manual Counting	Algorithm Counting	Accuracy (%)
1	Morning	159	169	93.71069
1	Afternoon	171	180	94.73684
1	Night	94	96	97.87234
2	Morning	70	71	98.57143
2	Afternoon	67	72	92.53731
2	Night	58	78	65.51724

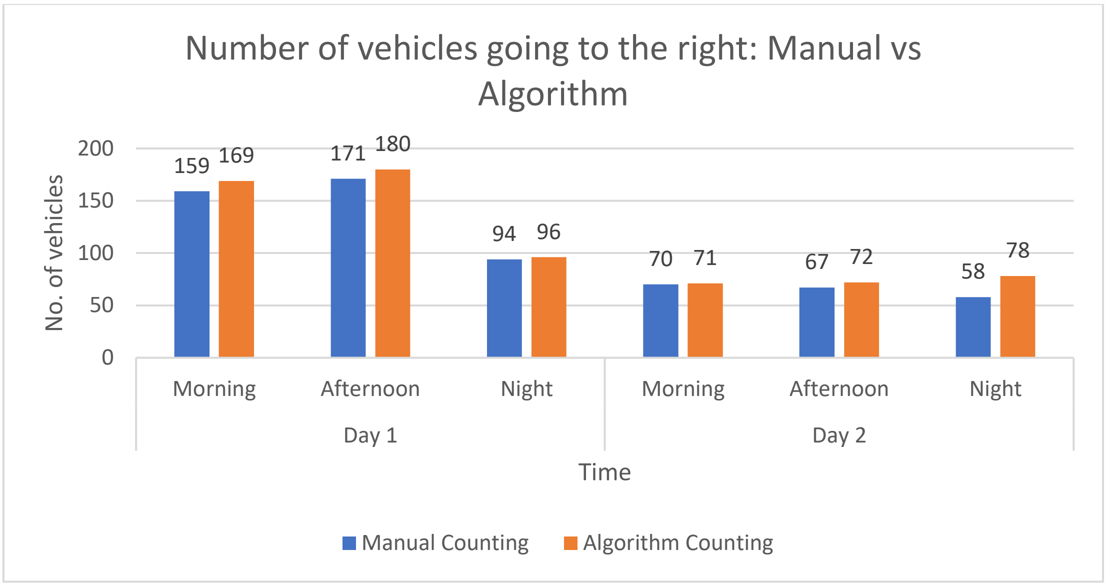


Figure 4. 31: Graph of number of vehicles going to the right (manual vs algorithm)

Table 4.3 compares the number of vehicles going to the right counted manually and counted by the algorithm and the accuracy of the algorithm. Figure 4.31 compares the number of vehicles going to the right counted manually and counted by the algorithm using a bar chart. From Figure 4.31, the algorithm overcounts vehicles going to the right for the six tests. The accuracy is maximum at 98.57% during the second morning, and minimum at 65.51% during the second night.

4.3.4 Errors Found for Vehicle Counting

4.3.4.1 Overlapping error

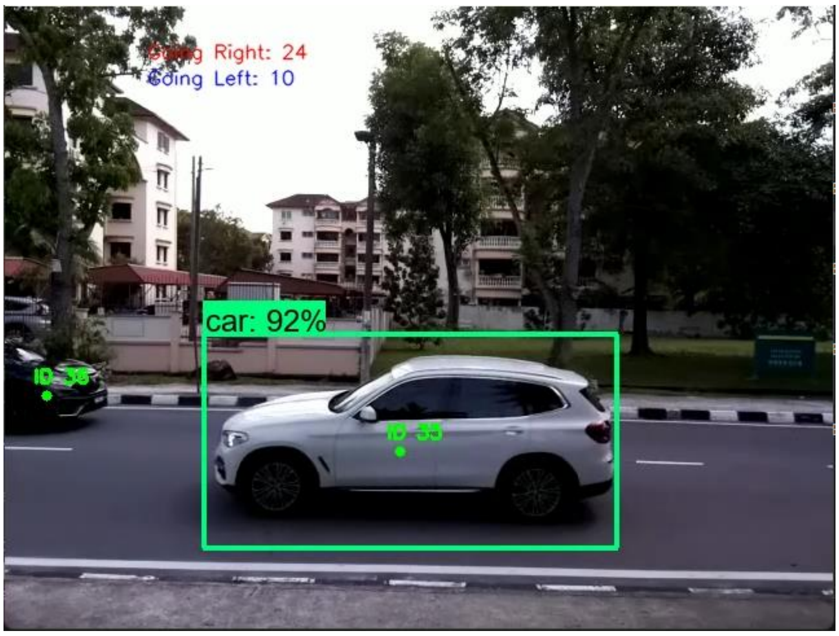


Figure 4. 32: Overlap error part 1

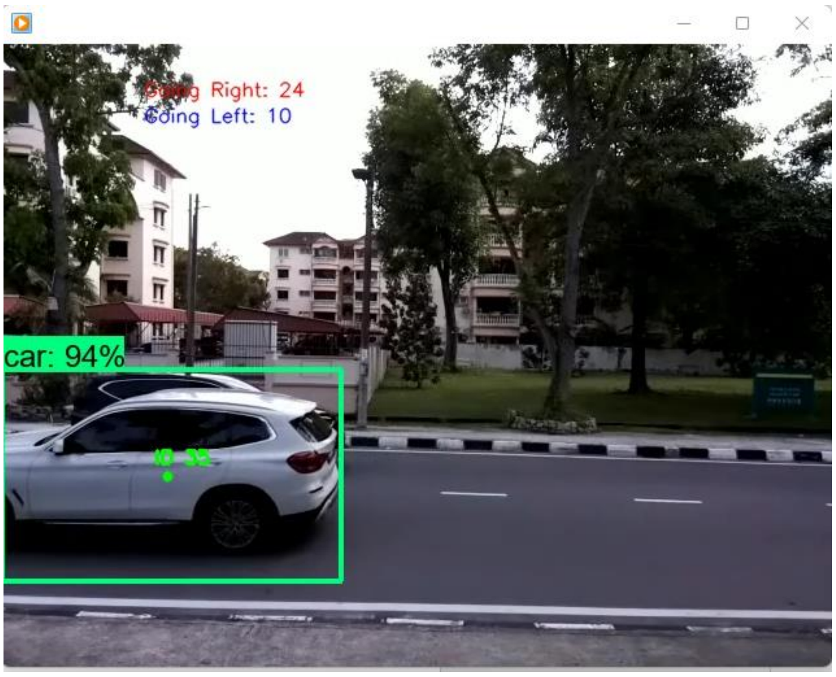


Figure 4. 33: Overlap error part 2

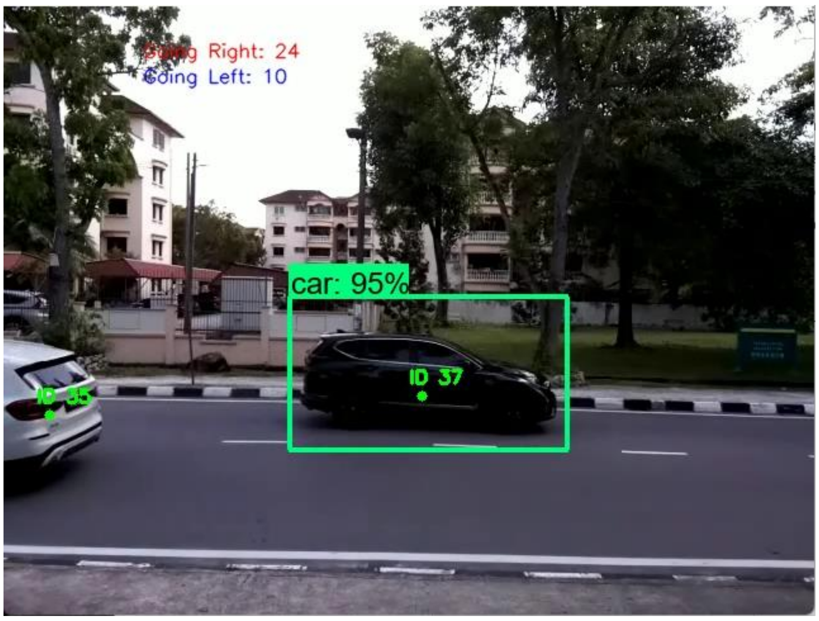


Figure 4. 34: Overlap error part 3

The main issue with placing the camera perpendicular to the road is the unavoidable overlapping error. This overlapping error is shown in Figure 4.30, 4.31 and 4.32. Vehicle 35 is going to the left while vehicle 36 is going to the right. However, the two vehicles overlap at one point, vehicle 35 blocks vehicle 36 from view. When vehicle 36 comes back to view, the algorithm registers it as a new vehicle since it suddenly appeared. This extra count happens when there are multiple vehicles in frame and the vehicle in near lane blocks the vehicle from the far lane. This confuses the tracker as it is treating the vehicle that was blocked and reappear as a new vehicle.

4.3.4.2 Detection error when scene is dark

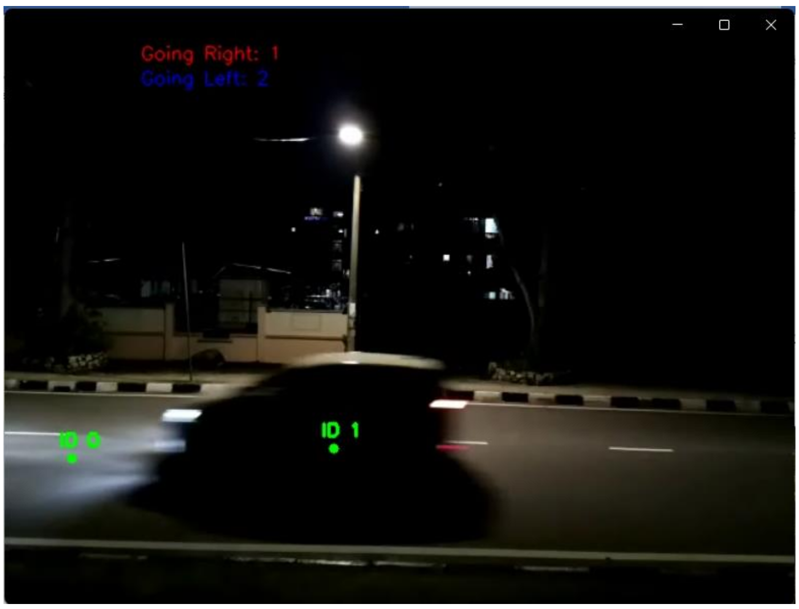


Figure 4. 35: Detection error when scene is dark

The algorithm detects the body and the light shone by the car as two different objects. The light is counted as a vehicle by the algorithm.

4.4 Vehicle Speed Measurement Evaluation

The main parameter of tuning for the speed measurement algorithm is the distance in meters for the frame width. To ensure the speed measured of each vehicle is accurate, a car is driven passing by the camera. This calibration is shown in Figure 4.36. The speed of the car can be seen on the speedometer of the car. When going to the right, the car’s speed is around 30km/h. When going to the left, the car’s speed is around 27km/h. After tuning the distance in meters of the algorithm, the algorithm can measure accurately the car’s speed.

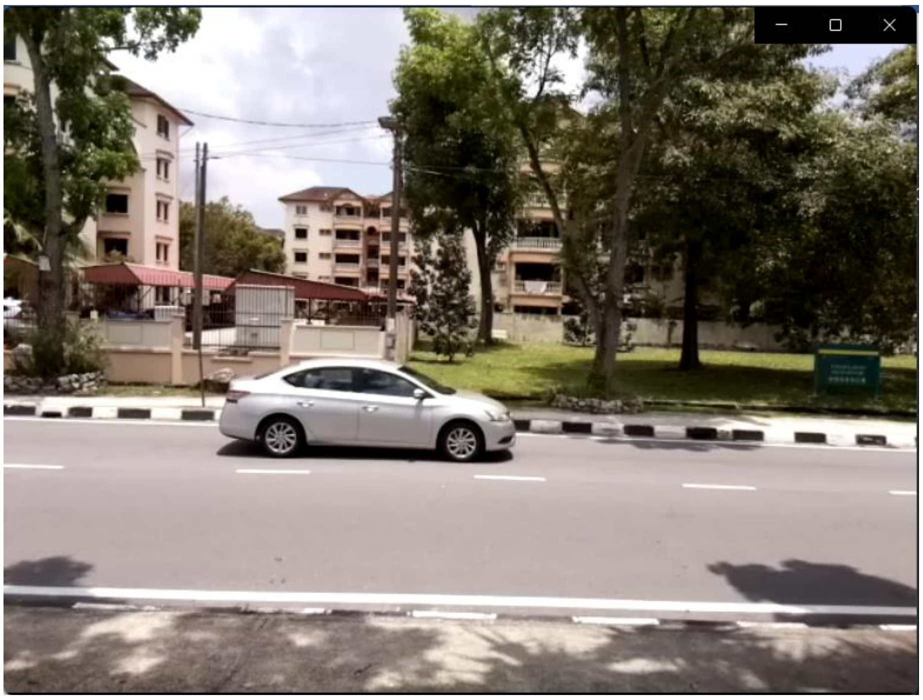


Figure 4. 36: Car with known speed driven pass the camera

Table 4. 4: Evaluation of speed measuring algorithm

Day	Time	Error Measurements	Total measurements	Error (%)
1	Morning	5	196	2.551020408
1	Afternoon	1	145	0.689655172
1	Night	4	87	4.597701149
2	Morning	4	204	1.960784314
2	Afternoon	1	118	0.847457627
2	Night	2	33	6.060606061

Table 4.4 shows the errors happening during speed measuring and the percentage of errors happening during the speed measuring. Error measurements happen when the speed of the vehicle cannot be measured, or the measured speed is above 100km/h. From the table, most errors happen in the morning. This is because there are many vehicles passing by in the morning, and this results in the overlapping

error. The algorithm tracks the vehicle, but when another vehicle overlaps with the vehicle, the algorithm tracks the other vehicle that is going to the opposite direction. The speed cannot be measured in this case. This overlapping error also results in measured speed of over 100km/h.

Overall, the error for speed measurement ranges from 0.69% to 2.55% for well-lit scenes and slightly higher at 4.60% to 6.06% for dark scenes.

4.5 Vehicle Classification Evaluation

Table 4. 5: Actual Cars vs Predicted Cars

Day	Time	Actual Cars	Predicted Cars	Accuracy (%)
1	Morning	171	156	91.22807018
1	Afternoon	121	103	96.42857
1	Night	124	77	62.09677419
2	Morning	172	163	94.76744186
2	Afternoon	85	81	95.29411765
2	Night	91	28	30.76923077

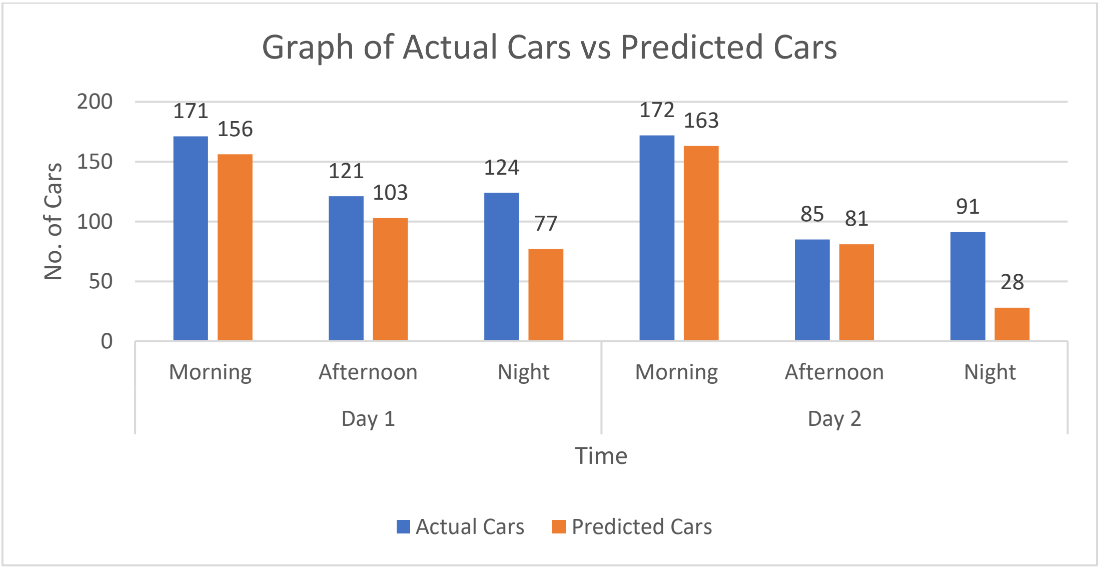


Figure 4. 37: Graph of Actual Cars vs Predicted Cars

Table 4.5 compares the actual cars in the video and the number of cars detected by the deep learning object detection model. Figure 4.37 plots the actual cars in the video and the number of cars detected by the deep learning object detection model. From Figure 4.35 and Table 4.5, the algorithm predicted cars decently during morning and afternoon, with a prediction accuracy ranging from 91.23% to 96.43%. However, the accuracy drops to 62.1% when the scene is darker in the first night. The accuracy drops drastically to 30.77% when the scene is totally dark with a single road lamp in the second night.

Table 4. 6: Actual Motorcycles vs Predicted Motorcycles

Day	Time	Actual Motorcycles	Predicted Motorcycles	Accuracy (%)
1	Morning	49	40	81.63265306
1	Afternoon	40	38	95
1	Night	58	10	17.24137931
2	Morning	57	41	71.92982456
2	Afternoon	33	33	100
2	Night	34	5	14.70588235

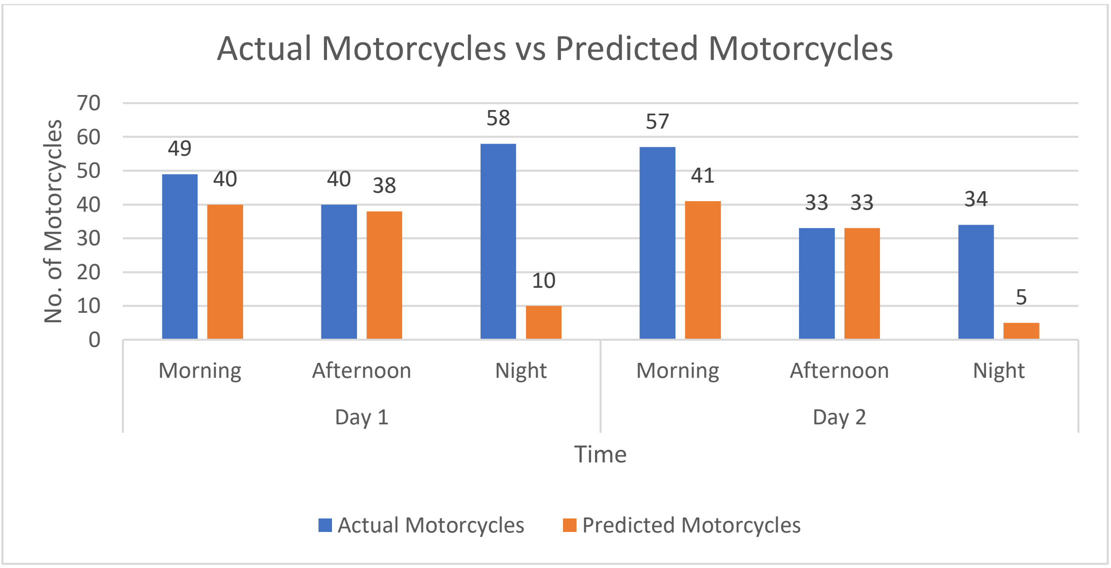


Figure 4. 38: Graph of Actual Motorcycles vs Predicted Motorcycles

Table 4.6 compares the actual number of motorcycles in the video and the number of motorcycles detected by the deep learning object detection model and shows the object detection model’s accuracy. Figure 4.38 plots the actual number of motorcycles in the video and the number of motorcycles detected by the deep learning object detection model. From Figure 4.38 and Table 4.6, the prediction accuracy for motorcycles ranges from a low 71.93% to a high 100% during the morning and afternoon when the scene is bright. The prediction accuracy for motorcycles drops to 17.24% and 14.71% when the scene is dark in the first and second night. The object detection model is trained with images of vehicles in the morning and afternoon. The visibility of the vehicles is also significantly reduced at night. Thus, vehicle classification is poor at night (dark condition).

4.6 Limitations

After conducting several experiments, some limitations in the developed system can be noticed. First, the vehicle counting, speed measuring and classification has poor performance when the scene is dark. This happens when the system is used at night after the sky has become dark.

Furthermore, when many vehicles pass by each other in the scene, the overlapping of vehicles result in extra vehicles being counted. This overlapping error also affects the vehicle speed measuring and vehicle counting function.

CHAPTER 5

CONCLUSION AND RECOMMENDATIONS

5.1 Conclusion

The objective of this project is to implement a low-cost hardware system that is able to capture video recording of vehicles on the road and a software program to be able to count, calculate the speed and categorize the vehicles in the video. The developed system can count vehicles, calculate speed of vehicles, and classify vehicles in a real-time video. The data collected is also logged into a csv file for future reference. This project aims to apply the power of computer vision and deep learning for the use of traffic data collection.

The system has been tested with 6 different video recordings recorded at different times. The number of vehicles, lighting condition of the video recordings are different. The performance of the system varies with different lighting condition. A summary for the analysis is the developed system performed well when the lighting condition is good in the morning and the afternoon. The counting and speed measurements still functions well when the scene is slightly dark but there is still light in the sky. However, the performance of vehicle counting, and speed measurements is poor when the scene is almost totally dark. The vehicle classification function also falls short when the lighting condition is dark at night with an accuracy of 30.77%.

5.2 Recommendation

For vehicle counting, the overlapping error is due to the camera placement being perpendicular to the road. A better angle for the vehicle counting will be a top view of the road, where every vehicle is able to be detected without any overlapping. By having this view the accuracy of the vehicle counting can be increased.

For vehicle counting, speed measuring, vehicle classification during bad lighting conditions, an infrared camera can be used instead of a normal camera to capture video of the vehicles. Infrared camera can capture the invisible infrared light that is emitted from objects. Therefore, IR cameras is able to see objects in the dark clearly (Cremins, 2017).

Besides, the neural network can be trained with more datasets to make it more accurate, and able to classify vehicles in different conditions. Future work can also focus on classifying vehicles into more specific classes such as trucks, vans, buses and others.

Other than that, future work can focus on testing the system in different weather such as raining and snowing to test the viability of the system in different environments.

REFERENCES

Adebisi, O., 1987. Improving Manual Counts on turning Traffic Volumes at Road Junctions. *Journal of Transportation Engineering-ASCE,* 113(3), pp. 256-267.

Analytics Vidhya, 2017. *Transfer learning and the art of using pre-trained models in deep learning.* [Online]
Available at: https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/
[Accessed 5 April 2022].

Baker, H. & Wallace, D., 1982. Software Technology gets Rid of the TIC in Manual Traffic. *ITE Journal - Institute of Transportation Engineers,* 52(4), pp. 30-31.

Batenko, A. et al., 2011. Weight-in-motion (WIM) measurements by fiber optic sensor: problems and solutions.. *Transport and Telecommunication,* 12(4), pp. 27-33.

Brownlee, J., 2021. *A Gentle Introduction to the Challenge of Training Deep Learning Neural Network Models.* [Online]
Available at: https://machinelearningmastery.com/a-gentle-introduction-to-the-challenge-of-training-deep-learning-neural-network-models/
[Accessed 5 August 2021].

Cremins, D., 2017. *Need a Night Vision Camera? Why Dynamic Smart IR is the Way to Go.* [Online]
Available at: https://www.marchnetworks.com/intelligent-ip-video-blog/need-a-night-vision-camera-why-dynamic-smart-ir-is-the-way-to-go/#:~:text=IR%20or%20night%20vision%20cameras,to%20see%20in%20the%20dark.
[Accessed 22 April 2022].

Findley, D. J., 2011. Comparison of mobile and manual data collection for roadway components. *Transportation Research Part C,* 19(3), pp. 521-540.

Godbehere, A. B., Matsukawa, A. & Goldberg, K., 2012. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. *2012 American Control Conference (ACC),* pp. 4305-4312.

IBM, 2021. *What is Computer Vision.* [Online] Available at: https://www.ibm.com/topics/computer-vision#:~:text=Computer%20vision%20is%20a%20field,recommendations%20based%20on%20that%20information.https://www.ibm.com/topics/computer-vision#:~:text=Computer%20vision%20is%20a%20field,recommendations%20based%20on%20th
[Accessed 24 June 2021].

Jalihal, S., Reddy, T. & Nataraju, J., 2005. Evaluation of automatic traffic counters under mixed traffic. *Journal of the Institution of Engineers (India): Civil Engineering Division,* Volume 86, pp. 96-102.

Le, J., 2018. *The 5 Computer Vision Techniques That Will Change How You See The World.* [Online]
Available at: https://heartbeat.fritz.ai/the-5-computer-vision-techniques-that-will-change-how-you-see-the-world-1ee19334354b
[Accessed 24 June 2021].

Li, D., Liang, B. & Zhang, W., 2014. *Real-time moving vehicle detection, tracking, and counting system implemented with OpenCV.* s.l., 2014 4th IEEE International Conference on Information Science and Technology.

Luca Massaron, A. B. A. G. A. T. R. S., 2018. *TensorFlow Deep Learning Projects.* 1st ed. Birmingham-Mumbai: Packt Publishing.

Marcomini, L. A. & Cunha, A. L., 2018. A comparison between background modelling methods for vehicle segmentation. *arVix preprint arXiv:1810.02835.*

Markoff, J., 2015. *A learning advance in artificial intelligence rivals human abilities..* [Online]
Available at: https://www.nytimes.com/2015/12/11/science/an-advance-in-artificial-intelligence-rivals-human-vision-abilities.htmlhttps://www.nytimes.com/2015/12/11/science/an-advance-in-artificial-intelligence-rivals-human-vision-abilities.htmlv
[Accessed 25 June 2021].

Marr, B., 2018. *What Is Deep Learning AI? A Simple Guide With 8 Practical Examples.* [Online]
Available at: https://www.forbes.com/sites/bernardmarr/2018/10/01/what-is-deep-learning-ai-a-simple-guide-with-8-practical-examples/?sh=710579128d4b
[Accessed 25 June 2021].

Meel, V., 2021. *YOLOv3: Real-Time Object Detection Algorithm (What's New?).* [Online]
Available at: https://viso.ai/deep-learning/yolov3-overview/
[Accessed 15 August 2021].

Murzovva, A., 2021. *Background Subtraction with OpenCV and BGS Libraries.* [Online]
Available at: https://learnopencv.com/background-subtraction-with-opencv-and-bgs-libraries/#basics
[Accessed 15 July 2021].

Nelson, J., 2020. *LabelImg forLabeling Object Detection Data.* [Online]
Available at: https://blog.roboflow.com/labelimg/
[Accessed 4 March 2022].

Nielsen, 2014. *Rising Middle Class Will Drive Global Automotive Demand in the Coming Two Years.* [Online]
Available at: https://www.nielsen.com/my/en/press-releases/2014/rising-middle-class-will-drive-global-automotive-demand/
[Accessed 21 June 2021].

OpenCV, 2016. *Background Subtraction.* [Online] Available at: https://docs.opencv.org/3.2.0/db/d5c/tutorial_py_bg_subtraction.html [Accessed 5 August 2021].

Ozer, J., 2012. *What is H.264?.* [Online] Available at: https://www.streamingmedia.com/Articles/ReadArticle.aspx?ArticleID=74735 [Accessed 28 July 2021].

Rey, J., 2017. *Faster R-CNN: Down the rabbit hole of modern object detection.* [Online] Available at: https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/ [Accessed 5 August 2021].

Rosebrock, A., 2019. *OpenCV Vehicle Detection, Tracking, and Speed Estimation.* [Online] Available at: https://pyimagesearch.com/2019/12/02/opencv-vehicle-detection-tracking-and-speed-estimation/ [Accessed 5 April 2022].

Schumann, R., 2001. Summary of Transportation Operations Data Issues. *PBS&J.*

Shaikh, R., 2020. *OpenCV (findContours) Detailed Guide.* [Online] Available at: https://medium.com/analytics-vidhya/opencv-findcontours-detailed-guide-692ee19eeb18 [Accessed 4 April 2022].

Silva, E. A. B. d. & Mendonca, G. V., 2005. Digirtal Image Processing . In: W. Chen, ed. *The Electrical Engineering Handbook.* Boston: Elsevier Academic Press, p. 891.

Stofan, D., 2020. *GoodVision Vs Manual Traffic Counters: Comparing User Experience on Traffic Surveys.* [Online] Available at: https://medium.com/goodvision/we-have-compared-user-experience-in-traffic-data-collection-between-goodvision-and-manual-traffic-79e9c4cbff09https://medium.com/goodvision/we-have-compared-user-experience-in-traffic-data-collection-between-goodvision-and-ma [Accessed 24 June 2021].

Tangkocharoen, T. & Srisuphab, A., 2017. *Vehicle detection on a pint-sized computer.* s.l., 9th International Conference on Knowledge and Smart Technology (KST).

TensorFlow, 2022. *Why TensorFlow.* [Online] Available at: https://www.tensorflow.org/about [Accessed 20 March 2022].

Tolotra Samuel, 2018. *What is the difference between step, batch size, epoch, iteration? Machine Learning Terminology.* [Online] Available at: https://tolotra.com/2018/07/25/what-is-the-difference-between-step-batch-size-epoch-iteration-machine-learning-terminology/ [Accessed 5 April 2022].

Wylie, M., 2010. Automating the collection of turning count data at signalised intersections in. *Traffic Engineering and Control,* 51(11), pp. 429-431.

Yatskiv, I., Grakovski, A. & Yurshevich, E., 2013. An overview of different methods available to observe traffic flows using new technologies. *Proceedings of the NTTS (New Techniques and Technologies for Statistics) International Conference, Eurostat.*

Zheng, P. & Mike, M., 2012. An investigation on the manual traffic count accuracy. *Procedia-Social and Behavorial Sciences,* Volume 43, pp. 226-231.

APPENDICES

APPENDIX A: Coding

Vehicle Counting Program

```
from locale import format_string
import cv2
import numpy as np
import time
import vehicles
import csv
import tensorflow as tf
import dlib
import vehicles
import centroidtracker
from datetime import datetime
import os
from imutils.video import FPS
import imutils
import six

from centroidtracker import CentroidTracker
from vehicles import TrackableObject
from tensorflow_detection import DetectionObj

from object_detection.utils import label_map_util
from object_detection.utils import ops as utils_ops
from object_detection.utils import visualization_utils_modded as
viz_utils
from object_detection.builders import model_builder
from object_detection.utils import config_util

"""
Object Detection
"""
paths = {'CHECKPOINT_PATH':"Tensorflow/workspace/models/my_ssd_mobnet"}
```

```
files = {
    'PIPELINE_CONFIG':
"Tensorflow/workspace/models/my_ssd_mobnet/pipeline.config",
    'TF_RECORD_SCRIPT': "Tensorflow/scripts/generate_tfrecord.py",
    'LABELMAP': "Tensorflow/workspace/annotations/label_map.pbtxt"
}
# Load pipeline config and build a detection model
configs =
config_util.get_configs_from_pipeline_file(files['PIPELINE_CONFIG'])

detection_model = model_builder.build(model_config=configs['model'],
is_training=False)

# Restore checkpoint
ckpt = tf.compat.v2.train.Checkpoint(model=detection_model)
ckpt.restore(os.path.join(paths['CHECKPOINT_PATH'], 'ckpt-
6')).expect_partial()

category_index =
label_map_util.create_category_index_from_labelmap(files['LABELMAP'])



def vehicle_counting():
    frame_width = 640
    frame_height = 480
    videoSource = "C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FYP
Project\\Code\\VideoSource\\Morning1.mp4"
    videoName= videoSource[64:-4]
    videoOut = f'C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FYP
Project\\Code\\Result\\{videoName}_detection.mp4'
    LogOutPath = 'C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FYP
Project\\Code\\Result'
    csv_name = f"{videoName}.csv"
    cap=cv2.VideoCapture(videoSource)     # Video Source
    out = cv2.VideoWriter(videoOut,cv2.VideoWriter_fourcc(*'mp4v'), 25,
(frame_width,frame_height)) #Write video to output file
    fgbg=cv2.createBackgroundSubtractorMOG2(detectShadows=False,history
=200,varThreshold = 90)       # Create Foreground mask
    kernalOp = np.ones((3,3),np.uint8)
    kernalOp2 = np.ones((5,5),np.uint8)
    kernalCl = np.ones((11,11),np.uint8)
    font = cv2.FONT_HERSHEY_SIMPLEX
    cars = []
    max_p_age = 5
    pid = 1
    cnt_left=0
    cnt_right=0
    cnt_left2=0
    cnt_right2=0
    cnt_car = 0
    cnt_motor = 0
    cnt_bicycle = 0
    cnt_nan = 0

    # maximum consecutive frames a given object is allowed to be marked
as "disappeared" until we need to deregister the object from tracking
    max_disappear = 8
    # maximum distance between centroids to associate an object if the
distance is larger than this maximum distance we'll start to mark the
object as "disappeared"
    max_distance = 175
    #number of frames to perform object tracking instead of object
detection
    track_object = 4
    #minimum confidence
    confidence = 0.4
    #frame width in pixels
    frame_width = 640
    #dictionary holding the different speed estimation columns
    speed_estimation_zone = (250,300, 350, 400)
    #real world distance in meters
    distance_left = 13
    distance_right = 15.1
    #speed limit in kmph
    speed_limit = 50

    #Meter Per Pixel
    meterPerPixel_left = distance_left / frame_width
    meterPerPixel_right = distance_right/ frame_width


    # count the number of frames
    frames = cap.get(cv2.CAP_PROP_FRAME_COUNT)
    # start the frames per second throughput estimator
    fps_2 = int(cap.get(cv2.CAP_PROP_FPS))

    # calculate dusration of the video
    seconds = int(frames / fps_2)
    duration = float(seconds/3600)

    print(f"frames: {frames} fps: {fps_2} seconds:{seconds}
duration:{duration}")
    print("Car counting and classification")

    line_left=310 #320 dv
    line_right=330 #450 dv
    left_limit=540 #470 for diagonal view (dv)
    right_limit=60 #300 for diagonal view

    # ---------------------------------- Speed Measurement Parameters -
--------------------------------------
    ct = CentroidTracker(maxDisappeared= max_disappear)
    trackers = []
    trackableObjects = {}
    # keep the count of total number of frames
    frame_count = 0
    # initialize the log file
    logFile = None
    # initialize the list of various points used to calculate the avg
of
    # the vehicle speed
    points = [("A", "B"), ("B", "C"), ("C", "D")]



    while(cap.isOpened()):
        timez = float(frame_count/fps_2)
        ret,frame=cap.read()
        ts = datetime.now()
        newDate = ts.strftime("%m-%d-%y")
        rects = []
        centroidz = []




        if frame is None:
            break

        if logFile is None:
            # build the log file path and create/open the log file
            logPath = os.path.join(LogOutPath, csv_name)
            logFile = open(logPath, mode="a")
            # set the file pointer to end of the file
            pos = logFile.seek(0, os.SEEK_END)
            # if we are using dropbox and this is a empty log file then
            # write the column headings

            if pos == 0:
                logFile.write("Year,Month,Day,Time,
ObjectID ,Direction, Vehicle Class, Speed(km/h)\n")
```

```
        # Object Detection
        image_np = np.array(frame)

        input_tensor = tf.convert_to_tensor(np.expand_dims(image_np,
axis=0), dtype=tf.float32)
        detections = detect_fn(input_tensor)

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

        # detection_classes should be ints.
        detections['detection_classes'] =
detections['detection_classes'].astype(np.int64)

        label_id_offset = 1
        image_np_with_detections = image_np.copy()

        class_label, rects2, _ =
viz_utils.visualize_boxes_and_labels_on_image_array(
                    image_np_with_detections,
                    detections['detection_boxes'],
                    detections['detection_classes']+label_id_offset,
                    detections['detection_scores'],
                    category_index,
                    use_normalized_coordinates=True,
                    max_boxes_to_draw=5,
                    min_score_thresh=.8,
                    agnostic_mode=False)

        image_np_with_detections = cv2.resize(image_np_with_detections,
(frame_width, frame_height))


        for i in cars:
            i.age_one()
        fgmask=fgbg.apply(frame)
        # if frame_count == 1080:
        #     cv2.imwrite(f'C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FY
P Project\\Code\\Result\\maskSub_{frame_count}.png',fgmask)






        if ret==True:
            ret,imBin=cv2.threshold(fgmask,200,255,cv2.THRESH_BINARY)
            #
cv2.imwrite(f'C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FYP
Project\\Code\\Result\\image_thresh_{frame_count}.png',imBin)
            mask = cv2.morphologyEx(imBin, cv2.MORPH_OPEN, kernalOp)
            #
cv2.imwrite(f'C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FYP
Project\\Code\\Result\\image_morphOp_{frame_count}.png',mask)
            mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernalCl)
            #
cv2.imwrite(f'C:\\Users\\anand\\Documents\\UTAR\\Y3S1\\FYP
Project\\Code\\Result\\image_morphClose_{frame_count}.png',mask)


            (countours0,hierarchy)=cv2.findContours(mask,cv2.RETR_EXTER
NAL,cv2.CHAIN_APPROX_NONE)
            for cnt in countours0:
                area=cv2.contourArea(cnt)


                if (area>2000 and area < 60000):

                    m=cv2.moments(cnt)
                    cx=int(m['m10']/m['m00'])
                    cy=int(m['m01']/m['m00'])
                    x,y,w,h=cv2.boundingRect(cnt)

                    rects.append((x, y, w, h))
                    centroidz.append((cx,cy))
                    # add the bounding box coordinates to the
rectangles list


                    # use the centroid tracker to associate the (1) old
object
                    # centroids with (2) the newly computed object
centroids

                    new=True
                    if cx in range(right_limit,left_limit): # If car
within limit
                        for i in cars:
                            if abs(x - i.getX()) <= w and  abs(y -
i.getY()) <= h :  #Check whether new car or old car that has moved
                                new = False
                                i.updateCoords(cx, cy) #Update
Coordinate if car moves

                                # Determine the direction of the car
                                if
i.going_LEFT(line_right,line_left)==True:
                                    cnt_left+=1

                                    if to.vehicleClass == "car":
                                        cnt_car+=1
                                    elif to.vehicleClass
=="motorcycle":
                                        cnt_motor+=1
                                    elif to.vehicleClass =="bicycle":
                                        cnt_bicycle+=1
                                    else:
                                        cnt_nan+=1

                                elif
i.going_RIGHT(line_right,line_left)==True:
                                    cnt_right+=1

                                    if to.vehicleClass == "car":
                                        cnt_car+=1
                                    elif to.vehicleClass
=="motorcycle":
                                        cnt_motor+=1
                                    elif to.vehicleClass =="bicycle":
                                        cnt_bicycle+=1
                                    else:
                                        cnt_nan+=1

                                break

                            # If reach limit stop bounding rectangle
                            if i.getState()=='1':
                                if i.getDir()=='right'and
i.getX()>right_limit:
                                    i.setDone()
                                elif i.getDir()=='left'and
i.getX()<left_limit:
                                    i.setDone()
                            if i.timedOut():
                                index=cars.index(i)
                                cars.pop(index)
                                del i

                        if new==True:
                            p=vehicles.Car(pid,cx,cy,max_p_age)
                            cars.append(p)
                            pid+1
                    cv2.circle(image_np_with_detections, (cx, cy), 2,
(0, 0, 255), -1) # Draw the centroid of the car
```

```
                    img=cv2.rectangle(mask,(x,y),(x+w,y+h),(244,255,100
),2) # Draw rectangle over the car

                    if frame_count == 1080:
                        frame=cv2.line(frame,(line_left,0),(line_left,4
80),(0,0,255),3,8)
                        frame=cv2.line(frame,(left_limit,0),(left_limit
,480),(255,255,0),1,8) # Display left limit

                        frame=cv2.line(frame,(right_limit,0),(right_lim
it,480),(255,255,0),1,8) # Display right limit
                        frame = cv2.line(frame, (line_right, 0),
(line_right, 480), (255, 0,0), 3, 8)
                        cv2.imwrite(f'C:\\Users\\anand\\Documents\\UTAR
\\Y3S1\\FYP
Project\\Code\\Result\\imageFindcontour_{frame_count}.png',frame)




            objects = ct.update(rects, centroidz)


            # loop over the tracked objects
            for (objectID, centroid) in objects.items():
                # check to see if a trackable object exists for the
current
                # object ID
                to = trackableObjects.get(objectID, None)
                # print(f"Object ID:{objectID} centroid:{centroid}" )

                # if there is no existing trackable object, create one
                if to is None:
                    to = TrackableObject(objectID, centroid)
                    to.vehicleClass = class_label






                # otherwise, if there is a trackable object and its
speed has
                # not yet been estimated then estimate it
                elif not to.estimated:
                    if to.vehicleClass is None or to.vehicleClass ==
"None":
                        to.vehicleClass = class_label
                    # check if the direction of the object has been
set, if
                    # not, calculate it, and set it
                    if to.direction is None or to.direction == 0:
                        y = [c[0] for c in to.centroids]
                        direction = centroid[0] - np.mean(y)
                        to.direction = direction

                        # print(f"npmeany: {np.mean(y)} direction:
{direction}")

                            # if the direction is positive (indicating
the object
                    # is moving from left to right)
                    if to.direction > 0:
                        # check to see if timestamp has been noted for
                        # point A
                        if to.timestamp["A"] == 0 :
                            # if the centroid's x-coordinate is greater
than

                            # the corresponding point then set the
timestamp

                            # as current timestamp and set the position
as the

                            # centroid's x-coordinate
                            if centroid[0] > speed_estimation_zone[0]:
                                to.timestamp["A"] = timez
                                to.position["A"] = centroid[0]
                        # check to see if timestamp has been noted for
                        # point B
                        elif to.timestamp["B"] == 0:
                            # if the centroid's x-coordinate is greater
than

                            # the corresponding point then set the
timestamp

                            # as current timestamp and set the position
as the

                            # centroid's x-coordinate
                            if centroid[0] > speed_estimation_zone[1]:
                                to.timestamp["B"] = timez
                                to.position["B"] = centroid[0]
                        # check to see if timestamp has been noted for
                        # point C
                        elif to.timestamp["C"] == 0:
                            # if the centroid's x-coordinate is greater
than

                            # the corresponding point then set the
timestamp
```

```
                            # as current timestamp and set the position
as the
                            # centroid's x-coordinate
                            if centroid[0] > speed_estimation_zone[2]:
                                to.timestamp["C"] = timez
                                to.position["C"] = centroid[0]
                        # check to see if timestamp has been noted for
                        # point D
                        elif to.timestamp["D"] == 0:
                            # if the centroid's x-coordinate is greater
than
                            # the corresponding point then set the
timestamp
                            # as current timestamp, set the position as
the
                            # centroid's x-coordinate, and set the last
point
                            # flag as True
                            if centroid[0] > speed_estimation_zone[3]:
                                to.timestamp["D"] = timez
                                to.position["D"] = centroid[0]
                                to.lastPoint = True

                    # if the direction is negative (indicating the
object
                    # is moving from right to left)
                    elif to.direction < 0:
                        # check to see if timestamp has been noted for
                        # point D
                        if to.timestamp["D"] == 0 :
                            # if the centroid's x-coordinate is lesser
than
                            # the corresponding point then set the
timestamp
                            # as current timestamp and set the position
as the
                            # centroid's x-coordinate
                            if centroid[0] < speed_estimation_zone[0]:
                                to.timestamp["D"] = timez
                                to.position["D"] = centroid[0]
                        # check to see if timestamp has been noted for
                        # point C
                        elif to.timestamp["C"] == 0:
                            # if the centroid's x-coordinate is lesser
than
                            # the corresponding point then set the
timestamp
                            # as current timestamp and set the position
as the
```

```
                                    # centroid's x-coordinate
                                    if centroid[0] < speed_estimation_zone[1]:
                                        to.timestamp["C"] = timez
                                        to.position["C"] = centroid[0]
                                # check to see if timestamp has been noted for
                                # point B
                                elif to.timestamp["B"] == 0:
                                    # if the centroid's x-coordinate is lesser
than
                                    # the corresponding point then set the
timestamp
                                    # as current timestamp and set the position
as the
                                    # centroid's x-coordinate
                                    if centroid[0] < speed_estimation_zone[2]:
                                        to.timestamp["B"] = timez
                                        to.position["B"] = centroid[0]
                                # check to see if timestamp has been noted for
                                # point A
                                elif to.timestamp["A"] == 0:
                                    # if the centroid's x-coordinate is lesser
than
                                    # the corresponding point then set the
timestamp
                                    # as current timestamp, set the position as
the
                                    # centroid's x-coordinate, and set the last
point
                                    # flag as True
                                    if centroid[0] < speed_estimation_zone[3]:
                                        to.timestamp["A"] = timez
                                        to.position["A"] = centroid[0]
                                        to.lastPoint = True

                            # check to see if the vehicle is past the last
point and
                            # the vehicle's speed has not yet been estimated,
if yes,
                            # then calculate the vehicle speed and log it if
it's
                            # over the limit
                            if to.lastPoint and not to.estimated:
                                # print(to.position["A"], to.position["B"],
to.position["C"], to.position["D"])
                                # initialize the list of estimated speeds
                                estimatedSpeeds = []

                                if to.vehicleClass is None or to.vehicleClass
== "None":
```

```
                            # if class_label is None:
                            #     to.vehicleClass = "motorcycle"
                            #     cnt_motor += 1
                            # else:
                            to.vehicleClass = class_label



                        # loop over all the pairs of points and
estimate the
                        # vehicle speed
                        for (i, j) in points:
                            # calculate the distance in pixels
                            d = to.position[j] - to.position[i]
                            distanceInPixels = abs(d)
                            # print(f"Distance In Pixel:
{distanceInPixels}")
                            # check if the distance in pixels is zero,
if so,
                            # skip this iteration
                            if distanceInPixels == 0:
                                continue
                            # calculate the time in hours
                            if to.timestamp[j] is str or
to.timestamp[i] is str:
                                estimatedSpeeds.append(100)
                            else:
                                # print(f"Timestamp J:
{type(to.timestamp[j])} Timestamp I: {type(to.timestamp[i])}")
                                timeInSeconds = abs(to.timestamp[j] -
to.timestamp[i])

                                # timeInSeconds =
abs(t.total_seconds())
                                timeInHours = timeInSeconds / (60 * 60)
                                # calculate distance in kilometers and
append the
                                # calculated speed to the list
                                if direction > 0:
                                    distanceInMeters = distanceInPixels
* meterPerPixel_right

                                elif direction < 0:
                                    distanceInMeters = distanceInPixels
* meterPerPixel_left


                                distanceInKM = distanceInMeters / 1000
                                estimatedSpeeds.append(distanceInKM /
timeInHours)
```

```
                            # else:
                            #     estimatedSpeeds.append(100)
                        # calculate the average speed
                        to.calculate_speed(estimatedSpeeds)
                        # set the object as estimated
                        to.estimated = True
                        print(to.vehicleClass)
                        print("[INFO] Speed of the vehicle that just 
passed"\
                            " is: {:.2f} KMPH {} 
{}".format(to.speedKMPH, objectID, to.direction))
                        # textz = "Speed: {:.2f}".format(to.speedKMPH)
                        # cv2.putText(image_np_with_detections, textz, 
(centroid[0] - 15, centroid[1] - 10)
                        # , cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 
2)

                # store the trackable object in our dictionary
                trackableObjects[objectID] = to

                # draw both the ID of the object and the centroid of 
the
                # object on the output frame
                text = "ID {}".format(objectID)
                cv2.putText(image_np_with_detections, text, 
(centroid[0] - 10, centroid[1] - 10)
                    , cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
                cv2.circle(image_np_with_detections, (centroid[0], 
centroid[1]), 4,
                    (0, 255, 0), -1)

                # check if the object has not been logged
                if not to.logged:
                    # check if the object's speed has been estimated 
and it
                    # is higher than the speed limit
                    if to.estimated:
                        # set the current year, month, day, and time
                        year = ts.strftime("%Y")
                        month = ts.strftime("%m")
                        day = ts.strftime("%d")
                        time = ts.strftime("%H:%M:%S")
                        if to.direction < 0:
                            cardirection = "Left"
                            cnt_left2 +=1
                        elif to.direction > 0:
                            cardirection = "Right"
                            cnt_right2+=1
                        # if to.vehicleClass == "car":
                        #     cnt_car+=1
                        # elif to.vehicleClass =="motorcycle":
                        #     cnt_motor+=1
                        # elif to.vehicleClass =="bicycle":
                        #     cnt_bicycle+=1



                        # log the event in the log file
                        info =
"{},{},{},{},{},{},{},{:.2f}\n".format(year, month, day, time,
to.objectID, cardirection,
                            to.vehicleClass, to.speedKMPH)
                        print(info)
                        logFile.write(info)
                        # set the object has logged
                        to.logged = True





            #-------------------------Display info on video ----------
---------------------------------------------------

            str_left='Going Right: '+str(cnt_left)
            str_right='Going Left: '+str(cnt_right)
            cv2.putText(image_np_with_detections, str_left, (110, 40),
font, 0.5, (0, 0, 255), 1, cv2.LINE_AA)
            cv2.putText(image_np_with_detections, str_right, (110, 60),
font, 0.5, (255, 0, 0), 1, cv2.LINE_AA)

            out.write(image_np_with_detections) # Export frame to video
            cv2.imshow('object detection',  image_np_with_detections)



            #----------------When to End Video-----------------------
            frame_count += 1

            if cv2.waitKey(10)&0xff==ord('q'):
                break

        else:
            break
```

```
    cap.release()
    out.release()
    cv2.destroyAllWindows()

    if logFile is not None:
        infoz = "Total Vehicle: {},Total Left: {},Total Right: {},Total
Cars: {}, Total Motorcycles: {}, Total Bicycles: {}\n".format((cnt_left
+ cnt_right), cnt_right, cnt_left, cnt_car, cnt_motor,
cnt_bicycle)
        logFile.write(infoz)

        logFile.close()




def detect_video():
    detection = DetectionObj(model='my_ssd_mobnet')
    detection.video_pipeline(video="C:\\Users\\anand\\Documents\\UTAR\\
Y3S1\\FYP Project\\Code\\VideoSource\\video5.2.mp4", audio=False)


@tf.function
def detect_fn(image):
    image, shapes = detection_model.preprocess(image)
    prediction_dict = detection_model.predict(image, shapes)
    detections = detection_model.postprocess(prediction_dict, shapes)
    return detections


if __name__ == '__main__':
    vehicle_counting()
```

Vehicle counting, speed measuring subprogram: vehicle.py

```
from random import randint
import time
import numpy as np
from scipy.spatial import distance as dist
from collections import OrderedDict

class Car:
    tracks=[]
    def __init__(self,i,xi,yi,max_age):
        self.i=i
        self.x=xi
        self.y=yi
        self.tracks=[]
        self.done=False
        self.state='0'
        self.age=0
        self.max_age=max_age
        self.dir=None

    def getTracks(self):
        return self.tracks

    def getId(self): #For the ID
        return self.i

    def getState(self):
        return self.state

    def getDir(self):
        return self.dir

    def getX(self):  #for x coordinate
        return self.x

    def getY(self):  #for y coordinate
        return self.y

    def updateCoords(self, xn, yn):
        self.age = 0
        self.tracks.append([self.x, self.y])
        self.x = xn
        self.y = yn

    def setDone(self):
        self.done = True

    def timedOut(self):
        return self.done

    def going_LEFT(self, mid_start, mid_end):
        if len(self.tracks)>=2:
            if self.state=='0':
                if self.tracks[-1][0]>mid_end and self.tracks[-
2][0]<=mid_end: # nested listing
                    state='1'
                    self.dir='left'
                    return True
                else:
                    return False
            else:
                return False
        else:
            return False

    def going_RIGHT(self,mid_start,mid_end):
        if len(self.tracks)>=2:
            if self.state=='0':
                if self.tracks[-1][0]<mid_start and self.tracks[-
2][0]>=mid_start:
                    start='1'
                    self.dir='right'
                    return True
                else:
                    return False
            else:
                return False
        else:
            return False

    def age_one(self):
        self.age+=1
        if self.age>self.max_age:
            self.done=True
        return  True

#Class2

class MultiCar:
    def __init__(self,cars,xi,yi):
        self.cars=cars
        self.x=xi
        self.y=yi
        self.tracks=[]
        self.done=False
```

```
class TrackableObject:
    def __init__(self, objectID, centroid):
        # store the object ID, then initialize a list of centroids
        # using the current centroid
        self.objectID = objectID
        self.centroids = [centroid]
        # initialize a dictionaries to store the timestamp and
        # position of the object at various points
        self.timestamp = {"A": 0, "B": 0, "C": 0, "D": 0}
        self.position = {"A": None, "B": None, "C": None, "D": None}
        self.lastPoint = False
        # initialize the object speeds in MPH and KMPH
        self.speedMPH = None
        self.speedKMPH = None
        # initialize two booleans, (1) used to indicate if the
        # object's speed has already been estimated or not, and (2)
        # used to indidicate if the object's speed has been logged or
        # not
        self.estimated = False
        self.logged = False
        # initialize the direction of the object
        self.direction = None
        #initialize vehicle Class
        self.vehicleClass = None

    def calculate_speed(self, estimatedSpeeds):
        # calculate the speed in KMPH and MPH
        self.speedKMPH = np.average(estimatedSpeeds)
        MILES_PER_ONE_KILOMETER = 0.621371
        self.speedMPH = self.speedKMPH * MILES_PER_ONE_KILOMETER
```

Vehicle counting, speed measuring subprogram: centroidtracker.py

```python
# import the necessary packages
from scipy.spatial import distance as dist
from collections import OrderedDict
import numpy as np

class CentroidTracker():
    def __init__(self, maxDisappeared=50):
        # initialize the next unique object ID along with two ordered
        # dictionaries used to keep track of mapping a given object
        # ID to its centroid and number of consecutive frames it has
        # been marked as "disappeared", respectively
        self.nextObjectID = 0
        self.objects = OrderedDict()
        self.disappeared = OrderedDict()
        # store the number of maximum consecutive frames a given
        # object is allowed to be marked as "disappeared" until we
        # need to deregister the object from tracking
        self.maxDisappeared = maxDisappeared

    def register(self, centroid):
        # when registering an object we use the next available object
        # ID to store the centroid
        self.objects[self.nextObjectID] = centroid
        self.disappeared[self.nextObjectID] = 0
        self.nextObjectID += 1

    def deregister(self, objectID):
        # to deregister an object ID we delete the object ID from
        # both of our respective dictionaries
        del self.objects[objectID]
        del self.disappeared[objectID]

    def update(self, rects, centroidz):
        # check to see if the list of input bounding box rectangles
        # is empty
        if len(rects) == 0:
            # loop over any existing tracked objects and mark them
            # as disappeared
            for objectID in list(self.disappeared.keys()):
                self.disappeared[objectID] += 1
                # if we have reached a maximum number of consecutive
                # frames where a given object has been marked as
                # missing, deregister it
                if self.disappeared[objectID] > self.maxDisappeared:
                    self.deregister(objectID)
            # return early as there are no centroids or tracking info
            # to update
            return self.objects

        # initialize an array of input centroids for the current frame
        inputCentroids = np.zeros((len(rects), 2), dtype="int")
        # loop over the bounding box rectangles
        for (i, (startX, startY, endX, endY)) in enumerate(rects):
            # use the bounding box coordinates to derive the centroid
            # cX = int((startX + endX) / 2.0)
            # cY = int((startY + endY) / 2.0)
            cX = centroidz[i][0]
            cY = centroidz[i][1]
            inputCentroids[i] = (cX, cY)

        # if we are currently not tracking any objects take the input
        # centroids and register each of them
        if len(self.objects) == 0:
            for i in range(0, len(inputCentroids)):
                self.register(inputCentroids[i])

        # otherwise, are are currently tracking objects so we need to
        # try to match the input centroids to existing object
        # centroids
        else:
            # grab the set of object IDs and corresponding centroids
            objectIDs = list(self.objects.keys())
            objectCentroids = list(self.objects.values())
            # compute the distance between each pair of object
            # centroids and input centroids, respectively -- our
            # goal will be to match an input centroid to an existing
            # object centroid
            D = dist.cdist(np.array(objectCentroids), inputCentroids)
            # in order to perform this matching we must (1) find the
            # smallest value in each row and then (2) sort the row
            # indexes based on their minimum values so that the row
            # with the smallest value is at the *front* of the index
            # list
            rows = D.min(axis=1).argsort()
            # next, we perform a similar process on the columns by
            # finding the smallest value in each column and then
            # sorting using the previously computed row index list
            cols = D.argmin(axis=1)[rows]

            # in order to determine if we need to update, register,
            # or deregister an object we need to keep track of which
            # of the rows and column indexes we have already examined
            usedRows = set()
            usedCols = set()
            # loop over the combination of the (row, column) index
            # tuples
            for (row, col) in zip(rows, cols):
                # if we have already examined either the row or
                # column value before, ignore it
                # val
                if row in usedRows or col in usedCols:
                    continue
                # otherwise, grab the object ID for the current row,
                # set its new centroid, and reset the disappeared
                # counter
                objectID = objectIDs[row]
                self.objects[objectID] = inputCentroids[col]
                self.disappeared[objectID] = 0
                # indicate that we have examined each of the row and
                # column indexes, respectively
                usedRows.add(row)
                usedCols.add(col)

            # compute both the row and column index we have NOT yet
            # examined
            unusedRows = set(range(0, D.shape[0])).difference(usedRows)
            unusedCols = set(range(0, D.shape[1])).difference(usedCols)

            # in the event that the number of object centroids is
            # equal or greater than the number of input centroids
            # we need to check and see if some of these objects have
            # potentially disappeared
            if D.shape[0] >= D.shape[1]:
                # loop over the unused row indexes
                for row in unusedRows:
                    # grab the object ID for the corresponding row
                    # index and increment the disappeared counter
                    objectID = objectIDs[row]
                    self.disappeared[objectID] += 1
                    # check to see if the number of consecutive
                    # frames the object has been marked "disappeared"
                    # for warrants deregistering the object
                    if self.disappeared[objectID] >
self.maxDisappeared:
                        self.deregister(objectID)

            # otherwise, if the number of input centroids is greater
            # than the number of existing object centroids we need to
            # register each new input centroid as a trackable object
            else:
                for col in unusedCols:
                    self.register(inputCentroids[col])
        # return the set of trackable objects
        return self.objects
```