DEVELOPMENT OF A CUSTOM YOLOV5N VEHICLE DETECTION ALGORITHM USING DEEPSORT TRACKING SYSTEM ON JETSON NANO PLATFORM

BY

ABUELGASIM SAADELDIN MANSOUR MOHAMED

A thesis submitted in fulfillment of the requirement for the degree of Master of Science in Engineering

Kulliyyah of Engineering
International Islamic University Malaysia

MAY 2023

ABSTRACT

In recent years, the advancements in deep learning and high-performance edge-computing systems have increased tremendously and have become the center of attention when it comes to the analyzing of video-based systems on the edge by making use of computer vision techniques. Intelligent Transportation Systems (ITS) are used for obtaining various road traffic analytics such as acquiring the total vehicle counts which is essential for maintaining constant traffic flows and it also replaces the use of traditional and laborious hardware devices with modern low-cost solutions. This thesis proposes and implements a modern, compact and reliable vehicle counting system which is based on the most recent and popular object detection algorithm as of writing this thesis known as the YOLOv5, combined with a state-of-the-art object tracking algorithm known as DeepSORT. The YOLOv5 will be used in the following system for the detection and classification of four different classes of vehicles whereas DeepSORT will be used for the tracking of those vehicles across different frames in the video sequence. Finally, a unique and efficient vehicle counting method will be implemented and used for the counting of tracked vehicles across the highway scenes. A new highway vehicle dataset consisting of four vehicle classes namely: car, motorcycle, bus and truck were collected, cleaned and annotated with a total of 11,982 images which will be published in the following study and used for the training of our robust vehicle detection model. From the results observed over real-world highway surveillance data, the following system was able to obtain an average vehicle detection mAP score of 96.1% and a vehicle counting accuracy of 95.39%, all while being able to be deployed on a compact Nvidia Jetson Nano edge computing device with an average speed of 15 FPS which outperforms other previously proposed tools in terms of both accuracy and speed.

ملخص البحث

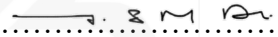
في السنوات الأخيرة، ازدادت التطورات في أنظمة التعلم العميق والحوسبة الطرفي ة عالية الأداء بشكل هائل، وأصبحت مركز الاهتمام فيما يتعلق بالتحليل الطر في للأنظمة القائمة على الفيديو من خلال الاستفادة من تقنيات الرؤية الحاسوبية. وتعد أنظمة النقل الذكية (ITS) من المجالات التي يمكن فيها استخدام التعلم العميق في العديد من المهمات، بما في ذلك أنظمة عد المركبات على الطرق السريعة، حيث يمكننا من خلال استخدام تقنيات الرؤية الحاسوبية، وجهاز حوسبة طرفية وكاميرات مثبتة في أماكن محددة على الطريق، يمكننا الحصول على نتائج دقيقة للغاية لعدد المركبات، حيث تستبدل الحلول الحديثة منخفضة التكلفة بالأجهزة التقليدية المرهقة. في هذه الدراسة، تم اقتراح وتنفيذ نظام حديث وصغير لعد للمركبات، بناء على أحدث خوارزمية لكشف الأجسام وأكثرها شيوعا حتى كتابة هذه الر سالة، المعروفة باسم (YOLOv5)، إضافة إلى خوارزمية تتبع الأجسام الأكثر حداثة، المعروفة باسم (DeepSORT). حيث سيتم استخدام (YOLOv5) في نظام التتبع للكشف وتصنيف أربع فئات مختلفة من المركبات، بينما سيتم استخدام (DeepSORT)لتتبع تلك المركبات عبر إطارات مختلفة في تسلسل الفيديو. أخيراً، سيتم تنفيذ طريقة فريدة وفعالة لعد المركبات، وسيتم استخدامها لعد المركبات المتعقبة عبر مشاهد الطريق السريع. لقد تم جمع بيانات جديدة لمركبات الطرق السريعة تتكون من أربع فئات من المركبات، هي: السيارة والدراجة النارية والحافلة والشاحنة، وتم تنظيف هذه البيانات والتعليق عليها بإجمالي 11982 صورة سيتم نشرها في الدراسة التالية واستخدامها لتدريب نموذجنا القوي للكشف عن المركبات. من النتائج التي لوحظت من خلال البيانات الحقيقية لمراقبة الطريق السريع، كان نظام التتبع قادراً على الحصول على متوسط دقة في الكشف عن المركبات مقداره 96.1٪ ودقة عد للمركبات مقدارها 95.39٪ كل ذلك مع القد رة على نشره على جهاز حوسبة طرفية صغير الحجم من نوع بمتوسط سرعة 15 إطار/ ث، الذي يتفوق على الأدوات الأخرى المقترحة سابقاً من حيث الدقة والسرعة.

APPROVAL PAGE

I certify that I have supervised and read this study and that in my opinion, it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Engineering (Mechatronics).

…….
Muhammad Mahbubur Rashid
Supervisor

…….
Nor Hidayati Diyana Binti Nordin
Co-Supervisor



…….
Jaffar Syed Mohamed Ali
Co-Supervisor

I certify that I have read this study and that in my opinion it conforms to acceptable standards of scholarly presentation and is fully adequate, in scope and quality, as a thesis for the degree of Master of Science in Engineering (Mechatronics).

…….
Amir Akramin Shafie
Examiner

…….
MD. Shabiul Islam
External Examiner

This thesis was submitted to the Department of Mechatronics Engineering and is accepted as a fulfilment of the requirement for the degree of Master of Science in Engineering.

…….
Ali Sophian
Head, Department of
Mechatronics Engineering

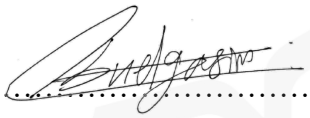
This thesis was submitted to the Kulliyyah of Engineering and is accepted as a fulfillment of the requirement for the degree of Master of Science in Engineering.

…….
Sany Izan Ihsan
Dean, Kulliyyah of Engineering

DECLARATION

I hereby declare that this thesis is the result of my own investigations, except where otherwise stated. I also declare that it has not been previously or concurrently submitted as a whole for any other degrees at IIUM or other institutions.

Abuelgasim Saadeldin Mansour Mohamed

Signature……  …… Date…04/05/2023…..

INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA

DECLARATION OF COPYRIGHT AND AFFIRMATION OF FAIR USE OF UNPUBLISHED RESEARCH

DEVELOPMENT OF A CUSTOM YOLOV5N VEHICLE DETECTION ALGORITHM USING DEEPSORT TRACKING SYSTEM ON JETSON NANO PLATFORM

I declare that the copyright holder of this thesis/dissertation are jointly owned by the student and IIUM.

By signing this form, I acknowledged that I have read and understand the IIUM Intellectual Property Right and Commercialization policy.

Affirmed by Abuelgasim Saadeldin Mansour Mohamed

…… …04/05/2023….

Signature Date

ACKNOWLEDGEMENTS

First and foremost, all praise is due to Allah (S.W.T), the Almighty and most Merciful, whose Grace and Blessings have been with me throughout the duration of this study and have allowed me to successfully complete this research work and writing of this thesis. I would also like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Muhammad Mahbubur Rashid as without his continuous supervision, support, guidance, immense knowledge and useful feedback, the successful completion of this research would not have been possible.

I would like to express my sincere gratitude and appreciation to my parents whom I am forever indebted to for all of the sacrifices and guidance that they have bestowed upon me and to my fellow colleagues and friends who have been with me since the start of this journey and have provided me with infinite support and spirit which have contributed to the successful completion of this research.

Finally, an honorable thanks is dedicated to the International Islamic University Malaysia, the Faculty of Engineering and the Center for Postgraduate Studies for providing me with access to the facilities required for conducting this research as well as with the technical hardware used for conducting the experiments.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ITS	Intelligent Transportation System
YOLO	You Only Look Once
SORT	Simple Online and Realtime Tracking
OSNet	Omni-Scale Network
R-CNN	Region-based Convolutional Neural Network
SSD	Single Shot Detector
SOT	Single Object Tracking
MOT	Multiple Object Tracking
DFT	Detection-Free Tracking
DBT	Detection-based Tracking
KF	Kalman Filter
CNN	Convolutional Neural Network
RPN	Region Proposal Network
ReID	Re-Identification
CBAM	Convolutional Block Attention Module

CHAPTER ONE

INTRODUCTION

1.1 OVERVIEW

In the past decade, the number of vehicles on the road have started to increase tremendously and this has led to huge amounts of traffic congestions as well as various other road safety concerns. Therefore, the study of traffic flow analysis began to rapidly grow as researchers try and identify ways of reducing congestions on the road for maintaining traffic control in order to ensure the safety as well as satisfaction of the general public. The data obtained from traffic flow analysis plays a significant role in gathering of essential information about roads and passing vehicles. This data can also help in the development of better traffic flow management systems by adjusting timings in traffic lights based on the traffic flow (Alpatov et al., 2018) or by extending roadways in certain areas which can result in not only a significant reduction in traffic congestions but also an increase in road safety and satisfaction.

Most of the vehicle counting systems that have been developed to date can be categorized as either being hardware or software-based systems (Mandal & Adu-Gyamfi, 2019). Traditionally, physical hardware sensors were placed underneath roadways and used to detect distinct moving objects and obtain the total vehicles count. Inductive loop detectors and Piezoelectric sensors were two of the most commonly used hardware devices (Matsuo et al., 1999). In recent years however, the use of traditional hardware sensors was found to be inefficient due to the high installation costs, regular site updates and inconvenient maintenances. Therefore recently, with the advancements in technology and computer vision techniques, researchers began to actively look into

vision-based solutions for help in maintaining constant traffic flows. Vision-based solutions are able to provide us with a more detailed information and are significantly easier to install and maintain as compared to traditional physical hardware sensors (Zhang et al., 2016). Consequently, there have been many vision-based solutions that have been deployed and many more which are actively being researched with the hopes of identifying a suitable solution which is low-cost, compact, efficient and can provide essential information about roads and passing vehicles reliably and in real-time.

The use of AI and computer vision in Intelligent Transportation Systems (ITS) is still considerably new and although there has already been wide research conducted in this field recently, there is still room for improvements as new advancements in the field of AI and edge-computing systems continue to emerge (Iyer, 2021). In this research, we will be making use of current state-of-the-art deep learning algorithms to develop a real-time, efficient and low-cost highway-based vehicle counting system which will be accurate and at the same time also compact enough to be able to run on a low computationally expensive edge computing device. A custom vehicle detection algorithm based on the YOLOv5n will be used in our system for the detection of four different classes of vehicles, meanwhile a Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT) will be used for the tracking of vehicles across different frames in the video sequence. Finally, a unique and efficient vehicle counting method will be developed and used for the counting of tracked vehicles across the highway scenes. The following system will be deployed on a compact Nvidia Jetson Nano edge computing platform and experiments will be conducted in real world test scenarios to evaluate the system performance.

1.2 PROBLEM STATEMENT

Traditionally, if not utilizing physical hardware devices placed underneath roadways, a human is usually behind the analysis of images being streamed from a CCTV camera directly towards a specified area of interest (Sreenu & Saleem Durai, 2019). As you can imagine, due to the human latency, it is almost impossible for a human to be able to accurately keep track and count the exact number of vehicles travelling across the highway manually and modern solutions designed to tackle this problem require high computational power (Barba-Guaman et al., 2020). Hence, by utilizing current state-of-the-art object detection and tracking algorithms along with modern edge computing devices, we are capable of solving all these problems.

1.3 RESEARCH OBJECTIVES

The primary objective of this research is to develop a low-cost, compact and efficient vehicle counting system which will be able to provide essential information about roads and passing vehicle both accurately and in real-time. The study leads to the following objectives:

1. To generate a new highway vehicle dataset consisting of car, motorcycle, bus and truck captured from a distance approximately 5 meters above the ground.
2. To train a custom YOLOv5 vehicle detection algorithm to detect as well as classify the four different vehicle classes with a relatively high accuracy score.
3. To integrate the trained YOLOv5 vehicle detector with a DeepSORT object tracker and develop a unique counter to count the vehicles as they cross the highway.
4. To deploy the entire system on an Nvidia Jetson Nano edge-computing platform and test and evaluate the performance in real-world scenarios.

1.4 RESEARCH MOTIVATION

The number of vehicles on the road have started to increase tremendously in modern times and this has led to various road safety as well as traffic concerns (Safiullin et al., 2018). The study of traffic flow analysis mainly for maintaining constant traffic flows and increasing the safety as well as satisfaction of the roads has started to increase amongst researchers (Iyer, 2021). By acquiring relevant data such as by obtaining the total vehicle counts, vehicle rates per hour, vehicle types along with other information surrounding roads and passing vehicles we can actually make use of this data for the development of better traffic management systems.

Traffic management plays an important role in city planning and regulating the density of vehicles on the road (Verani & Pitsiava-Latinopoulou, 2019). By actually analyzing the data that is obtained from these traffic managements systems, there are several actions which can be taken in order to minimize the level of congestions on the road and in result also increase the safety as well as satisfaction of the general public. Some of these actions include adjustments of traffic lights or expansions of roads based on the traffic flow (Alpatov et al., 2018).

1.5 THESIS ORGANIZATION

This thesis comprises of five chapters. Chapter Two presents the literature review which consists of research that has been conducted on the topics of vehicle detection, tracking and counting. It also covers research performed on previously proposed tools that are currently being utilized for the purpose of vehicle counting and reviews their methods. Chapter Three discusses the methodology of our proposed vehicle counting system and explains how it has been developed from data collection up to deployment. This chapter

talks about the dataset description, how the vehicle detection algorithm was developed and trained by using a YOLOv5 object detection algorithm as the base architecture, how it has been integrated with a custom DeepSORT object tracking algorithm and finally how the unique vehicle counter was developed. Chapter Four presents the evaluations of our vehicle counting system on never-before-seen test data and results are compared with other previously proposed tools. It also includes experiments that have been conducted on real-world test scenarios by deploying the system on an Nvidia Jetson Nano edge-computing platform and the performance as well as results are discussed. Finally, Chapter Five presents the conclusion and future works that can be made to the system in order to further increase its accuracy and speed.

1.6 SUMMARY

This chapter has provided a synopsis of the thesis and has highlighted on the main topic which is on vehicle counting systems. It opens with a brief background on the study and further describes the field of traffic management systems and the current issues that are being faced in our day-to-day lives and which need to be addressed. This is followed by the research objectives, stating the clear and concise goals of what we hope to achieve from this research work, followed by the research motivation stating the significance of the study and how it will contribute to the reduction of traffic congestions on the road as well as maintenance of constant traffic flows. Finally, the chapter concludes with the organization of the thesis and a chapter summary.

CHAPTER TWO

LITERATURE REVIEW

2.1 INTRODUCTION

The importance of vehicle counting in Intelligent Transportation Systems (ITS) is widely recognized. By accurately identifying the exact number of vehicles that are travelling on a roadway at a certain time, it is possible to perform various road traffic analysis such as obtaining the vehicle rate per hour which can be used as a key indicator for identifying solutions to reduce road traffic congestions. Vehicle counting and analysis started to become an active research topic amongst researchers in the early 19th century as the number of vehicles on the road started to increase and road transportation departments found the necessity in performing traffic flow management.

This chapter reviews previously proposed tools which have been identified for solving the problem of vehicle counting in highway scenes. Generally, most of the methods that have been identified for obtaining the vehicle counts can be classified as either being hardware or software-based solutions (Mandal & Adu-Gyamfi, 2019). Hardware based solutions although are very accurate, they can be fairly expensive and laborious to maintain. Therefore, in recent years researchers have mainly been focusing on identifying suitable software-based solutions for solving the problem of vehicle counting in highway scenes. When building of a comprehensive software-based vehicle counting system, vehicle detection, tracking and counting are the most crucial steps and collaboratively form the methods which most of the current research is based on.

2.2 BACKGROUND OF VEHICLE COUNTING

Computer vision-based vehicle counting is an interesting topic which has been widely researched and applied in recent years. The approaches used by researches to develop a software-based vehicle counting system can be broadly classified into five main categories which are by using: frame differencing (Tsai & Yeh, 2013), counting by detection (Toropov et al., 2015), motion based counting (Chen et al., 2012), counting by density estimation (H. Li & Zahr, 2012) and finally counting by using deep learning based approaches (Fachrie, 2020). The first three methods of counting vehicles by using frame differencing, detection and via motion have all been found to be environmentally sensitive and perform considerably poor in occluded environments and in videos with low frame rates (Toropov et al., 2015). The density-based vehicle counting approach has also been found to have a limited scope of detection and performs rather poorly on images that contain oversized vehicles (Mandal & Adu-Gyamfi, 2019). Out of all the methods identified by researches for obtaining the vehicle counts, deep learning-based approaches have shown the greatest results and development in recent years and therefore in this study, we will mainly be focusing on vehicle counting methods that have been developed by using deep learning-based architectures.

The main components of almost all deep learning-based vehicle counting system consists of three parts which are vehicle detection, tracking and counting. For the first part regarding the vehicle detection, different state-of-the-art deep learning-based object detection algorithms including two-stage object detectors such as the R-CNN, Fast R-CNN and Faster R-CNN will be reviewed and studied in detail along with one-stage object detectors such as YOLO, SSD and RetinaNet which will also be reviewed and discussed. As for the second part, regarding the vehicle tracking, cosine metric learning

used for training re-identification models for keeping track of vehicles in video sequences will also be studied thoroughly along with popular light-weight one-shot object tracking algorithms including SORT and DeepSORT will also be reviewed and discussed. Lastly, for the third part, several vehicle counting methods which have been developed for obtaining the total vehicle counts in highway scenes will also be studied and evaluated based on their effectiveness and accuracy.

2.3 VEHICLE DETECTION ALGORITHMS

2.3.1 Introduction

Vehicle Detection refers to the combination of both vehicle localization as well as classification. Vehicle localization refers to the drawing of a bounding box and pinpointing where exactly a vehicle is located in an image, meanwhile, vehicle classification refers to the assigning of a class label to that vehicle based on the vehicle features. Due to its involvement in the combination of both object localization as well as classification in a single algorithm, this makes it one of the most challenging topics in the domain of computer vision (Del-Blanco et al., 2012). Object detection algorithms used for performing the vehicle detection can be found in both machine learning as well as in deep learning, however, in this thesis, we will be mainly be focusing on deep learning-based algorithms which have been used for performing the vehicle detection.

Object detection algorithms have witnessed rapid revolutionary growths in the field of computer vision and have been utilized in several areas including robotics, medical imaging, telecommunications and mechanical engineering (Lohia, 2021) due to its ability to be able to locate where exactly an object is present in an image as opposed to image classification which just informs us that an object is present in the image but without knowing its exact location (Lohia, 2021). Object detection algorithms

make use of the distinct features present in every object in order to help differentiate it from other objects, identify its location and assign it with a class label. The Figure 2.1 shows the general overview of an object detection workflow in a typical computer vision task.

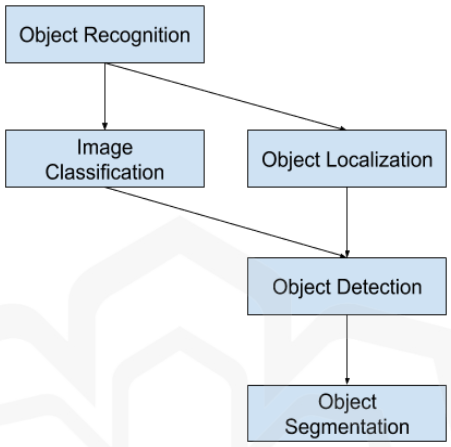


Figure 2.1 Overview of Object Detection in Computer Vision (Brownlee, 2019)

2.3.2 Two-Stage Object Detectors

The current deep learning-based object detection algorithms can be categorized as either being one-stage or two-stage object detectors. The Region-based Convolution Neural Network (R-CNN) was one of the first and most successful two-stage object detection algorithm to use deep learning and Convolutional Neural Networks (CNN) to solve the problem of object localization and classification (Girshick et al., 2014). The way the R-CNN works is by utilizing a selective search method to extract Regions of Interest (ROI) from an input image. Here an ROI refers to the rectangular coordinates that represents the boundaries of an object in the image. It is also possible for an image to have multiple ROIs as can be seen in the Figure 2.2. R-CNN will perform a selective search and generate over 2000 candidate regions of interest. Once all the candidate ROIs have been generated, they are then fed one by one into a Convolutional Neural Network (CNN)

where a 4096-dimensional feature vector is outputted. The CNN acts as a feature extractor and the outputs are then passed onto a Support Vector Machine (SVM) classifiers which will determine the type or class of the object that is present in the ROI. The R-CNN algorithm has shown remarkable results in the task of vehicle detection, however, due to the region proposals taking considerable amounts of time, this makes it unsuitable for deployment in real-time solutions.

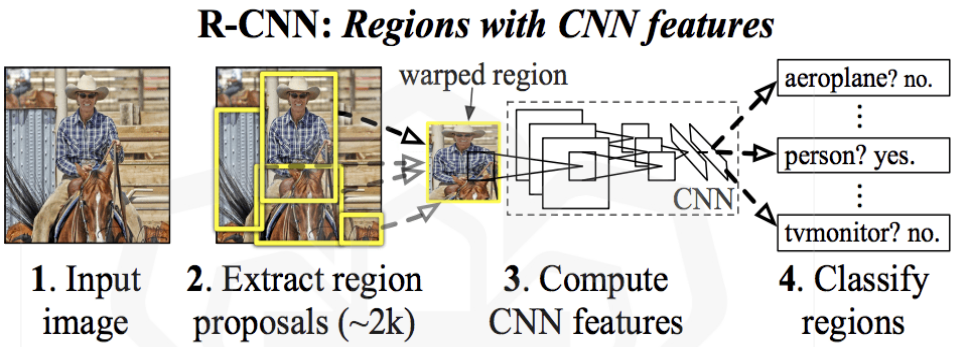


Figure 2.2 Summary of the R-CNN Model (Girshick et al., 2014)

Fast R-CNN is an extension of the R-CNN which was released by the same author (Girshick, 2015) a year later as a rich features hierarchies for accurate object detection and faster response times. It mainly addresses the speed issues that were faced by the R-CNN. The main difference is that Fast R-CNN is a single model instead of a pipeline and it is designed to learn and output regions and classifications directly from the input image. Fast R-CNN works by firstly passing the entire input image into a deep CNN where the end of the deep CNN is a custom layer called an ROI Pooling Layer. The ROI Pooling Layer is used to extract features specific to the given input candidate. The output from the CNN is then passed into a fully connected layer which produces two outputs, the first output being the class predictions using a SoftMax activation function and the second output being a linear output for the bounding boxes. The process is shown in the Figure 2.3 and is repeated multiple times for each input image.

The revised algorithm is significantly faster to train and to make predictions as compared to the previous R-CNN algorithm (Girshick, 2015), but it still requires a set of candidate regions to be proposed along with each input image which requires significant amounts of time and memory preventing it from being able to provide real-time solutions.

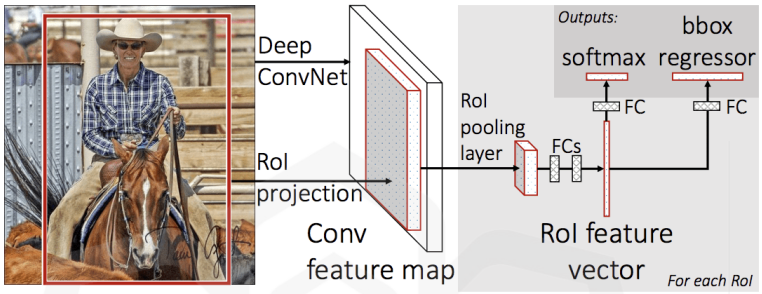


Figure 2.3 Summary of the Fast R-CNN Model Architecture (Girshick, 2015)

Faster R-CNN is a modified version of the Fast R-CNN model architecture with improvements in the speed of training as well as in performance. It was published by (Ren et al., 2017) and was designed to propose as well as refine the region proposals to become a part of the training process which is referred to as Region Proposal Network (RPN). An RPN is basically a fully Convolutional Neural Network which simultaneously predicts object bounds as well as object scores. The Faster R-CNN object detector is trained end-to-end to generate high quality region proposals which are then used to make predictions of objects that fall within that region. Due to the following improvements, Faster R-CNN reduces the overall number of region proposals generated and further accelerates the inference time to operate in almost real-time. The Faster R-CNN algorithm produces state-of-the-art results and eliminates the use of the selective search which is found in both R-CNN as well as Fast R-CNN algorithms, meanwhile here, the network is able to learn the region proposals simultaneously with the object bounds and object scores.

The Figure 2.4 shows a summary of the Faster R-CNN model architecture, as can be seen, the image is firstly fed into a deep CNN which outputs a convolutional feature map. Now, instead of using a selective search algorithm on the feature map to identify the region proposals, an RPN is used which takes the output of the deep CNN and predicts the region proposals. The predicted region proposals are then reshaped using the ROI pooling layer and used to classify the objects within the proposed region as well as predict the offset values for the bounding boxes.

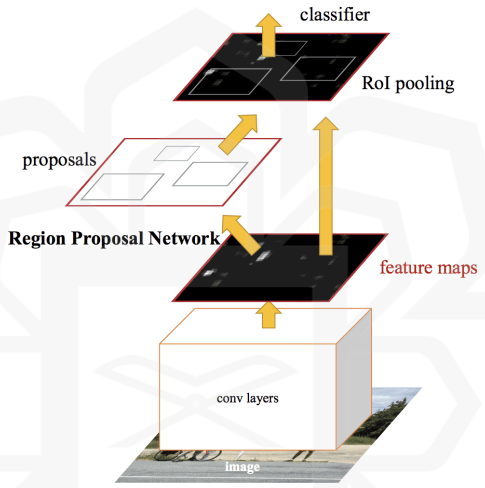


Figure 2.4 Summary of the Faster R-CNN Model Architecture (Ren et al., 2017)

2.3.3 One-Stage Object Detectors

One-stage object detectors refers to the class of object detectors which make use of only a single deep neural network for the detection of objects in an image. You Only Look Once (YOLO) is a popular one-stage object detector which has achieved state-of-the-art results in terms of both accuracy and speed (Redmon et al., 2016). YOLO is a real-time object detection algorithm which avoids spending too much time in generating region proposals and instead focuses on locating objects in images really quickly. The R-CNN family of object detectors may generally be more accurate than the one-stage object detectors but when it comes to speed, one-stage object detectors are far more

superior and have faster inference times. The first version of the YOLO algorithm was developed by (Redmon et al., 2016) and the approach involved only a single neural network that was trained end-to-end to detect objects in images. It takes in a raw image as input and predicts multiple bounding boxes proposals along with class labels for each of those bounding box directly in a single neural network. Due to its extremely fast inference time with an average speed of 65 Frames Per Second (FPS) (Anka, 2020), the predictive accuracy of the neural network was slightly compromised.

Technically, the YOLO algorithm works by firstly splitting the input image into grids cells which have custom pre-defined sizes as shown in the Figure 2.5. Each grid cell is then responsible for predicting bounding boxes and confidence scores of all objects that fall within that grid cell. Each cell also predicts a vector class which is composed of five components as represented in the equation (2.1) where $\boldsymbol{y}$ is the cell, $\boldsymbol{p_c}$is the confidence score, $\boldsymbol{b_x}, \boldsymbol{b_y}$ are the midpoints of the bounding box, $\boldsymbol{b_h}, \boldsymbol{b_w}$ are the width and height of the bounding box and finally $\boldsymbol{c}$ is the corresponding label of the object. Bounding boxes are then drawn with the more confident the prediction the bolder the bounding box. Then, a Non-Maximum Suppression (NMS) is used to filter out the bounding boxes according to their confidence scores which is pre-determined by a set threshold value. Finally, only the bounding boxes with accuracy scores above the pre-determined threshold value are kept while the rest are withdrawn resulting with only the best bounding boxes. A visual representation of the entire process flow of the YOLO object detection algorithm can be seen in the Figure 2.5.

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{p_c} \\ \boldsymbol{b_x} \\ \boldsymbol{b_y} \\ \boldsymbol{b_h} \\ \boldsymbol{b_w} \\ \boldsymbol{c} \end{bmatrix} \tag{2.1}$$

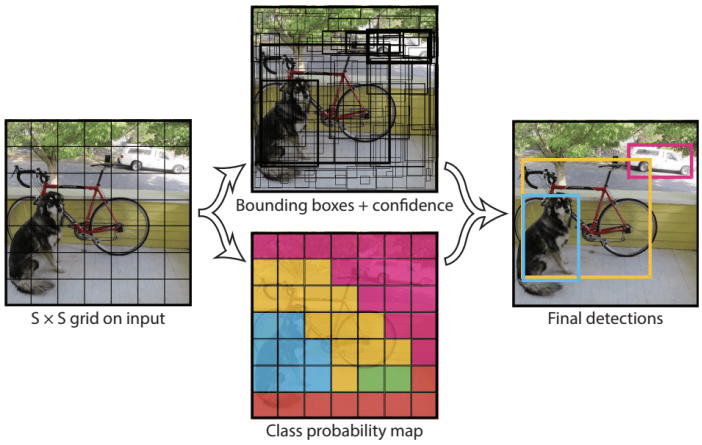


Figure 2.5 Flow of the YOLO Object Detection Algorithm (Redmon et al., 2016)

The YOLO algorithm was then updated by the original authors (Redmon & Farhadi, 2017) in an attempt to further improve the models detection accuracy. The algorithm was named YOLOv2 and a few of the notable changes included the addition of a batch normalization after each layer as well as including high resolution input images (Redmon & Farhadi, 2017). The major improvement however was in the addition of anchor boxes similar to what we have seen in the Faster R-CNN where we can set pre-defined bounding box sizes which we would expect to see from our dataset. This can be very useful for when we have two objects that have overlapping center points because the output vector of each grid cell is only able to output one class, therefore, by setting anchor boxes we are able to generate longer grid cell vector and associate multiple classes with each grid cell by comparing the detected objects with the pre-defined anchor boxes.

YOLOv2 is known for having a superfast network, however after some time various other alternatives came along with better accuracy scores such as the Single Shot Detector (SSD) which although may be slower than the YOLOv2, it overtakes it when it comes to accuracy (Chaudhari et al., 2020). YOLOv3 was then released a year later by (Redmon & Farhadi, 2018) and improved on the YOLOv2 by making use of a

more complex DarkNet-53 model architecture as the backbone which uses 53 convolutional layers instead of the previous DarkNet-19 model architecture with only 19 convolutional layers. The YOLOv3 model architecture is also complete with residual blocks and up-sampling networks which allowed for improvements in the performance, particularly in the detections of smaller objects. After the release of the YOLOv3, the original author Joseph Redmon stepped down from the computer vision research and in April 2020 Alex Bochkovskiy, one of the original co-authors released YOLOv4 (Bochkovskiy et al., 2020), this release was also written in the Darknet framework and included improvements in the feature aggregation, addition of a Cross Mini Batch Normalization and Mosaic Data Augmentation. Mosaic Data Augmentation refers to the combination of four training images into one image with certain aspect ratios, allowing for the model to learn to identify objects at a smaller scale than usual.

After a month of the release of the YOLOv4, researcher Glenn Jocher and his team at Ultralytics LLC published a new version of the YOLO algorithm, known as the YOLOv5 (Jocher et al., 2020). As opposed to all of the other previous versions, the YOLOv5 was written in Python and makes use of one of the most popular deep learning frameworks known as PyTorch. Glenn Jocher was also the original inventor of the Mosaic Data Augmentation which was first introduced in the YOLOv4 and acknowledged by Alex Bochkovskiy in his YOLOv4 paper (Bochkovskiy et al., 2020). The YOLOv5 included minor improvements such as addition of adaptive anchor boxes which allowed for the network to automatically learn the best anchor boxes from the dataset and use them during training (Hui, 2020). Due to this improvement along with the Mosaic Data Augmentation, the YOLOv5 was found to have a higher performance as compared to the YOLOv4 in certain circumstances (Thuan, 2021), but regardless, its main advantage is in the fact that it is written in the Python programming language

which makes the installation and integration with edge IoT edge devices much easier as compared to the previous versions which were all written in C programming language. Furthermore, the PyTorch open-source community is also much larger than that of the Darknet community which means that it will receive much more contributions, improvements and overall potential growth. However, because the YOLOv5 has been written in a different framework than its predecessors, it makes it slightly challenging to accurately compare the two object detection algorithm in terms of overall accuracy, speed and performance (Thuan, 2021).

In conclusion, object detection is one of the most popular streams in the field of computer vision today and there are many researchers that are currently working on this field and using it to solve various problems include vehicle detection in highway scenes. The two greatest advancements in the field of object detection came from the discovery of the one-stage and two-stage object detection algorithms. Two-stage object detectors were found to have a higher localization and recognition accuracy meanwhile one-stage object detectors were found to have higher inference speeds and are more suitable for deployments on edge devices. The breakdown of some of the most popular one-stage and two-stage object detectors that are commonly being used for the task of vehicle detections is shown in the Figure 2.6 and an overview of the discussed object detection algorithms can be seen in the Table 2.1.

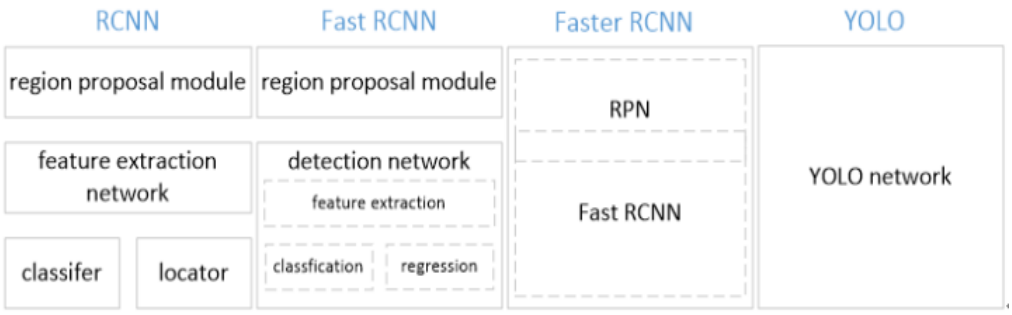


Figure 2.6 One-stage and Two-stage Object Detectors Breakdown (Weijie et al., 2022)

Table 2.1 Overview of Object Detection Algorithms

Object Detectors	Method Used	mAP@.5 (VOC 2012)	FPS	Model Size (MB)	Reference
R-CNN	Two-stage	40.9	0.02	103	(Girshick et al., 2014)
Fast R-CNN	Two-stage	52.7	0.5	149	(Girshick, 2015)
Faster R-CNN	Two-stage	73.2	11.23	182	(Ren et al., 2017)
SSD512	On-stage	79.2	19	188	(Liu et al., 2016)
YOLO	One-stage	57.9	45	179	(Redmon et al., 2016)
YOLOv2	One-stage	76.8	67	258	(Redmon & Farhadi, 2017)
YOLOv3	One-stage	76.3	41.28	236	(Redmon & Farhadi, 2018)
YOLOv4	One-stage	88.09	51.26	246	(Bochkovskiy et al., 2020)
YOLOv5s	One-stage	94.9	43.48	14	(Jocher et al., 2020)

2.4 VEHICLE TRACKING ALGORITHMS

2.4.1 Introduction

In vehicle detection, we basically detect a vehicle in a single frame, place a bounding box around that vehicle and classify it, while this area has been receiving a lot of attention from the computer vision community recently, another area which is less commonly known but with a great potential of widespread applications is in the tracking of vehicles over a sequence of frames (MacHiraju et al., 2021). Vehicle tracking refers to the process of predicting the position of a target vehicle in consecutives frames in a

video once its initial detection has been identified (Maiya, 2019). In vehicle tracking, we basically take an initial set of vehicle detection points, create a unique identification ID for that vehicle and then track the vehicle as it moves around different frames in the video while still maintaining its same ID assignment (MacHiraju et al., 2021).

Tracking of objects is a fundamental task in any video-based computer vision application because it allows you to not only see things in a pixel to pixel static image, but capture the overarching pattern of an object in a video (Maiya, 2019). The aim of object tracking is to be able to discover the route of an object or multiple objects in a video from a succession of images. It has a multitude of real-life applications, some of which include security and surveillance (Stauffer & Grimson, 2000), robot vision (Sakagami et al., 2002), medical diagnosis systems (Jakubiak & Radomski, 2013), movement control (Shehu, 2010) and traffic monitoring (Tian et al., 2011).

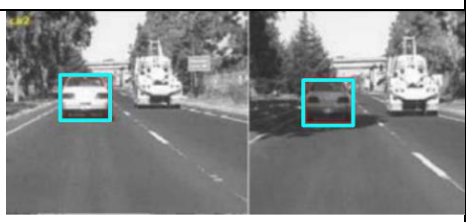
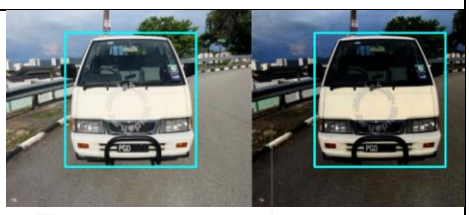
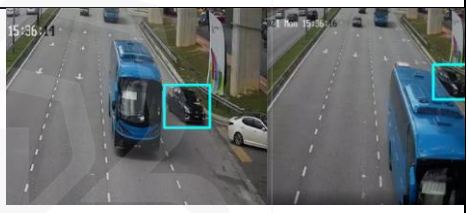
Generally, the current object tracking algorithms that have been identified and used for the purpose of vehicle tracking can be classified into two main categories which are: Single Object Tracking (SOT) and Multiple Object Tracking (MOT). In the SOT approach, only a single vehicle is tracked over subsequent frames, even when the environment contains multiple vehicles that are present, meanwhile in the MOT approach, all of the vehicles that are present in the environment are tracked across the entire video stream (Barla, 2022). In this thesis, we will mainly be focusing on research that has been conducted on the MOT approach, the common challenges that are faced when working with MOT systems and the deep learning-based solutions that have been developed for tackling of these challenges.

2.4.2 Challenges of Vehicle Tracking

Tracking of non-stationary objects through a sequence of frames has always been very challenging task in the field of computer vision, especially when working with MOT systems as you would need to deal with the challenges faced by both SOT as well as MOT systems combined (W. Luo et al., 2021). Nevertheless, due to its enormous growth and wide use of applications such as in surveillance, human-computer interaction, sports events and navigation systems (Lee et al., 2014), researchers continue to actively try and identify solutions to these challenges.

Generally, the most common challenges faced when working with MOT systems for the task of vehicle tracking include: illumination variation, dynamic background, occlusion and video noise. Illumination changes refers to the changes that are observed in the target vehicle due to changes in its surroundings, for example the force of light on the target vehicle may change from time to time which makes it hard for the tracker to be able to keep track of the vehicle across the entire video sequence. Dynamic background refers to the changes in the background scenery in which the vehicle is present, for instance, when tracking of outdoor vehicles, the weather may change from time to time throughout the day resulting in a dynamic background. Occlusion refers to the blockage of the vehicle from the camera view and this is one of the most common challenge faced when working with vehicle trackers today and it often occurs when two vehicles come very close to each other to the point where they seemingly merge or combine with one another. Finally, video noise refers to the noise that is generated during video processes when there is no transmission signal or when the signal is weak which leads to the degradation of the quality of video signals. The MOT challenges discussed above are presented visually in the Table 2.2.

Table 2.2 Vehicle Tracking Challenges

Challenge	Description	Example
Illumination Variation	The illumination of the target object is significantly changed due to its surroundings.	
Dynamic Background	The background or scenery of the target object is changed.	
Occlusion	The target object is partially or fully occluded.	
Video Noise	The number of pixels on the target object bounding box is low resulting in noise.	

2.4.3 Multiple Object Tracking (MOT)

Multiple Object Tracking (MOT) is a popular field in object tracking which has gained an increasing amount of attention due to its academic as well as commercial potential (W. Luo et al., 2021). MOT refers to the approach where the tracking algorithm is able to keep track of multiple objects of interest simultaneously in a sequence of images. In the case of vehicle tracking, it involves locking onto each vehicle in the frame, uniquely identifying it and maintaining its identity as it moves around in different frames in the video sequence. The main goal of the MOT is to be able to discover multiple vehicles

simultaneously in individual frames and recover their identities across continuous frames (Rhodes, 2018).

Most of the MOT algorithms that have been discovered to date can be divided into three main parts which are Detection, Prediction and Data association (Rhodes, 2018) as can be seen in the Figure 2.7. The first part is the detection module which involves detecting a vehicle of interest or multiple vehicles of interest in the video frame, this can be done by using various techniques such as by using background subtraction or using one of the deep learning one-stage or two-stage object detection algorithms which have been discussed in the previous section. Secondly, in order to track a vehicle over time, you need to be able to predict its location in the next frame. The simplest method that can be used to predict a vehicle's location is by using its last known location, this is an effective approach for videos with high frame rates as the difference between each frame and the next is very subtle and so the next location of the vehicle can be guessed, however, this method becomes inefficient when the target object is moving at varying speeds, hence, the Kalman Filter is used.

Kalman Filter (KF) is a very popular predictive method which makes use of the vehicle's previously observed velocity and motion in order to predict its next location (Q. Li et al., 2016). It is able to perform estimations of past, present and future states of a vehicle without even knowing its precise location. It is also a well-known practical tool due to its simplicity and ability to work very fast making it very suitable for real-time solutions and deployment in low computationally expensive edge devices. Finally, the last stage of almost all MOT algorithms is the data association which refers to the process of associating detections of the corresponding vehicle across the different frames (Rakai et al., 2022).

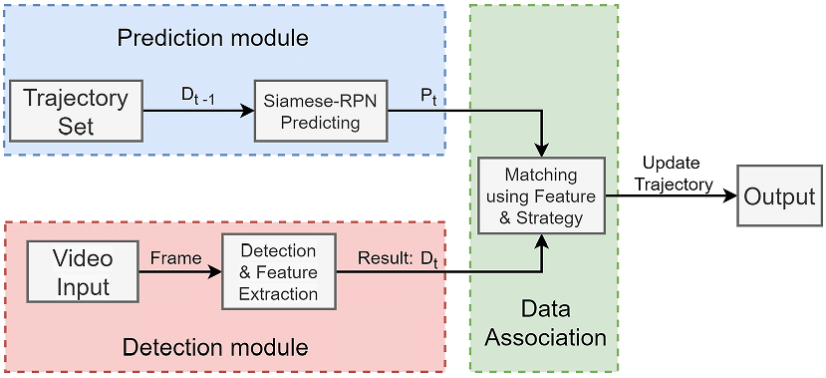


Figure 2.7 Flowchart of the Different Stages in MOT algorithms (Gao et al., 2022)

The existing MOT frameworks that have been developed and used for the purpose of vehicle tracking can be categorized as either being: Detection-Free Tracking (DFT) and Detection-Based Tacking (DBT) (Rhodes, 2018). DFT refers to the manual initialization of a fixed number of vehicles in the first frame, then utilizing a tracker to localize the vehicles in the subsequent frames. DBT on the other hand is the more popular MOT framework, here the vehicles are firstly detected by using a trained vehicle detector and then are linked into trajectories (Rhodes, 2018). By using this method, the tracker is able to automatically discover newly detected vehicles and also terminate those vehicles that have already left the frame.

The DBT framework works by firstly applying a type-specific vehicle detector in each frame and obtaining an object hypothesis, the tracking is then conducted in order to link the vehicle detection hypothesis into trajectories (W. Luo et al., 2021). By using this approach, there are two main issues worth noting, the first is that since the vehicle detector is trained in advance, the majority of the Detection-Based Tracking will focus mainly on specific trained targets. Secondly, the performance of the Detection-Based Tracking will be highly dependent on the performance of the deployed object detector, hence, selection of an accurate vehicle detector is crucial. The Figure 2.8 shows the typical procedure flow of a vehicle Detection-Based Tracking approach.

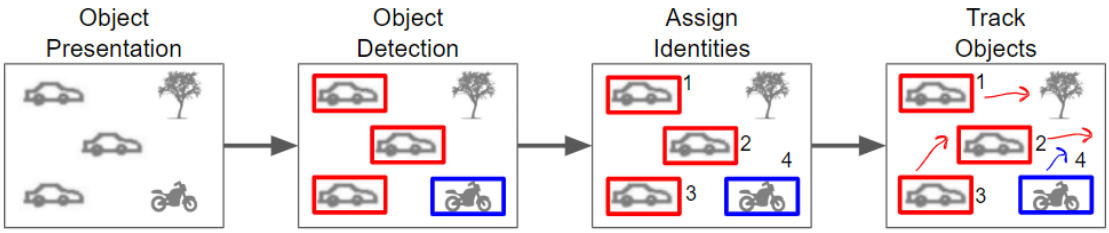


Figure 2.8 Typical Flow of a Vehicle Detection-Based Tracking

The DBT framework is the more prominent framework for the task of vehicle tracking due to its ability to be able to automatically discover newly detected vehicles and terminate those vehicles that have already left the cameras field of view. As for the DFT framework it is free from any pre-trained vehicle detectors, however, it is unable to accommodate for cases in which secondary vehicles suddenly appear (Rhodes, 2018).

2.4.3.1 Kalman Filters

Kalman Filter (KF) is an algorithm that was first pioneered by Rudolph Kalman in the early 1960s (Kalman, 1960). The Kalman filter is an algorithm that provides estimates of some unknown variable given its measurement observed over time (Kim & Bang, 2019). Due to its relatively simple form and low computational power, it has long been regarded as the go-to algorithm when dealing with computer vision tasks that require keeping track of an object or multiple objects over time (Maiya, 2019). The Kalman Filter falls under the category of Point Tracking and it works by providing estimates of some unknown variable given its measurements over a period of time (Gunjal et al., 2018).

The Kalman Filter is a commonly used algorithm for the task of vehicle tracking and can be typically divided into two main stages which are Prediction and Correction (Maiya, 2019). In the Prediction stage, the Kalman Filter will be used to predict the current state of a vehicle based on some previous value along with providing the

uncertainties of that prediction by using the *state equations* shown in the Figure 2.9. Once the predicted measurements have been obtained, the second stage, known as the Correction, takes those predicted measurements and corrects the predictions based on the *measurement equation* shown on the right-hand side of the Figure 2.9 and the optimal state of the vehicle is obtained. So, the algorithm works by firstly making a prediction of a state, based on some previous value, the measurement of that state is then obtained by using sensor data and the predictions are updated based on the errors. The algorithm is a recursive method that runs in real-time and only uses the present input value measured and previously calculated state along with the uncertainty matrix in order to derive the correct location of the vehicle (Monica & Nigel, 2017).

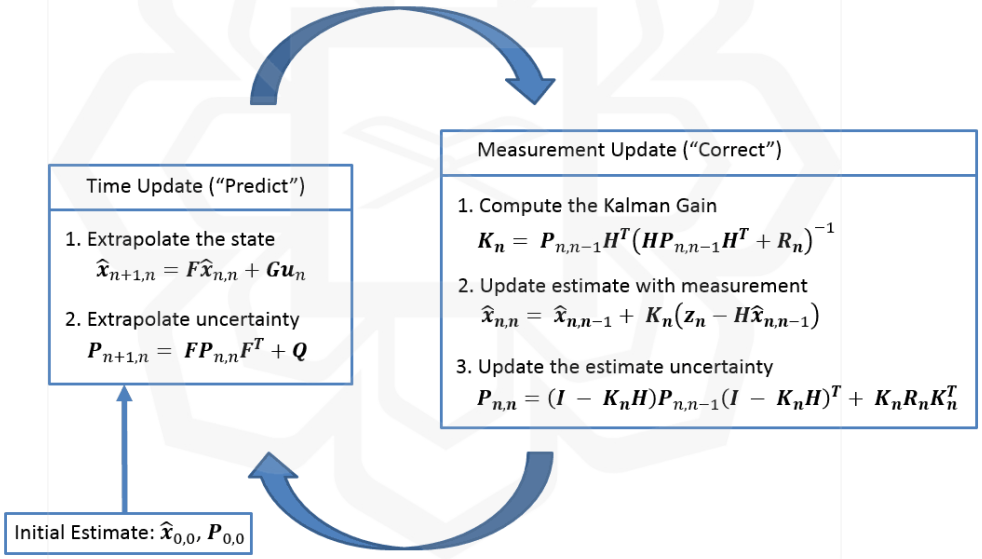


Figure 2.9 Time and measurement update of a Kalman Filer (Roehrig & Müller, 2009)

The Kalman Filter is a very powerful algorithm as it has the ability to perform estimations of past, present and future states of a vehicle without even knowing its precise location. It is also a well-known practical tool due to its simplicity and ability to work very fast making it very suitable for real time solutions and in embedded systems. Furthermore, the Kalman Filter is also very useful in systems that are very

noisy as its main goal is to filter out the noise from an input data as can be observed in the Figure 2.10 which shows that much of what the Kalman Filter does is just propagating and updating gaussians and their covariances. The Kalman Filter firstly predicts the next state of a vehicle from the provided initial state transition, then the noisy measurement information is incorporated in the correction stage. Finally, the state of the vehicle is obtained by using sensor data and predictions are updated based on the errors in order to derive the correct location of the vehicle and the algorithm is repeated all over again.

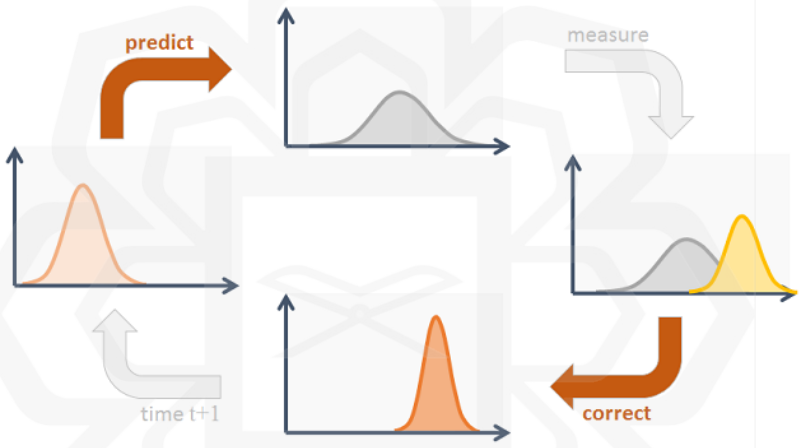


Figure 2.10 The cycle of a Kalman Filter (Q. Li et al., 2016)

The main idea of the Kalman Filter is to make use of the available detections of an object as well as the previous predictions in order to come up with an estimate of the current state of that object, while also keeping the possibility of errors in the process (Q. Li et al., 2016). This algorithm has been used in many modern deep learning applications where the tracking of an object or multiple objects is required. Some of the applications where the Kalman Filter has been utilized in include the tracking of a football in a sports arena (Monica & Nigel, 2017), location and navigation systems (Zahaby et al., 2009) and in computer graphics (Kim & Bang, 2019).

2.4.3.2 DeepSORT

Deep Simple and Online Real-time Tracking (DeepSORT) is one of the fastest and most robust DBT algorithm to date (Lyu et al., 2022). It is an extension of the Simple Online and Real-time Tracking (SORT) (Bewley et al., 2016) and makes use of a deep learning-based approach in the early stages which has shown remarkable results in the handling of occlusion and keeping track of target objects for longer durations of time, effectively reducing the number of identity switches because the network is now able to recognize tracked objects by their extracted features, therefore making it a superior method (Mandal & Adu-Gyamfi, 2019).

The original SORT algorithm comprises of three main stages which are: Detection, Estimation and Association (Bewley et al., 2016). The first stage, the Detection, involves detecting an object of interest in the initial stage i. Once the object location has been identified, Estimation is performed which involves predicting the future location of the object i+1 by making use of Kalman Filters. As discussed in the previous section, it is also worth noting that the Kalman Filter only approximates the object's new location along with the possibility of errors, hence, it needs to be optimized. Once the Kalman Filter has obtained an estimate of the future location of the object i+1, it is optimized by using the ground truth location which is obtained by detecting the position of the object in the position i+1. The problem is then solved optimally by using the Hungarian algorithm for Association and identifying if the object detected in the current frame is the same as that in the previous frame (Vision, 2019).

The DeepSORT algorithm enhances on the SORT by incorporating deep learning techniques in the early stages in order to allow for longer tracking of the detected objects (Maiya, 2019). It makes use of deep neural network to allow for the

SORT to estimate the objects location with greater robustness against object misses and occlusions because the network is now able to recognize the tracked object by using its extracted features (Wojke et al., 2018). Essentially, the deep neural network is trained on a task-specific dataset for the task of object detection, once the training had completed and the model has obtained a fairly high accuracy score, the classifier is then stripped off and only the extracted features from the dataset remains. This extracted features are then utilized by the SORT algorithm to keep track of the moving objects across subsequent frames (Barla, 2022).

The Kalman Filter is a very crucial part in the DeepSORT algorithm as it helps in factoring out the noise in detections and make use of the prior state in order to come up with a good prediction for the bounding box (Maiya, 2019). Once the new bounding box have been predicted by the Kalman Filter, association of new detections is required with these new predictions, to accomplish this, the DeepSORT author makes use of the squared Mahalanobis distance (Wojke et al., 2018) to encompass the uncertainties achieved by the Kalman Filter. Finally, the Hungarian algorithm is used for association and ID attribution, identifying if the object in the current frame is the same as that observed in the previous frame (Vision, 2019). The block diagram of the discussed flow of data in the DeepSORT algorithm can be seen in the Figure 2.11.

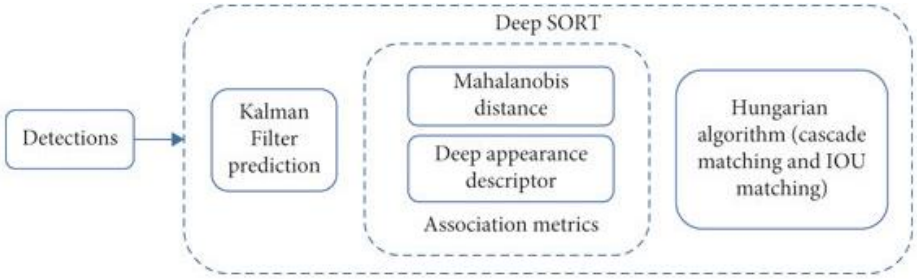


Figure 2.11 Flow of Data in the DeepSORT algorithm (Dinh et al., 2021)

The CNN architecture of the DeepSORT algorithm is shown in the Table 2.3, as can be seen, a deep residual network consisting of eight layers is applied, comprising of two convolutional layers followed by six residual blocks. Once the deep neural network has been trained and had achieved a reasonably good accuracy, the final classification layer is then stripped off and this is known as the appearance descriptor (Barla, 2022). In the Table 2.3, the "Dense 10" layer is considered as the appearance descriptor with a dimensional feature vector of 128. Finally, batch and l2 normalization are then applied to make the neural network faster and more stable through re-centering and re-scaling (Mandal & Adu-Gyamfi, 2019).

Table 2.3 Overview of the DeepSORT CNN Architecture (Mandal & Adu-Gyamfi, 2019)

Name	Patch Size/Stride	Output Size
Conv 1	$3 \times 3/1$	$32 \times 128 \times 64$
Conv 2	$3 \times 3/1$	$32 \times 128 \times 64$
Max Pool 3	$3 \times 3/2$	$32 \times 64 \times 32$
Residual 4	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 5	$3 \times 3/1$	$32 \times 64 \times 32$
Residual 6	$3 \times 3/2$	$64 \times 32 \times 16$
Residual 7	$3 \times 3/1$	$64 \times 32 \times 16$
Residual 8	$3 \times 3/2$	$128 \times 16 \times 8$
Residual 9	$3 \times 3/1$	$128 \times 16 \times 8$
Dense 10		128
Batch and ℓ_2 normalization		128

To summarize, the DeepSORT algorithm works is by firstly obtaining the bounding box coordinates of the target object by using a CNN based object detector, once the detections for one frame is obtained, a unique ID is assigned to each detected object and matching is performed for similar detections with respect to the previous

frame by using Kalman Filters. Once the predicted states of the object have been identified by using the Kalman Filter, the new detection is associated with the previous frame. This association is calculated by using the Hungarian algorithm with an association metric that measures the bounding box's overlap after a specified "n" number of frames (Santos et al., 2020).

2.5 VEHICLE COUNTING METHODS

A vehicle counter is one which is able to count the number of vehicles as they pass through a specified area in a roadway. Most of the systems that have been developed for solving the problem of vehicle counting can be divided into three main stages which are: vehicle detection, tracking and counting (Alpatov et al., 2018). The vehicle detection and tracking has already been discussed in the earlier sections, as for this section, we will be reviewing various methods and techniques that have been identified by researchers for obtaining the total vehicle counts in highway scenes. Before jumping into the study, it is important note that highway scenes used by researchers for obtaining the total vehicle counts can be divided into two main types which are one-way highway scenes and two-way highway scenes as can be seen in the Figure 2.12 (A) and (B) respectively. Most of the vehicle counting methods that have been identified are able to function well on both scenes, however, there are some methods that have been developed solely for counting vehicle in the one-way highway scenes.

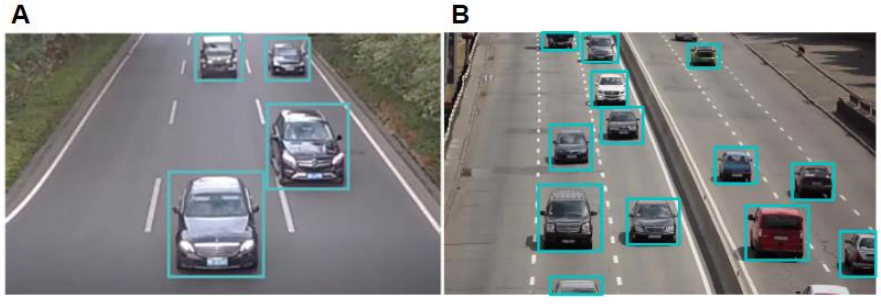


Figure 2.12 One-way and Two-way Highway Scenes

Technically, the methods used by researchers for obtaining the total vehicle counts can be classified as either being hardware or software-based solutions (Mandal & Adu-Gyamfi, 2019). Inductive loop detectors and piezoelectric sensors are two of the most commonly used hardware devices used for obtaining the vehicle counts (Samadi et al., 2012). Inductive loop detectors consist of wires that are coiled up to form a loop which is usually in the shape of a square as can be seen in the Figure 2.13, these wires are installed under the surface of roadways and are used to detect and count the number of vehicles by using induced currents from the loop coils which changes as the vehicle crosses over the loops. These sensors are extensively used in highways and roads as they possess very high accuracies and are economical as well (Samadi et al., 2012).

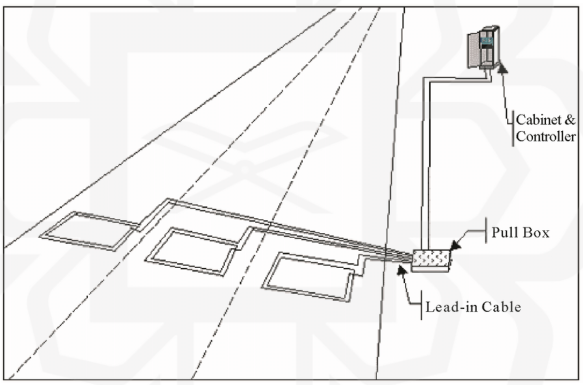


Figure 2.13 Inductive Loop Sensors (Samadi et al., 2012)

In recent years however, with the advancements in computer vision and image processing techniques, hardware-based sensors have been found to be inefficient due to their high installation costs, regular site updates and inconvenient maintenances (Mandal & Adu-Gyamfi, 2019). Therefore, recently, researchers have started looking into software-based solutions for obtaining the vehicle counts. The researcher (Song & Liang, 2019) in his paper proposed a software-based vehicle counting technique which makes use of the trajectory analysis of vehicles in order to obtain the vehicle counts. In

this paper, the author makes use of an Oriented Fast and Rotated Brief (ORB) algorithm in order to obtain the trajectory of the vehicle, the travelling direction of the vehicle is then automatically derived and by setting a virtual line counter, the researcher was able to obtain the total vehicle counts efficiently for both vehicles travelling on on-going as well as on out-going traffic (Song & Liang, 2019).

Another method proposed by (Fachrie, 2020) makes use of a different strategy which can count the number of vehicles without having to track their movement or trajectories from frame to frame. This method was designed specifically for counting vehicles in one-way highway scenes and it works by firstly placing a virtual border line on the video, then detecting the different moving vehicles by using a YOLOv3 object detection algorithm and calculating the distance from the border line to the vehicle's centroid as illustrated in the Figure 2.14. Next, if the distance between the border line and the vehicle's centroid is less than or equal to a pre-determined threshold value, the vehicle count is incremented accordingly (Fachrie, 2020). The suggested method performs really well in one-way highway scenes and achieves an average vehicle counting accuracy of 93.20% outperforming the ORB algorithm, however, many trial and errors are required each time the camera angle changes in order to obtain an ideal threshold value for the distance.

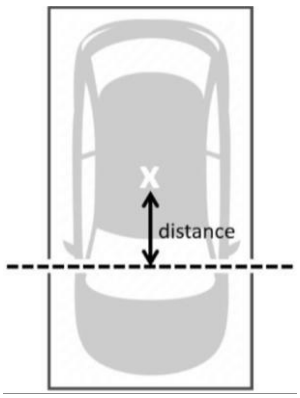


Figure 2.14 Vehicle Counting based on Centroid Distance (Fachrie, 2020)

Another technique suggested by (Xiang et al., 2018) makes use of a regional mark and a virtual test line to perform the vehicle counting. Here, the regional mark is defined as an area in the video which is used to count the vehicles in, whereas, the virtual line is a border line which is drawn on the video. The researcher here makes use of a stand-alone vehicle tracking algorithm to keep track of the vehicles across the highway and store their ID and vehicle direction information which will be used to count the vehicles as they enter into the regional mark area. For example, let's assume that the number of vehicles tracked at frame f is t. Then, if the vehicle tracked at frame f+1 has a different ID, a plus 1 is added to the counter (Xiang et al., 2018). Due to this technique only making use of a tracking algorithm to perform the vehicle counts, no additional computational load is required which allows the system to work considerably fast (Xiang et al., 2018). However, if there are any identity switches that are experienced during the inference, this may result in inaccurate vehicle counts. Furthermore, by using this method you are unable to obtain the individual vehicle counts for cars, buses or trucks but only obtain the total vehicles count. The Figure 2.15 illustrates the proposed method in action using an aerial video at a height of 50 meters.

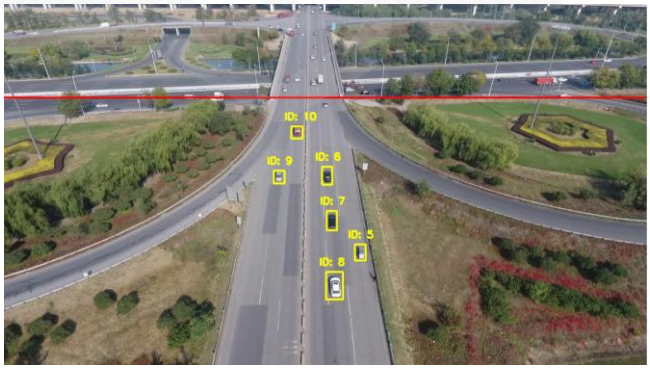


Figure 2.15 Vehicle Tracking Results using Aerial Video (Xiang et al., 2018)

Lastly, another efficient vehicle counting method that has been used by both researchers (Abdullah & Oothariasamy, 2020) and (Lyu et al., 2022) makes use of a

virtual reference line counter which is placed on the road and separates the scene into two regions. The number of vehicles is counted as they cross through the virtual reference line and move from one region onto the next. This technique firstly involves the detection of the vehicles by using a CNN architecture, then assigning of the vehicles with a unique ID and tracking them as they move across in different frames until they reach the virtual line counter. Once the vehicles have crossed through the reference line, they enter into a different region and the vehicle counts is incremented accordingly (Abdullah & Oothariasamy, 2020). For example, let's assume that we have two vehicles as shown in the Figure 2.16, where (x1, y1) & (x2, y2) represents the center coordinates of the first vehicle and (m1, n1) & (m2, n2) represents the center coordinates of the second vehicle. Now, as the center coordinates of the first vehicle crosses over the reference line, the Euclidean distance is calculated which determines if the vehicle has crossed over the first region or not and if so then the vehicle count is increased by one. This method works well in one-way highway scenes and has achieved an average vehicle detection accuracy score of 80.90% using the YOLOv3 network with DarkNet19 architecture and achieved an average counting accuracy of 66.29% (Abdullah & Oothariasamy, 2020). The following method however was found to be unsuitable in two-way highway scenes as it assumes that the vehicles are moving in only a single direction.

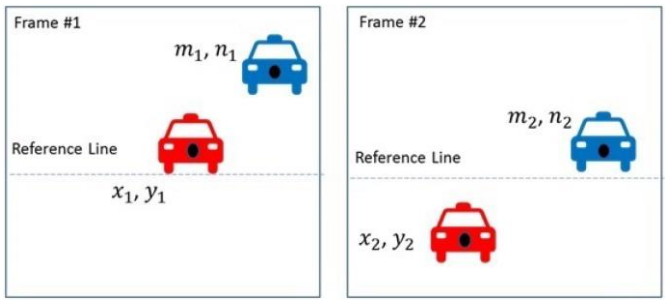


Figure 2.16 Illustration of Two Vehicles at Consequent Frames (Abdullah & Oothariasamy, 2020)

2.6 DEEP LEARNING MODEL DEPLOYMENT

2.6.1 Model Optimization

Deep learning and Machine learning are well known for producing models that have high accuracy and prediction capabilities, this however usually comes at the expense of a higher power and memory consumption (D'costa, 2020). In order to train a deep neural network to have a high accuracy and prediction score, massive computational resources including Graphical Processing Unit (GPU) and Tensor Processing Unit (TPU) are required. By performing model optimization, this significantly helps in reducing the model size and allows for deployment on low computationally expensive edge computing devices with real-time capabilities (Industries et al., 2019).

There are many techniques that can be used to optimize deep learning models in order to make them suitable for deployment on embedded devices, some of which include Pruning and Parameter Quantization (Industries et al., 2019). Pruning is a commonly used technique that is used to reduce the size of a model by eliminating unnecessary values in weighted tensors (D'costa, 2020). It works by setting a few of the neural network parameters to zero, reducing the unnecessary computation that contains zero values and resulting in a smaller model with less computational resources while still maintaining the same or a minimal decrease in the accuracy score. The Figure 2.17 illustrates the pruning technique used for model optimization. Parameter Quantization is another technique that is commonly used to optimize deep learning models and it works by rounding up a neural network's weights from that of a floating point, typically float32 to that of an integer, int8. By reducing the bit width of the numbers, this can drastically reduce the computational cost and memory requirement of the system resulting in a more compact model (Industries et al., 2019).

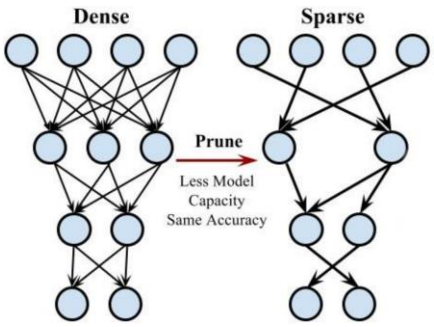


Figure 2.17 Pruning Model Optimization technique (D'costa, 2020)

2.6.2 Model Deployment on Edge Devices

Model deployment is usually one of the last stages in almost any deep learning project lifecycle and it comes right after the model optimization stage as can be seen in the Figure 2.18. Model deployment involves integrating your deep learning model into production and actually testing your system in real-world test scenarios to see how well your model is able to perform when provided with real-world input data. When working with private and confidential data such as in Intelligent Transportation Systems (ITS), it is often desirable to deploy these systems on edge computing devices in order to further increase the system security and allow for a more scalable system instead of relying solely on cloud computing (Leroux et al., 2022).

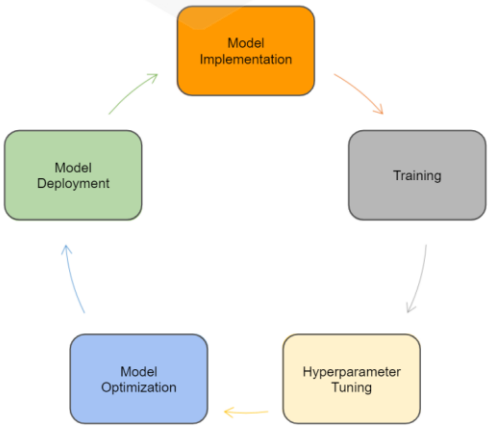


Figure 2.18 The typical lifecycle of a Machine Learning project

There are approximately 15 billion edge devices that are available in the market today that can be used for the deployment of AI systems (Industries et al., 2019). Edge-computing devices were first introduced after the enormous growth of IoT devices which are able to connect to the internet and transmit and receive data over the cloud (Zhang et al., 2019). IoT devices have a great potential when deployed alone and are able to transmit as well as receive data across different networks effortlessly, however, as the number of IoT devices increase, huge volumes of data are generated which can result in bottlenecks and an increase in cost, especially when working with visual sensors such as cameras (Leroux et al., 2022). Hence, edge-computing devices were introduced which brings the data storage and computation closer to the hardware where its collection is taking place instead of relying on a central server which is located several miles away for performing the computation.

In recent years, AI edge systems have undergone ground-breaking computing transformation and are now able to process large amounts of data locally while also having other substantial advantages over conventional system including scalability, optimized resources, reliability, and security (Kristiani et al., 2020). Edge-computing devices are relatively small in size, power-efficient and are application-oriented which makes them suitable for deployment of AI systems. They are capable of processing data locally and use sensors and signals which are generated from the device itself, allowing for it to be independent and make decisions in real-time without the help from any external connections (Kristiani et al., 2020). Furthermore, the decentralized character of edge-computing platforms also allows for them to become more robust since the individual edge nodes are able to operate autonomously and provide offline capabilities (Leroux et al., 2022).

In critical AI applications including ITS and vehicle counting systems which are used for maintaining traffic control and ensuring the safety of the general public, a highly reliable infrastructure is required which can only be provided by edge-computing platforms. Moreover, due to the processing of data locally, this also eliminates the streaming and storing of strictly confidential data to the cloud, including individual's vehicle plate numbers which can be exploited, meanwhile, edge-computing platforms allows for only the vehicle counts to be transmitted over to the sever in order to perform the required analysis.

2.7 SUMMARY

This chapter touched on the main components which are required in order to build a comprehensive deep learning-based vehicle counting system which comprises of: vehicle detection, tracking and counting. It reviewed and analyzed the various algorithms that have been discovered by researchers for performing the vehicle detection and tracking. Then, various vehicle counting techniques that have been used by researchers for obtaining the total vehicle counts were also analyzed and discussed, including the performance, pros and cons of each method. Eventually, the most suitable algorithms that has been identified for solving the problem of vehicle detection in highway scenes was found to be by using the state-of-the-art YOLOv5 object detection algorithm as the base architecture and modifying the layers for small vehicle detection. As for the vehicle tracking, a robust DeepSORT object tracking algorithm was found to be suitable for the purpose vehicle tracking in highway scenes. Finally, model optimization techniques were also discussed which are required in order to further increase the detection accuracy of vehicles and improve the model's inference speed for deployment on edge-computing platforms.

CHAPTER THREE

RESEARCH METHODOLOGY

3.1 INTRODUCTION

This chapter presents the methodology of our proposed vehicle counting system by combination of state-of-the-art object detection and tracking algorithms and explains how it has been developed from data collection up to deployment and describes the environmental setup that has been developed for testing of the system. Here, we introduce the data description, collection, cleaning and annotation process, this is followed by an elaboration on how the custom YOLOv5 object detection algorithm was optimized and trained for the purpose of vehicle detection. Then, we present how the system has been integrated with a robust DeepSORT object tracking algorithm and a fast OSNet ReID model for the tracking of vehicles across different frames in the video sequence. Finally, we introduce a new and efficient vehicle counting method which will be used for counting the different vehicle types as they cross through a virtual polygon area in the highway in real-time. In this study, our dataset consists of 4 main classes of vehicles which will be detected and counted which are: car, motorcycle, bus and truck.

3.2 RESEARCH METHODOLOGY

Firstly, the research commences with the gathering of essential information on the three main topics which are object detection, tracking and vehicle counting. Next, the objectives of the project are constructed and the research commences with a thorough literature review obtained from various sources including online journals, books and conference papers on the topics of vehicle detection, tracking and counting in highway

scenes and the latest methods which are currently being deployed. Once enough information has been gathered, a conceptual solution of the entire system is identified including the physical hardware components and sensors which will be used for deployment.

The next phase of the project is the data collection, cleaning and annotation phase. Here, large amounts of images containing cars, motorcycles, buses and trucks captured from a distance approximately 5 meters above the ground are collected from various sources including open-source datasets, footages from CCTV cameras and manually captured images by using a smartphone camera. The vehicle dataset is then cleaned and annotated with the following label maps: car, motorcycle, bus and truck. Once the images have been annotated, a custom augmentation technique is performed on the images with effects such as adding noise, rain, adjusting brightness and applying random rotations in order to increase the quality of the dataset and boost the model's ability to generalize to new scenarios. Next, a custom YOLOv5 object detection algorithm is developed and trained on the captured dataset to detect and classify the four different classes vehicles with a relatively high accuracy score. A DeepSORT object tracker together with a light-weight OSNet Re-Identification (ReID) model is then used for the tracking of detected vehicles across different frames in the video sequence.

Finally, once the custom YOLOv5 vehicle detection algorithm has been trained and integrated with the DeepSORT vehicle tracking algorithm, the last stage of the project involves the development of a unique and efficient vehicle counting method. A reliable vehicle counting method will be implemented by using image processing techniques and will be used for accurately counting the number of vehicles that have crossed the highway in real-time and with a relatively high accuracy score by making

use of a “virtual polygon area” approach. The virtual polygon area will be placed on the highway and used to detect as the vehicle’s center coordinates has entered into the specified region and the vehicle count will be incremented accordingly. Finally, the output from the system will be the vehicle detection results along with the number of vehicles that have crossed the highway shown separately for each vehicle class. The entire system will then be deployed on an Nvidia Jetson Nano edge computing device and its performance will be tested in real world test scenarios. The Figure 3.1 shows the summary of our proposed solution.

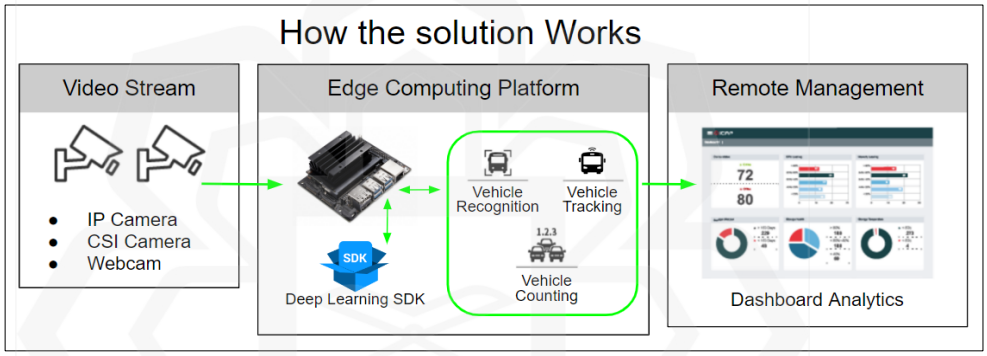


Figure 3.1 Summary of the Proposed Vehicle Counting Solution

The primary focus of this research can be summarized as follows:

1. Generate a new highway vehicle dataset consisting of 10,000 images captured from CCTV point-of-view and perform cleaning, annotation and augmentation.
2. Develop and train a new custom vehicle detection algorithm based on the YOLOv5n to detect as well as classify the four different vehicle classes.
3. Integrate the trained vehicle detection model with a DeepSORT cosine metric learning algorithm to keep track of the detected vehicles across different frames.
4. Develop a reliable and efficient vehicle counting method which can count the number of vehicles in real-time as they cross through the highway.
5. Deploy the entire system on an Nvidia Jetson Nano and run tests to validate the accuracy and performance of the proposed system in real world scenarios.

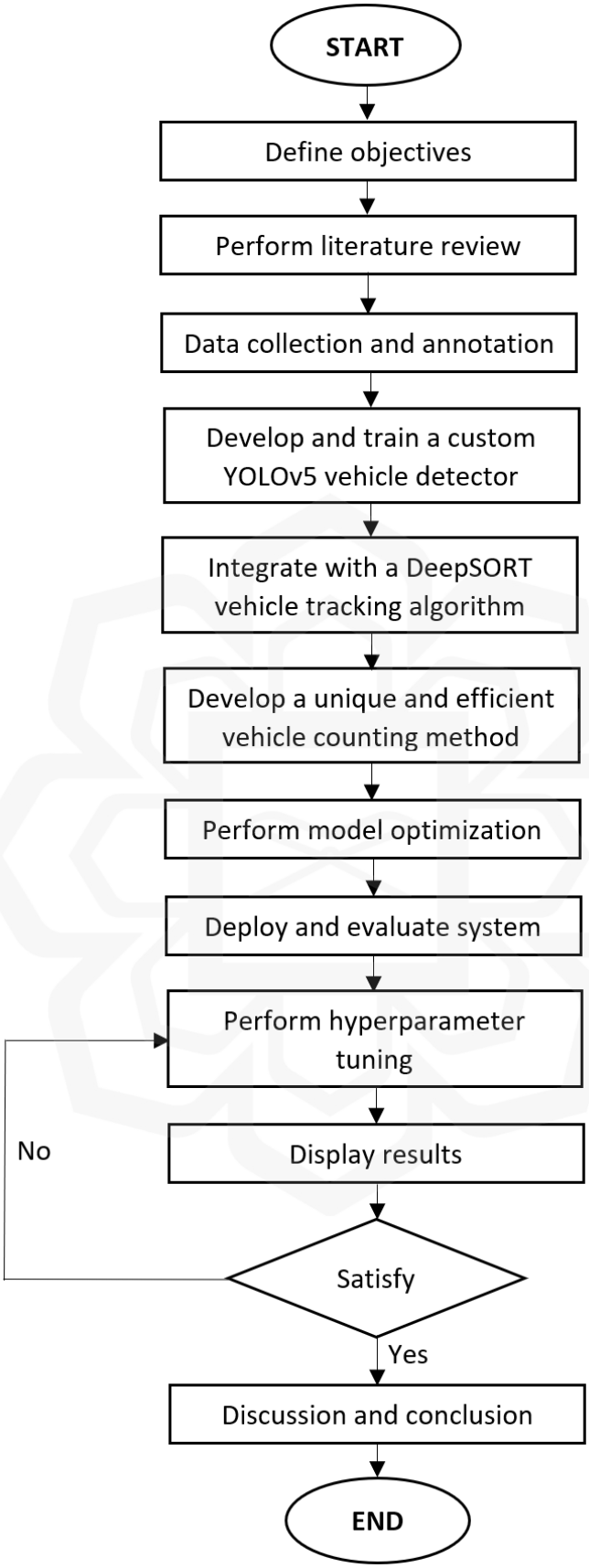


Figure 3.2 Flowchart of Research Methodology

3.3 SYSTEM ARCHITECTURE

Although a vast majority of vehicle counting systems have been implemented in the past, new advancements in the field of AI, object detection and tracking continue to emerge, introducing new features and significant improvements from one another. The Figure 3.3 shows the summary of our proposed vehicle counting system architecture which was developed by utilizing a custom YOLOv5 object detection algorithm along with a robust DeepSORT object tracker for tackling the problem of vehicle counting in highway scenes. The system is divided into three main components which are the object detection which will be trained for vehicle detection and classification by using our own custom generated dataset and architecture, multi-object tracking which will be integrated and used for the vehicle ID assignment and tracking and finally a unique vehicle counter which will be developed by using image processing techniques and used for the counting of vehicles as they pass through a virtual polygon area in the highway.

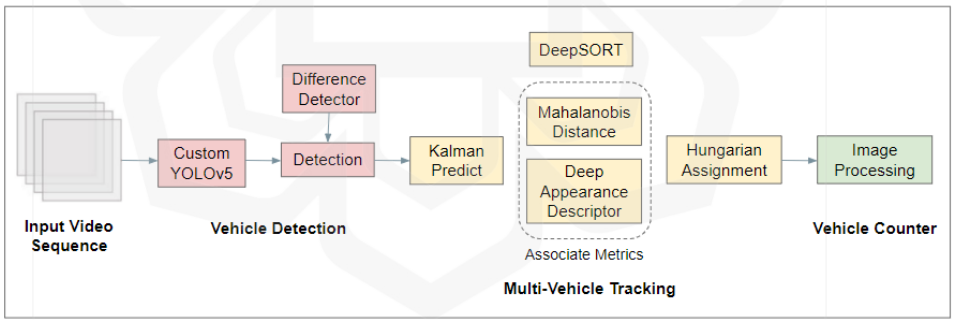


Figure 3.3 Proposed System Architecture

The following is the outline of the proposed vehicle counting system:

a) Input: Video sequence of vehicles on the highway.

b) Output: Total vehicle count for each of the four trained vehicle classes.

c) Main components and their functions

i) Feature Extraction: Convert the live video feed into a sequence of images to allow the algorithm to detect vehicles based on the trained features.

ii) Detection and Classification: Draw a bounding box to pinpoint where exactly the vehicle is located in the image and provide it with a label.

iii) Kalman Filter: Use the available detections of the vehicle along with its previous location to estimate its next location in the sequence of images.

iv) ID Assignment and Tracking: Create a unique ID and track the vehicle as it moves around in different frames in the video sequence.

v) Vehicle Counter: Determine if the vehicle center coordinates has entered into the virtual polygon area and increment the counts accordingly.

The combination of YOLO and DeepSORT object detection and tracking algorithms has already been implemented in the past and was proven to be useful in the problem of people counting in dense supermarket areas (Valencia et al., 2021). In this paper, the researcher makes use of an older version of the YOLO algorithm, known as the Tiny-YOLOv4 trained for the task of people detection and combined it with a DeepSORT object tracking algorithm trained for the task of people tracking which had achieved an average people counting accuracy of 92.94% (Valencia et al., 2021).

In this work, we validate this approach in the vehicle counting problem by utilizing a modified structure of the latest version from the YOLO family, known as the YOLOv5 and codes implemented in the PyTorch framework instead of TensorFlow. After the combination of YOLOv5 and DeepSORT vehicle detection and tracking algorithms, a unique and robust vehicle counter will be implemented and used to detect if a new tracked vehicle has entered into a virtual polygon area and if so then the vehicle count will be incremented accordingly. Finally, the vehicle counting results will be

displayed on a window showing the total vehicle count for each vehicle class separately. The overall summary of the proposed system can be seen in the Figure 3.4.

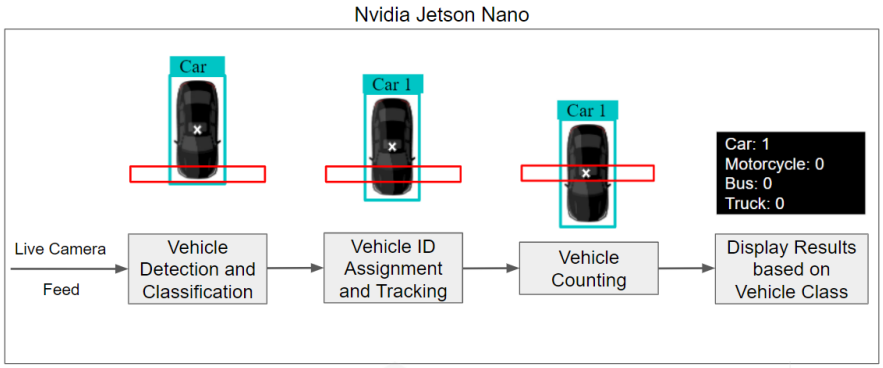


Figure 3.4 Overall Summary of the Proposed System

3.4 DATA COLLECTION AND DESCRIPTION

Data collection refers to the task of gathering essential information on certain variables of interest for your research. It is used to train systems, test hypotheses and evaluate outcomes. Regardless of the field of study, accurate data collection is essential for maintaining the integrity of research, obtaining accurate analytics and drawing valuable conclusions. When working with deep learning systems, there are three main approaches to data collection which are data discovery, data augmentation and data generation (Whang & Lee, 2020). Data discovery refers to the searching of datasets from online sources and existing research works and revealing relevant insights from that data. Data augmentation refers to a set of unique techniques that are applied to the dataset and used to artificially increase its size by adding modified copies in order to improve the model's performance and ability to generalize to different environments. Finally, data generation refers to the extraction of several images from a dataset based on a qualitative study.

Highway traffic images from various sources including open-source datasets, footages from CCTV cameras and manually captured images were the kinds of data that were used in this study. The main open-source datasets that were used included the highway vehicles dataset (Song & Liang, 2019) which is a large dataset consisting of vehicles captured from 23 surveillance cameras located in Hangzhou, China. Another open-source dataset is the MIO-TCD dataset (Z. Luo et al., 2018), this is a very popular traffic dataset that was acquired from cameras deployed all over Canada and the United States. The highway vehicle dataset consists of 3 vehicle classes namely; car, bus & truck meanwhile, the MIO-TCD Dataset consists of 11 classes including pedestrians and bicycles which were all filtered out and cleaned leaving only the labels and images which are fitted with our research study. CCTV camera footages obtained from different locations in Shah Alam, Malaysia were also dissected and used in the following dataset. Finally, manually captured images included hours of video streams which were recorded by using a 12-megapixel smartphone camera operating at 30 fps and 1920*1080 resolution that was situated on top of a pedestrian bridge pointing directly towards a busy highway located in Kuala Lumpur, Malaysia and taken from different angles as illustrated in the Figure 3.5.

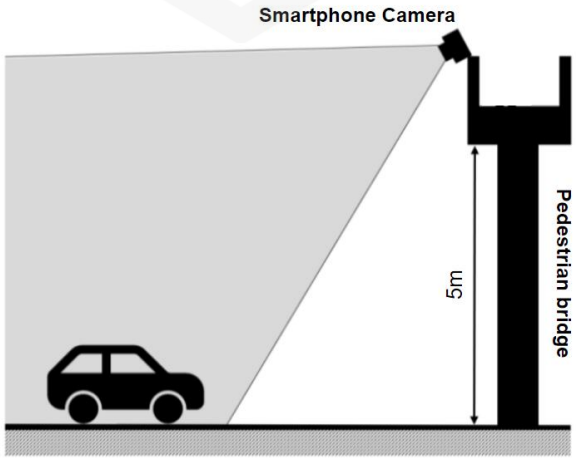


Figure 3.5 Illustration of Data Acquisition Process

The generated vehicles dataset consisted of four different classes of vehicles namely: Car, Motorcycle, Bus and Truck as shown in the Figure 3.6 which were all manually labelled and the annotations were obtained from either the open-source datasets together with the images and filtered out or were manually annotated by using the makesense.ai labelling tool (Skalski, 2019). Makesense.ai is a free and comprehensive online labelling tool which is equipped with advanced AI functionalities that helps in significantly reducing the amount of time required in labelling of object detection data (Skalski, 2019).

Figure 3.6 Sample images for each of the four vehicle classes

The highway vehicle dataset also contained vehicles that have dramatic changes in scale, including far away vehicles as well as nearby ones which were all labelled and this allowed for an improvement in the detection accuracy of smaller vehicle objects. Furthermore, the dataset also consisted of images that were captured from various locations, different timings throughout the day and different weather conditions allowing for a more robust model that is able to generalize well in different environments and weather conditions.

To train the models used in this study, altogether 11,982 manually annotated images were generated for all of the different vehicle classes. All the images are in RGB format and have varying resolutions including standard, HD and Full HD which have all been resized to 416*416 for training. The dataset generated in this study has been published in the following Google Drive link: https://tinyurl.com/yc2eycje. In this dataset, cars accounted for 58.23%, motorcycles accounted for 7.35%, buses accounted for 11.27% and trucks accounted for 23.15%. There are 3.35 annotated instances in each image on average.

The Table 3.1 shows detailed information about the published vehicle dataset along with the number of instances for each of the four vehicle classes in each train, valid and test sets. As can be seen, the number of cars accounted for the highest rate of captured instances, followed by trucks, buses and finally motorcycles which accounted for the least number of instances. Generally, the imbalance in dataset was expected as most of the vehicles that are travelling on the highways were found to be cars that were privately owned and single-unit trucks used for commercial use.

Table 3.1 Vehicle Dataset Information

Subset	Number of Images	Number of Cars	Number of Motorcycles	Number of Buses	Number of Trucks
All	11,982	23,360	1,950	2,975	9,286
Train	8,237	16,352	890	1,382	6,500
Validation	2,350	5,487	60	395	2,653
Test	1,184	2,336	45	198	929

3.4.1 Annotation

Annotation refers to the process of labelling data used for training of deep learning models. In this study, all of the manually captured images were annotated by using the open-source makesense.ai labelling tool which was developed by Piotr Skalsi (Skalski, 2019) and bounding boxes are generated for each vehicle object that is present in the images. There are few principles that need to be followed during the labelling process in order to ensure the quality of the annotated dataset. These principles include that all of the generated bounding boxes must be tightly fitted around the vehicle object, all detected vehicles on the screen must be properly annotated and if a vehicle is partially or fully occluded, then that vehicle must be ignored (Valencia et al., 2021).

Moreover, in order to speed up the annotation process, makesense.ai also includes advanced AI functionalities which consists of an SSD model that has originally been trained on the COCO dataset and that is able to classify over 80 class labels. The SSD model was used during the labelling process and does most of the work in drawing of the bounding boxes around the vehicle objects and suggesting suitable labels, subsequently, manual verification is then required for adjusting the bounding boxes to fit tightly around the vehicles and for verification of labels. The Figure 3.7 shows the makesense.ai labelling tool user interface together with an example generated bounding box displaying the vehicle class, bx & by which represents the center coordinate of the bounding box and bw & bh which represents the width and height of the bounding box respectively. These are all required for generating the annotation files.

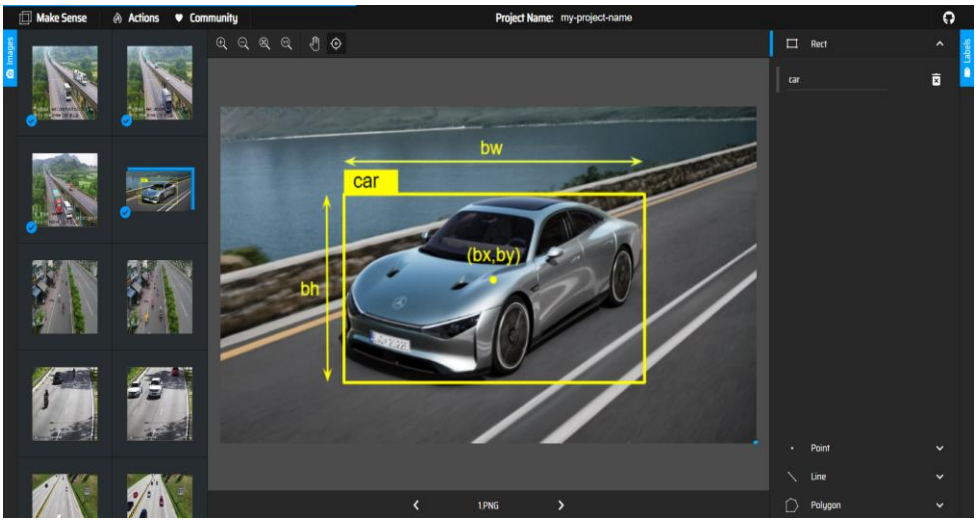


Figure 3.7 Makesense.ai labelling tool User Interface

Once the labelling process has completed, the dataset is then exported and the annotations are saved in a YOLO text format which is a very common format for storing of bounding boxes used for training of object detection models. The bounding box data in the YOLO format consists of the following information: the vehicle class, the center coordinated of the vehicle's bounding box and the normalized value of the bounding box width and height for each detected vehicle as shown in the Figure 3.8. Object classes are zero-indexed (start from 0), therefore the labels 0, 1, 2 and 3 refer to the classes: car, motorcycle, bus and truck respectively. Furthermore, as can be seen in the Figure 3.6, the bounding box coordinates must be normalized in a range between 0 and 1. Hence, the $\boldsymbol{x_{center}}$ and $\boldsymbol{width}$ are divided by the image width, meanwhile the $\boldsymbol{y_{center}}$ and $\boldsymbol{height}$ are divided by the image height resulting in a value between 0 and 1.

<object-class-ID> <X center> <Y center> <Box width> <Box height>

0	0.383	0.439	0.183	0.628
2	0.507	0.454	0.191	0.713

Figure 3.8 YOLO Text Annotation File Format

3.4.2 Splitting Data into Train, Valid and Test Set

Once the annotation files have been generated, the next step is to separate the dataset into train, valid and test splits. Train-valid-test split is a commonly used technique in machine learning for building of reliable and robust models by evaluating their performance in an unbiased manner. There is no one optimal split ratio that is suitable for every situation and one has to come up with a ratio that suits his requitements and meets the model's needs. In our case, we split the dataset into 0.7, 0.2 and 0.1 ratios for train, valid and test sets respectively following the general standard principle.

The training set is a set of images that are used for training the model to learn the hidden features and patterns from the dataset. It usually includes a diversified set of inputs so that the model is able to learn from different kinds of scenarios and has the ability to generalize well with new vehicle data samples which it has never seen before. In each training epoch, the same training set is fed into the neural network architecture repeatedly allowing for the model to continue to learn the different unique features present in the dataset. The validation set on the other hand is a separate set of images which are used for validating the model's performance during the training process. It provides unbiased evaluation of the model and gives us information which can help in tuning the model's hyperparameters and configurations accordingly.

The main idea behind the validation set is to prevent the model from overfitting where it learns the details and noise of the training samples so much so that it is unable to generalize well and make accurate predictions on new never before seen data. Finally, the test set consists of a separate set of images which are used to test the model's performance after the model had already completed training. It provides an unbiased model performance metrics in terms of accuracy, precision, recall and mAP. The codes

written for randomly splitting the dataset into train, valid and test subset has been open-sources and is available in the following link: https://github.com/abuelgasimsaadeldin/dataset-preparation/blob/main/train-test-valid-split.py

3.4.3 Creating the data.yaml file

The YOLOv5 algorithm accesses the split datasets and uses them as input for training, validation and testing through a yaml file which is commonly called as `data.yaml`. The `data.yaml` file contains a summary information of the dataset which includes the path locations of the train, valid and test images, the number of custom classes that are present in the dataset and the names of each class. The vehicle dataset `data.yaml` file that has been created and used in this study is presented in the Figure 3.9.

```yaml
# Path directory to train, valid and test splits
train: ../Highway_Vehicle_Dataset/images/train
val: ../Highway_Vehicle_Dataset/images/valid
test: ../Highway_Vehicle_Dataset/images/test

# Number of Classes
nc: 4

# Class Names
names: ['bus', 'car', 'motorcycle', 'truck']
```

Figure 3.9 The data.yaml file created for vehicle dataset

3.5 YOLOV5 VEHICLE DETECTION

In this study, a CNN based object detection algorithm known as YOLO (You Only Look Once) is implemented and used for the vehicle detection. YOLO is a very popular object detection algorithm which is able to detect as well as recognize various objects in an image and is able to do so in real-time. The YOLO algorithm works by firstly dividing the input image into grid cells which have pre-defined sizes. Each cell is then

responsible for generating bounding boxes and confidence scores of all objects that fall within that grid cell. Each cell also predicts a vector class which is composed of five elements as represented in the equation (3.1) where $\boldsymbol{y}$ is the grid cell, $\boldsymbol{p_c}$ is the confidence score, $\boldsymbol{b_x}, \boldsymbol{b_y}$ are the midpoints of the bounding box, $\boldsymbol{b_h}, \boldsymbol{b_w}$ are the width and height of the bounding box and lastly $\boldsymbol{c}$ is the corresponding label of the object.

$$\boldsymbol{y} = \begin{bmatrix} \boldsymbol{p_c} \\ \boldsymbol{b_x} \\ \boldsymbol{b_y} \\ \boldsymbol{b_h} \\ \boldsymbol{b_w} \\ \boldsymbol{c} \end{bmatrix} \tag{3.1}$$

Over the past decade, the YOLO algorithm has witnessed enormous growth and there are currently different model variations and versions of YOLO that have been released. As of writing this thesis, YOLOv5 is the latest version of the YOLO family of object detectors to be released (Jocher et al., 2020), this however is still considered to be a debatable statement as many researchers still tend to believe that the YOLOv4 is the last version due to the change in authors and a failure to submit a detailed research paper along with the study (Meel, 2021).

The YOLOv1 to YOLOv3 were all released by the same author Joseph Redmon along with his advisor Ali Farhadi. These versions undergo large improvements from one another and had vastly different features, with each having its own published research paper which shows a significant improvement from the last (Meel, 2021). The YOLOv4 was then researched and introduced by Alexey Bochkovskiy who had continued the legacy since Joseph Redmon stopped his computer vision research due to ethical concerns. Shortly after, the YOLOv5 was released by Glenn Jocher, Founder &

CEO at Ultralytics and this quickly gained traction amongst the computer vision community.

The YOLOv5 objection detection algorithm is written in Python and makes use of the PyTorch deep learning framework as opposed to the Darknet framework which was used by all of the other previous versions of YOLO. Due to the following changes in framework along with other improvements, the models built by using the YOLOv5 have shown remarkable speeds while training and are significantly smaller in size making it more accessible to edge devices and deployment in real-world applications. The YOLOv5 provides five different scales of their model and they are the Nano, Small, Medium, Large and Extra-large as shown in the Figure 3.10. The difference between each one of these scales is merely in the depth and width of the models which has been expanded, meaning that the overall structure of the model remains the same, however, the size and complexity of the model increases.

The Large and Extra-large scales of the YOLOv5 are suitable when there are no limitations in computation and the highest possible accuracy score is desired. On the other hand, smaller models such as the Nano and Small are preferred when deploying the models on embedded devices with low computing power but also with a compromise in accuracy. In our experiment, we utilize the smallest, fastest base model, the YOLOv5n and modify the structure by adding extra layers to improve the detection accuracy of smaller vehicles and we use transfer learning techniques to change the last layer to output the four vehicle classes, i.e., car, motorcycle, bus and truck instead of the original 80 classes which it has been trained on using the MS COCO dataset.

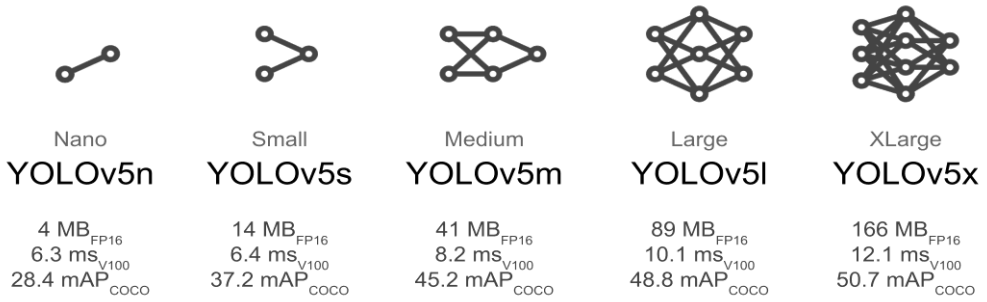


Figure 3.10 The Different YOLOv5 Model Scales

3.5.1 CNN Model Architecture

The YOLOv5 has a deep learning Convolutional Neural Network (CNN) architecture which is based on four parts: the input, backbone, neck and head or output detection layer as can be seen in the Figure 3.11. In the input layer, a Mosaic data augmentation is applied to the training images which randomly selects four training samples and scales, cuts and combines them into one image with certain aspect ratios, thereby increasing the model's ability to detect objects of smaller scales than usual. Additionally, when calculating the batch normalization, the data of four training samples are calculated at the same time, thereby, allowing for a smaller batch size to be set during the training which is suitable especially when training on single GPUs.

After the Mosaic data augmentation, a function is then applied to adjust the anchor boxes and the picture adaptivity, the YOLOv5 does this automatically and learns the anchor boxes from the training set improving the model's ability to be able to detect overlapping objects. The YOLOv5 network then uniformly modifies the dataset images into a size of 416x416, thereby improving the inference and detection speeds without losing much detail.

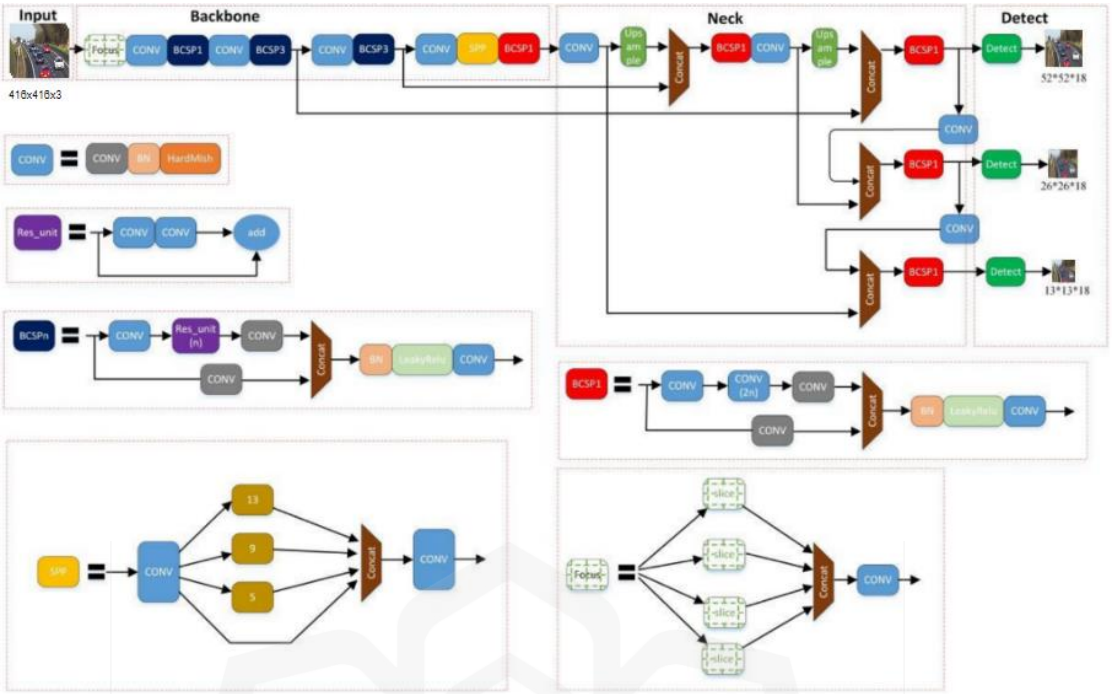


Figure 3.11 YOLOv5 Model Architecture Diagram

The backbone layer of the YOLOv5 then takes the input image and extracts feature maps from it, containing contextual information about the objects present. This is a crucial step in object detection algorithms as it extracts unique features from the input image which are used to identify that object. The backbone network contains a focus module, this is used for slicing of the input image. Let's assume that we have an input image with the size of 416x416x3 which is placed into the focus module, the output will then be an image of size 208x208x12. After the slicing operation is complete, the image is then fed into a convolutional operation consisting of 32 convolutional layers and the result is a feature map with the size of 208x208x32.

The backbone network also contains a Bottleneck Cross Stage Partial (BCSP) module which is used for improving the learning ability of the CNN by making the model smaller while still maintaining the same accuracy score which is important especially when deploying the models on AI edge systems. It also contains a Special

Pyramid Pooling (SPP) module which is used for increasing the receptive field of the model while still maintaining the same calculation speed. The SPP module consists of three pooling cores of sizes: 13x13, 5x5 and 9x9 respectively. The last core does not have a pooling layer and is directly joined with the Concat module and passes through the convolutional layer output. Due to the BCSP and SPP modules, the YOLOv5 is able to have improved inference as well as detection speeds.

The neck network of the YOLOv5 is placed between the head and backbone layers and adopts a Feature Pyramid Network (FPN) and Path Aggregation Network (PAN) structure. The neck network is mainly used for generating feature pyramids by taking a single-scale image of an arbitrary size as input and outputting proportionally sized feature maps at multiple levels as shown in the Figure 3.12. The construction of the FPN involves a bottom-up and a top-bottom pathway, it also includes four connection layers, four convolution layers and five Cross Stage Partial (CSP) layers. This is used for enhancing the model's ability to detect objects at different scales and also recognize different sizes and standards of the same object. It is also used to speed up the transmission of feature information and feature fusion. Furthermore, as can be seen from the Figure, after a couple of down samples, the FPN module outputs images of the following sizes 52x52, 26x26 and 13x13 respectively.

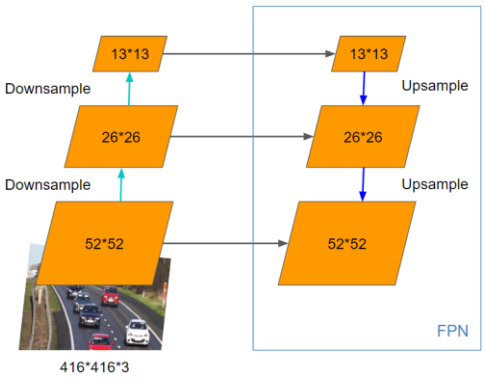


Figure 3.12 YOLOv5 Neck Module

Finally, the output detection layer of the YOLOv5 also known as the prediction layer makes use of a Generalized Intersection over Union (GIoU) as its loss function for the bounding boxes. This is a commonly used evaluation indicator in object detection algorithms and works by calculating the difference between the predicted and ground truth bounding box. Three outputs are then produced with different sizes depending on the input image size, in our case, the following sizes of 20*20*255, 40*40*255 and 80*80*255 are outputted. This is used for the detection of different vehicle sizes.

3.5.1.1 Custom Model Architecture

We now need to write the model configuration for our custom vehicle detector which should include the input, backbone, neck and output detection layer. Similarly with the `data.yaml` which was used by algorithm to locate the dataset path, the YOLOv5 model architecture is also written in a yaml file format which will be read and generated by the `train.py` script on PyTorch. The YOLOv5 comes with five default model architectures which are the: YOLOv5n, YOLOv5s, YOLOv5m, YOLOv5l and YOLOv5x. The default model architectures can be configured by increasing the number of hidden layers, adding more neurons, performing batch normalization and weight initialization. This however is rarely required in most cases as the default architectures were designed following best design principles are able to provide decent results.

The purpose of this study is to a develop and train an accurate and light-weight YOLOv5 vehicle detection model which is able to detect vehicles captured from a distance approximately 5 meters above the ground. Due to the vehicles being relatively small in size and the generated dataset containing vehicles that have dramatic changes in scale, it can be difficult for the YOLOv5 to be able to detect far away vehicles by

default using the smallest YOLOv5n model architecture. Therefore, addition of modules in order to increase the detection accuracy of smaller vehicles objects and increase the speed of inference of the model is necessary. Thus, we have added an attention mechanism layer to the YOLOv5n architecture along with some additional up-sampling and down-sampling layer to the feature map in order to further increase the vehicle detection accuracy score. The attention mechanism layer is a Convolutional Block Attention Module (CBAM) which replaces the original CONV module allowing for a more detailed information about passing vehicles by reducing the attention from roads and other complex backgrounds. The Figure 3.13 presents the improved model architecture where the rows 12, 24-27 and 47-49 represent the replaced attention mechanism and additional small target detection modules respectively. The following model configuration was used for training and was named `custom_yolov5n.yaml`.

```yaml
nc: 4  # number of vehicle classes
depth_multiple: 0.33
width_multiple: 0.25

anchors:
  - [10,13, 16,30, 33,23]
  - [30,61, 62,45, 59,119]
  - [116,90, 156,198, 373,326]

backbone:
  [[-1, 1, Focus, [64, 3]],
   [-1, 1, Conv_CBAM, [128, 3, 2]],
   [-1, 3, C3, [128]],
   [-1, 1, Conv, [256, 3, 2]],
   [-1, 6, C3, [256]],
   [-1, 1, Conv, [512, 3, 2]],
   [-1, 9, C3, [512]],
   [-1, 1, Conv, [1024, 3, 2]],
   [-1, 3, C3, [1024]],
   [-1, 1, SPP, [1024, 5]],
  ]

neck:
  [[-1, 1, Conv, [768, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 8], 1, Concat, [1]],
   [-1, 3, C3, [768, False]],

   [-1, 1, Conv, [512, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 6], 1, Concat, [1]],
   [-1, 3, C3, [512, False]],

   [-1, 1, Conv, [256, 1, 1]],
   [-1, 1, nn.Upsample, [None, 2, 'nearest']],
   [[-1, 4], 1, Concat, [1]],
   [-1, 3, C3, [256, False]],

   [-1, 1, Conv, [256, 3, 2]],
   [[-1, 14], 1, Concat, [1]],
   [-1, 3, C3, [512, False]],

   [-1, 1, Conv, [512, 3, 2]],
   [[-1, 10], 1, Concat, [1]],
   [-1, 3, C3, [1024, False]],

   [ -1, 1, Conv, [ 768, 3, 2 ] ],
   [ [ -1, 12 ], 1, Concat, [ 1 ] ],
   [ -1, 3, C3, [ 1024, False ] ],
  ]

head:
   [[17, 20, 23], 1, Detect, [nc, anchors]]
```

Figure 3.13 Proposed Vehicle Detection Deep Neural Network

In order to accurately detect the small-sized vehicles on roadways, after the focus module in the backbone network, an attention mechanism CONV_CBAM module is added which replaces the original CONV module. By doing so, the algorithm is now able to obtain more detailed information about passing vehicles and reduce the inference from the roads and other complex backgrounds. The Convolutional Block Attention Module (CBAM) pays attention to the channel information, which solves the loss problem caused by the different weights in the feature graph. The CBAM module works by taking the output of the channel attention module as input, then it goes through two pooling operations and a convolution operation with a convolution kernel of 7x7. Finally, a feature graph with the size of HxWx2 is obtained. The main innovation in this network is in the addition of the CBAM and additional up-sampling and down-sampling layers which allowed for the model to learn the spatial attention features of the vehicles through the relationship between the channel and space.

3.5.2 Model Training

In general, model training in deep learning refers to the process of iteration from the forward propagation to the backward propagation in a neural network in order to try and fit the best combination of weights and biases to an algorithm by minimizing the loss function and achieving global convergence. After the preparation of our custom dataset, the dataset summary information (`data.yaml`) and the custom model architecture (`custom_yolov5n.yaml`), we are now able to start training our vehicle detection model. The YOLOv5 provides a default training script which allows for easy hyperparameters adjustments in order to adhere to the custom dataset that is being trained. The Table 3.2 shows the main hyperparameters adjustments that have been used for the successful training of our custom vehicle detection model, their roles and values.

Table 3.2 Tuned Vehicle Detection Hyperparameters

Hyperparameter	Role
Image size	Used to define the input size of the image. The original dataset contains images with various sizes which are all resized to 416 x 416 to allow a for faster training time without losing much detail.
Batch size	Used to determine the no. of samples passed to the network. The train set contains 8237 images, with a batch size of 16, the number of batches used for training per epoch will be 515 batches.
Epochs	Define the number of epochs. This represents the number of times the model trains on the inputs and updates the weights to get closer to the ground truth. The number of epochs is set to 300.
Data	Specify the file containing summary of the dataset. Based on the previously defined `data.yaml`, this will be used by the algorithm for accessing the train, valid and test directories.
Configuration	Specify our model configuration. Based on the custom YOLOv5 model architecture that was previously defined, this will be passed for compiling and building of our architecture for training.
Weights	Specify weights to be used. The vehicle model will be trained using pretrained weights as the starting point and using randomly initialized weights and a comparison will be made.

In the following study, we will be training two vehicle detection models, the first will be by using transfer learning technique where we will be using pretrained weights which have originally been trained on the COCO dataset as the starting point, using this method, we can leverage knowledge that has already been learnt from that task to solve our vehicle detection problem. Utilizing transfer learning helps in achieving optimal performance, allows for faster training times and provides more accurate vehicle detection results (MacHiraju et al., 2021). The second method will be by training our model from scratch, here, instead of starting from pretrained weights, we start the training with randomly initialized weights allowing for a new custom trained model to be generated for our specific task. The Figure 3.14 shows an illustration

of training the model from scratch and by using transfer learning technique. The training process for the vehicle detection requires heavy computation and therefore in our case an Nvidia RTX 3060 laptop GPU was used for the training in order to speed up the learning operation by utilizing multiple cores and large memory bandwidth for high-speed parallel processing.

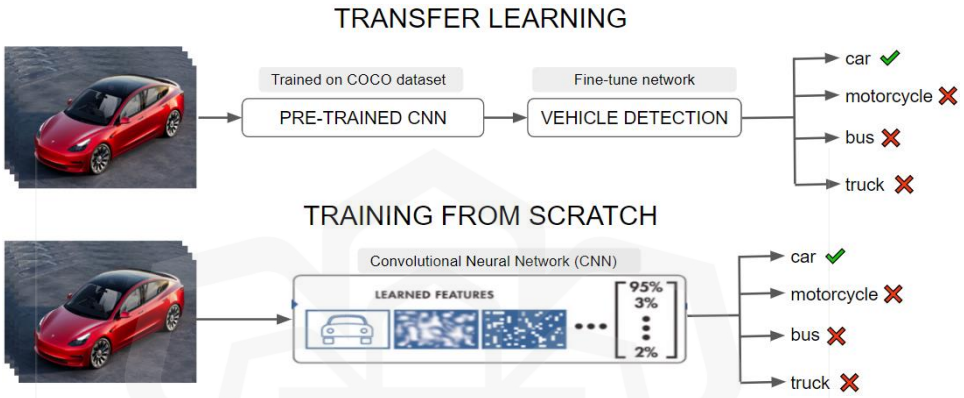


Figure 3.14 Transfer Learning vs Training from Scratch

3.5.3 Model Optimization

Because the size of vehicles captured from various cameras located approximately 5 meters above the ground can be relatively small in size, it can be difficult for the YOLOv5 to be able to accurately detect the individual vehicles by default and with a relatively high accuracy score. Thus, in order to improve the vehicle detection accuracy of the model, we apply several augmentation techniques, add an attention mechanism layer and some up-sampling and down-sampling layer in order to further increase the detection accuracy and performance of the model.

The custom augmentation that has been applied to our dataset included image rotations, adding noise, rain and varying brightness in order to increase the quality of the dataset and allow for a highly generalizable model which is able to adapt to different environments and weather conditions. The Figure 3.15 shows a sample input image

which has gone through the different custom augmentation techniques and the results are also displayed. The codes written for the vehicle augmentation has also been open-sources and is available in the following link: https://github.com/abuelgasimsaadeldin/dataset-preparation/blob/main/ imgAug.py.

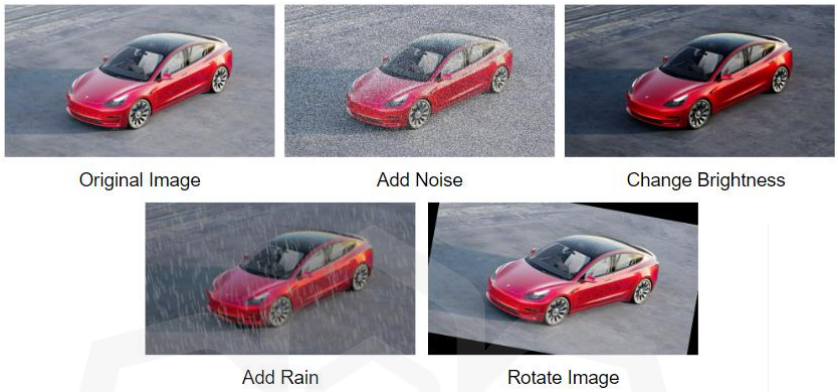


Figure 3.15 Different Augmentations techniques applied to Sample Image

Furthermore, due to the size of the vehicles being relatively small and the dataset containing vehicles that have dramatic changes in scale, it can be difficult for the YOLOv5 to be able to detect far away vehicles by default using the smallest YOLOv5n model architecture. Therefore, addition of modules in order to further increase the detection accuracy of small vehicle objects and improve the speed of inference of the model is required. Thus, we have added an attention mechanism layer to the YOLOv5n model architecture along with some additional up-sampling and down-sampling layers to the feature map in order to further improve the model's accuracy on smaller vehicle detections. The attention mechanism layer is a Convolutional Block Attention Module (CBAM) which replaces the original CONV module, allowing for more detailed information about passing vehicles and reduces the inference from roads and other complex backgrounds. Meanwhile, the up-sampling and down-sampling layers enhance the model's ability to detect vehicles at different scales and also recognize different sizes and standards of the same vehicle.

3.6 DEEPSORT VEHICLE TRACKING

Once the vehicles have been detected by using the custom YOLOv5 algorithm, the vehicle features are then extracted and a DeepSORT multi-object tracking algorithm is used for the matching of features with the other video frames in order to achieve a correlation between the same vehicle and other similar vehicles. The DeepSORT algorithm works by using a combination of Kalman Filter and Hungarian algorithm for the tracking. Here, the Kalman Filter is used for making a prediction of the current state of the vehicle based on some previous value along with providing the uncertainties of that prediction by using the state equations shown in (3.2) and (3.3). Once the predicted measurements are obtained, they are then corrected by using the measurement equations shown in (3.4), (3.5) and (3.6) and the optimal state estimate of the vehicle is obtained. This is a recursive method which runs in real-time and only requires the present input value measured and previously calculated state along with the uncertainty matrix in order to derive the correct location of the vehicle.

Initial Estimate

$$Initial\ Estimate: \hat{x}_{0,0}, P_{0,0} \tag{3.2}$$

Time Update ("Predict")

$$1.\ Extrapolate\ the\ state: \hat{x}_{n+1,n} = F\hat{x}_{n,n} + Gu_n \tag{3.3}$$

$$2.\ Extrapolate\ Uncertainty: P_{n+1,n} = FP_{n,n}F^T + Q \tag{3.4}$$

Measurement Update ("Correct")

$$1.\ Compute\ the\ Kalman\ Gain: K_n = P_{n,n-1}H^T(HP_{n,n-1}H^T + R_n)^{-1} \tag{3.5}$$

$$2.\ Update\ estimate\ with\ measurement: \hat{x}_{n,n} = \hat{x}_{n,n-1} + K_n(z_n - H\hat{x}_{n,n-1}) \tag{3.6}$$

$$3.\ Update\ estimate\ uncertainty: P_{n,n} = (1 - K_nH)P_{n,n-1}(1 - K_nH)^T + K_nR_nK_n^T \tag{3.7}$$

Once the vehicle location has been obtained by using the state and measurement equations, a Hungarian algorithm is then used for the vehicle association and ID attribution, assigning a unique identification to the vehicle and identifying if the vehicle present in the current frame is the same as that observed in the previous frame. By utilizing a CNN, the vehicle tracker is able to achieve great robustness against object misses and occlusion, maintain the same ID assignment and also preserve the trackers ability to quickly implement to online and real-time scenarios (Mandal & Adu-Gyamfi, 2019). In our case, the purpose of this study is to be able to keep track of the detected vehicles and maintain their ID assignment throughout the video sequence and to accomplish this, a trained Re-Identification (ReID) model is required. Re-Identification in object tracking refers to the task of comparing between two images and determining if the object present in both of those images are the same or not.

Throughout the years there have been many MOT challenges which have been released with the aim of identifying the best pre-trained ReID model which is able to adapt to various scenarios and objects. The MOT challenge consists of difficult sequences of people Re-Identification with ground truths and a common evaluation tool providing speed, recall and precision measures for an easy way to compare state-of-the-art ReID models (Dendorfer et al., 2020). The largest and most popular database for training ReID models is the Market 1501 which consists of 32,668 pedestrians in supermarkets captured from six different camera angles (Zheng et al., 2015). For our vehicle tracking implementation, we employed an Omni-Scale Network (OSNet) ReID feature extractor which has originally been trained on the Market 1501 dataset and had achieved an average mAP score of 84.9%. The OSNet ReID model will be used in our system for extracting better representational feature vectors for associating vehicles detected in highways.

OSNet specializes in multi-scale predictions and it designs its network with depth-wise separable convolutions which is essential in the development of advanced MOT algorithms (Paik & Kim, 2022). Furthermore, the combination of DeepSORT and OSNet ReID models are light-weight allowing for a fast inference time and the ability to be deployed on edge-computing devices. The DeepSORT is a fast single-shot object tracker meanwhile the OSNet is an efficient ReID model which is able to adapt to various object and scenarios. The overall flow of data computation for the vehicle tracking is described as follows. Firstly, the DeepSORT object tracker tracks the detected vehicles by making use of the Kalman Filter to make prediction of the current state of the vehicle based on some previous value, it then uses the measurement obtained by using sensor data and updates the prediction to obtain the optimal state estimate of the vehicle as illustrated in the Figure 3.16. During the tracking process, ID switches may occur from time to time due to various reasons including occlusion and illumination variation which will be supplemented in the counting process. Furthermore, after the tracking process, representational features are extracted by using the OSNet ReID model, this also helps in significantly reducing the amount of ID switches and further improves the vehicle tracking performance as opposed to only using the DeepSORT model for the tracking.

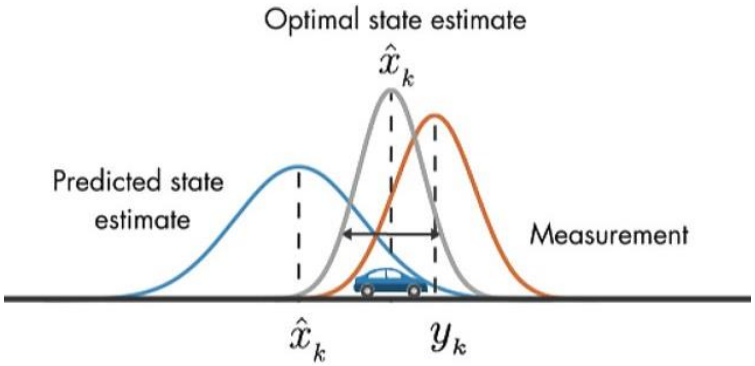


Figure 3.16 The cycle of a Kalman Filter

3.7 VEHICLE COUNTING METHOD

Due to the very similar features observed in some vehicles, the assigned vehicle tracking ID may jump from time to time even when using an optimum trained OSNet ReID model for the tracking and this may result in errors in the quantitative statistics. Therefore, in order to increase the robustness of the vehicle counter by not relying solely on the vehicle tracker and ID assignment for the counting, we introduced a "virtual polygon area" in order to accurately predict the total number of vehicles that have crossed through the highway and their type with a relatively high accuracy score. The virtual polygon area works by dividing the scenes into two areas, Zone 1 and Zone 2 as can be seen in the Figure 3.17. Zone 1 refers to the region that is located outside of the virtual polygon area, whereas, Zone 2 refers to the region which is located within the specified polygon area. The vehicle counting is performed once the vehicle has crossed the highway and its center coordinates enters from Zone 1 into Zone 2 and the vehicle ID assignment remains unique.

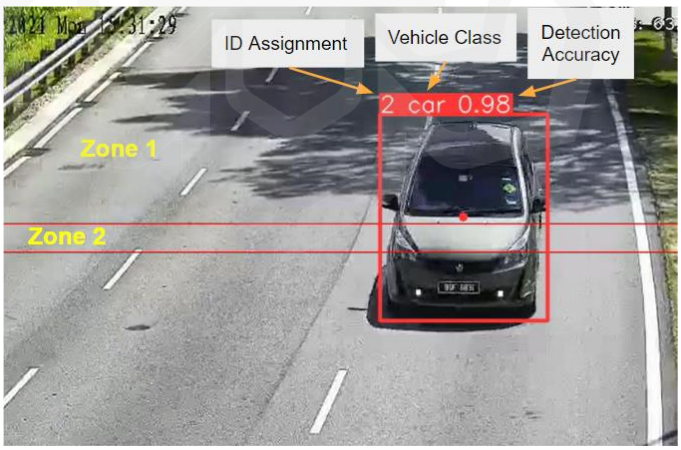


Figure 3.17 Virtual Polygon Area used for Counting

For example, let us assume that our vehicle detection model has detected a vehicle and it is currently being tracked with a given ID assignment of 1, now due to the illumination variation caused by changes in the background scenery, the ID

assignment of the same vehicle switches and now the vehicle has an ID assignment of 2 as observed in the Figure 3.15. If we rely solely on the vehicle tracker and ID assignment for the vehicle counting, this would have resulted in an inaccurate vehicle count, however, by making use of the virtual polygon area along with the vehicle tracking ID assignment for the counting, we are able to solve these problems and obtain an accurate prediction of the total vehicle count. As long as the assigned vehicle ID remains the same within the Zone 2 region, the vehicle counter will only be incremented by 1 for that specified vehicle class.

The virtual polygon area is defined at the start of the counting process by the user, even before the vehicle detection and tracking process is performed, the user selects four points based on the camera view and angle. These points will then be used to draw the virtual polygon area on the image window and perform the vehicle counts. The Figure 3.18 illustrates six different virtual polygon areas which are placed by the user in different camera locations and used to obtain the vehicle counts in highway scenes. Any vehicle that travels across that highway and its center coordinates entered into the virtual polygon area was counted and appended to the total vehicle counts for its respective category.

Figure 3.18 Different Highway Scenes with Virtual Polygon Area

3.8 EXPERIMENTAL SETUP

The Jetson Nano edge-computing device based on an ARM architecture and produced by Nvidia was applied in this study for the testing of our vehicle detection, tracking and counting system. The main goal of this study was to develop a vehicle counting system which was light-weight, reliable and provides accurate vehicle counting results. The device is small, approximately 70 x 45 mm in size and brings in the performance of an efficient computer to the edge by use of a Single Board Computer (SBC). The edge-computing device is equipped with a Quad-core ARM Cortex-A57 CPU, 128 Nvidia CUDA Cores and 4GB of RAM with 64-bit LPDDR4 (NVidia Corporation, 2019).

A Logitech C922 Pro HD Webcam, a 7-inch 1024 x 600 LCD Display and a portable 10,000mAh Mi Power bank were added to the device in order to form an edge-computing system platform as shown in the Figure 3.19. The edge-computing platform was deployed at a busy highway located at Kuala Lumpur, Malaysia for testing of the vehicle counting system. The decentralized character of edge platforms allows for it to have highly reliable infrastructures and due to the processing of data locally, this also allows for offline capabilities. The alternative would be to continuously stream video data to the cloud for processing which is not desirable from a privacy point of view.

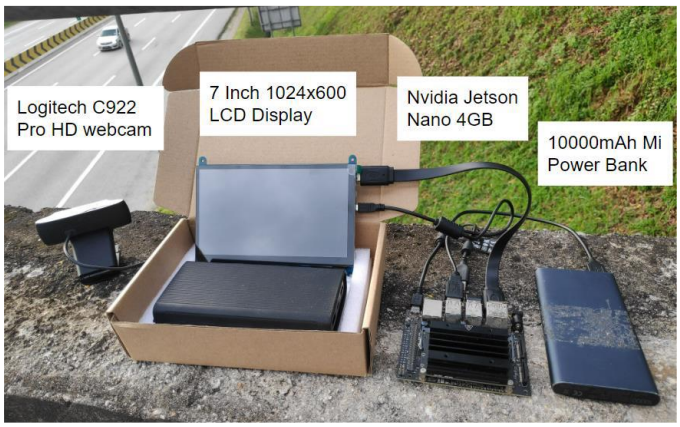


Figure 3.19 Edge-computing System Platform

In order to verify the performance of the vehicle counting system in real-world test scenarios, we deployed the vehicle detection and tracking models on an Nvidia Jetson Nano and made use of its hardware performance, especially the deep learning accelerator engine for running of the model inference. The edge-computing system platform was deployed at the Genting Sempah Highway, Malaysia and the setup was situated at a pedestrian bridge located approximately 5 meters above the ground. The Figure 3.20 illustrates the vehicle counting system flowchart that was used for counting the vehicles on the highway scene.

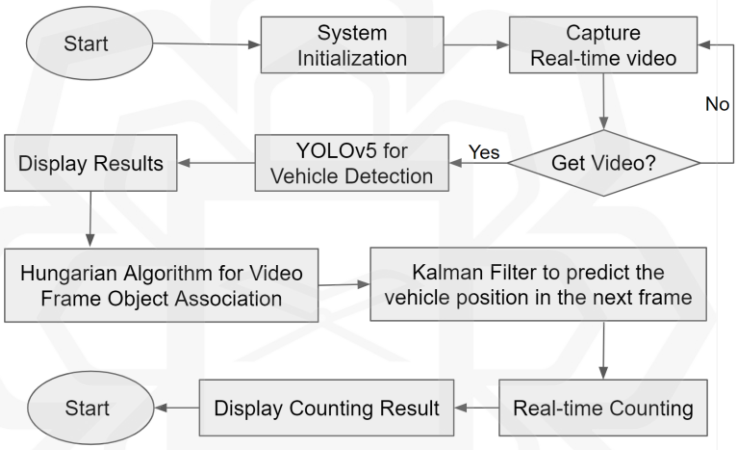


Figure 3.20 Vehicle Counting System Flowchart

The vehicle counting system flowchart works as follows, the Nvidia Jetson Nano firstly obtains the video stream of vehicles from a camera that is pointing directly towards the busy highway, in our case we utilized a Logitech High-definition Pro HD Webcam for obtaining the video streams, however, other sources including remote CCTV or CSI cameras is also possible and can be selected via the command line. Once the video streams have been obtained, the information is then transmitted to the RAM of the edge-computing device. The Jetson Nano then uses that information as input and runs inference of the vehicle detection and tracking models while the Nvidia CPU controls the modules such as the CUDA and Tensor cores using heterogenous parallel

computing in order to accelerate the model by hardware. Finally, the counting is performed by using the “virtual polygon area” and the output from the system is the vehicle detection results as well as the number of vehicles that have crossed the highway shown separately for each vehicle class which are all displayed on the 7-inch LCD display.

3.9 SUMMARY

This chapter presented the methodology of our proposed vehicle counting system and explained how it has been developed from data collection up to deployment and describes the environmental setup that was used for testing of the system. It included detailed description about the generated dataset, this was followed by an elaboration on how the vehicle detection model was developed and trained by using the YOLOv5n model architecture and the main innovation that was made to the network which allowed for it to learn the spatial attention features of the vehicles and improved the detection accuracy of smaller vehicle objects. Next, it described how the trained vehicle detection model was integrated with a robust DeepSORT vehicle tracking algorithm and a light-weight OSNet ReID model for the tracking of vehicles across the highway scenes. Finally, the uniquely developed vehicle counting method for counting the number of vehicles was discussed.

CHAPTER FOUR

RESULTS AND DISCUSSION

4.1 INTRODUCTION

This chapter presents the performance and evaluation of our proposed vehicle counting system which was developed by utilizing a custom vehicle detection algorithm and combined with a DeepSORT vehicle tracking algorithm for the detection, tracking and counting of vehicles in highway scenes. It analyzes the performance metrics that were used to determine the vehicle detection accuracy score and evaluates the efficiency as well as accuracy of the developed vehicle counting method. Furthermore, this chapter also discusses on the vehicle counting experiments which have been conducted by utilizing the edge-computing system platform using live camera inference obtained by using the Logitech Pro HD Webcam as well as using real-world highway surveillance videos obtained from YouTube. Finally, the results of our vehicle detection model and vehicle counting accuracy are presented, discussed and compared with other similar systems.

4.2 TRAINING RESULTS

The vehicle detection model training was performed by utilizing 8,237 images and 2,350 images were used for validation of the model's performance. Several model variations were trained with the goal of identifying the best model which was able to obtain the highest possible accuracy score. The performance of the vehicle detection model was evaluated based on the loss function curve as well as the Mean Average Precision, mAP@.5. Data augmentation was applied to the training images with the aim

of improving the model's ability to generalize to new scenarios and environments, which resulted in an increased in the number of training samples to 11,597 images by applying random image rotations, adding noise, rain and varying brightness, while the validation and test sets remained unchanged.

The mean average precision (mAP) is a popular evaluation metric in object detection and was used to evaluate our vehicle detection model performance. It makes use of the Confusion Matrix, Intersection over Union (IoU), Recall and Precision for the calculation. In order to obtain the mAP score, the vehicle predictions are firstly generated by the model and are converted into class labels, the Confusion Matrix is then generated containing the values of TP, FP, TN and FN. The Precision and Recall are then calculated by using the equations shown in (4.1) and (4.2) respectively. Finally, the average precision is measured by calculating the area under the Precision-Recall curve and the mAP is calculated by finding the average precision for each class of vehicle and then averaging by the total number of vehicle classes. The mAP score is a value between 0 to 1, with the higher the value, the more accurate the model predictions.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{4.1}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{4.2}$$

The Table 4.1 presents the model training results obtained after the successful training of four different models. The first model was by trained by utilizing the default YOLOv5n model architecture and the original dataset, which had obtained an average mAP@.5 score of 93.2% after 300 epochs of training. The second model was trained by utilizing the default YOLOv5n model architecture and applying data augmentation

to the training images which helped to improve the vehicle detection model performance and increased the mAP@.5 score to 93.6%. As for the third model, we applied data augmentation to the training images as well as modified the default YOLOv5n model architecture by adding a small object detection layer and a Convolutional Block Attention Module (CBAM) which helped in improving the vehicle detection accuracy of smaller vehicle objects and obtained an average mAP@.5 score of 95.7%. Finally, the last model was trained by utilizing the same modified YOLOv5n model architecture and the original dataset without applying any augmentations and had achieved the highest average mAP@.5 score which is 96.1%.

Table 4.1 Trained Vehicle Detection Models

Data Augmentation	Small Object Detection Layer	CBAM	mAP@.5	Recall	Epochs
			93.2	89.9	300
√			93.6	90.2	300
√	√	√	95.8	92.0	300
	√	√	96.1	91.3	300

The training process for the vehicle detection model requires heavy computation and therefore we utilized transfer learning techniques and made use of pre-trained weights which have originally been trained on the MS COCO dataset as the starting point for our vehicle detection training process. This helped in significantly increasing the speed of our model convergence and achieved a high accuracy score faster as compared to when starting the training by utilizing randomly initialized weights. Furthermore, we also utilized an Nvidia GeForce RTX 3060 laptop GPU for the training of our model which helped in speeding up the learning operations by utilizing multiple cores and a large memory bandwidth for high-speed parallel processing. In total, the

time taken for training of our vehicle detection model was 13 hours and 27 minutes and the results obtained on the validation set for the Precision, Recall and average mAP@.5 score for each of the vehicle classes are shown in the Table 4.2.

Table 4.2 Model Training Results on Validation set

Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95:
all	2350	7937	0.918	0.913	0.961	0.709
bus	2350	337	0.923	0.884	0.954	0.746
car	2350	4921	0.918	0.908	0.963	0.704
motorcycle	2350	55	0.889	0.945	0.953	0.652
truck	2350	2624	0.944	0.913	0.974	0.735

Finally, after the successful training of our vehicle detection model, the loss function plots were saved and can be seen in the Figure 4.1 displaying three main types of losses which are the box, objectness and classification losses. The box regression loss represents how well the algorithm was able to locate the center of the vehicles and how well the predicted bounding boxes covered the detected vehicles. Objectness loss refers to the measure of the probability that a vehicle exists in a predicted region of interest and finally the classification loss refers to the ability of the algorithm to be able to predict the correct vehicle class given a detected vehicle object.

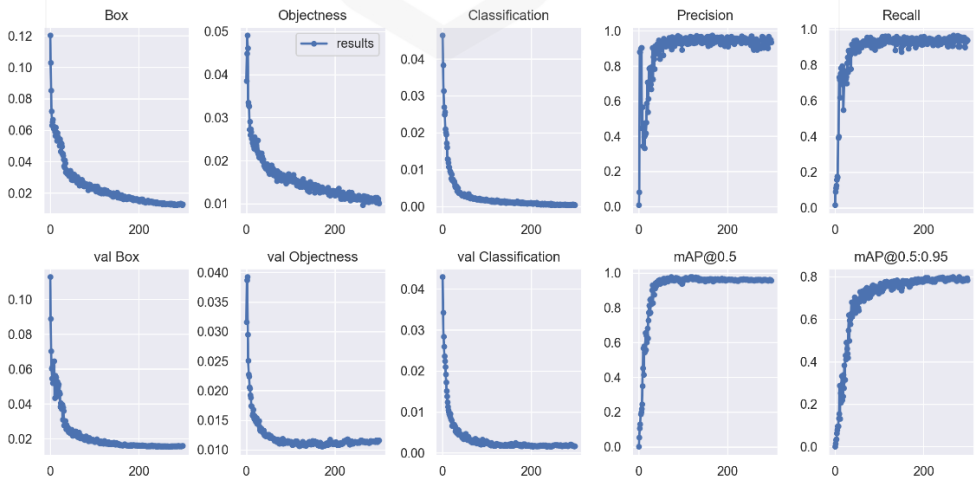


Figure 4.1 Model Training Plots over Epochs Results on the Validation set

The Precision and Recall plots were also shown in the Figure 4.1 and represent fundamental metrics which are used to identify whether the vehicle detection model was able to make good classifications on detected vehicles. The Precision equation was shown earlier in the equation (4.1) and makes use of the True Positive (TP) as well as False Positive (FP) detections to determine what proportion of positive vehicle identifications were actually identified correctly. As for the Recall, the equation was shown earlier in the equation (4.2) and makes use of the True Positive (TP) as well as False Negative (FN) detections to determine what proportion of actual positives were identified correctly. A high Precision and Recall means that the model is capable of making accurate classification of positive vehicles and that it is able to classify all positive vehicles. The Precision-Recall curve for the vehicle detection model is shown in the Figure 4.2 and is obtained by plotting the model's precision over recall values as a function of the model's confidence score threshold.

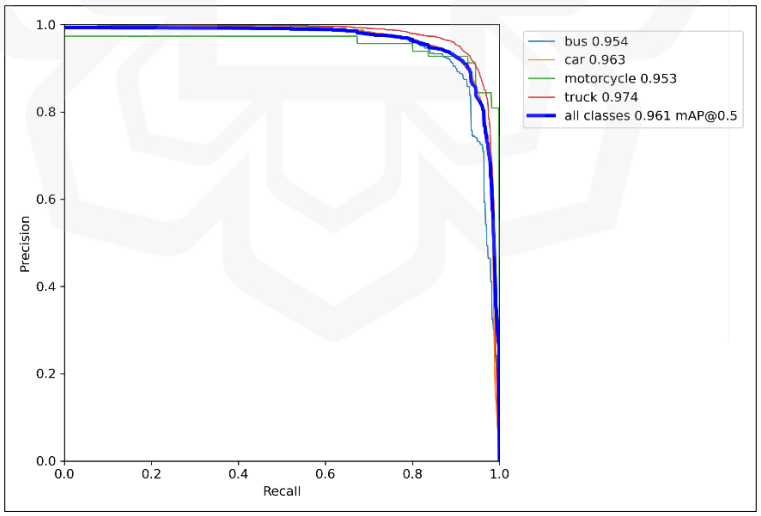


Figure 4.2 Precision-Recall Curve

Evaluating the model performance on the validation set is sufficient enough to be able to obtain accurate model performance, however, at times the model may still be prone to overfitting. Therefore, it is always recommended to have a separate set of

images which has never been seen before by the model during the training phase, known as the test set. The test set will be used for further validation of the model's performance and to ensure that it is free from any overfitting. In our case, 1,184 images were randomly extracted from the original dataset containing cars, motorcycles, buses and trucks and were kept for further testing of the model's performance after the training had completed. After evaluation of the trained model on the test set, the model was able to achieve an average mAP@.5 score of 95.89% which validated that the model is free from any overfitting. The Figure 4.3 shows some of the predictions made by the trained model on the new and unseen test images along with the model's confidence scores.

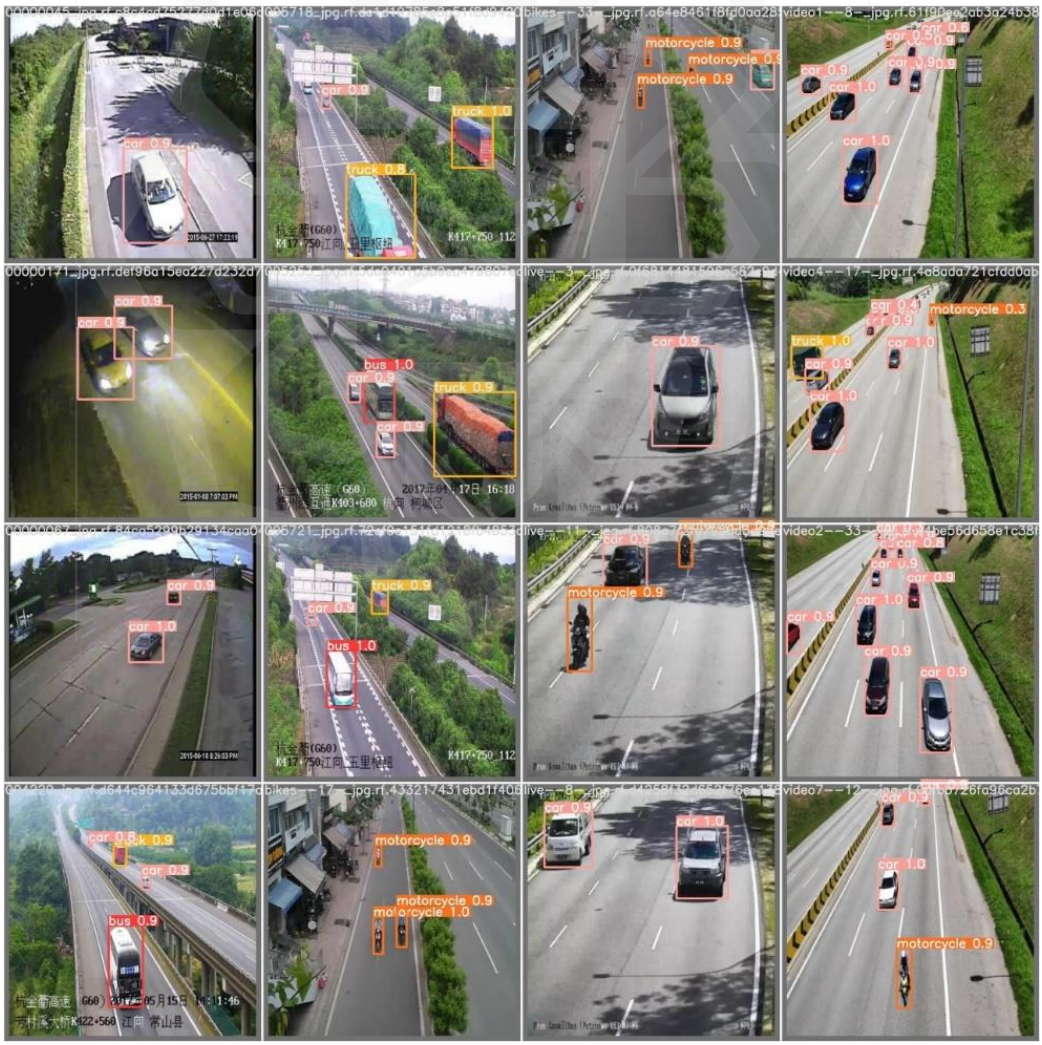


Figure 4.3 Vehicle Detection Results on Test Images

4.3 EXPERIMENTAL RESULTS AND ANALYSIS

An edge-computing system platform based on the Nvidia Jetson Nano was integrated and used for the testing of our vehicle counting system in real world test scenarios. The experimental setup was deployed at the Genting Sempah Highway located at Kuala Lumpur, Malaysia and in an afternoon when the weather condition was relatively bright and therefore results obtained may vary depending on the weather conditions. The edge-computing system platform was set up at a pedestrian bridge located approximately 5 meters above the ground and a Logitech C922 Pro HD webcam was used for obtaining the live video stream. The Nvidia Jetson Nano was used to run the inference and perform the vehicle detection, tracking and counting. Finally, the vehicle detection results along with the vehicle counts for each class of vehicle were displayed on a 7-inch LCD display in real-time showing a visualization similar to that observed in the Figure 4.4.

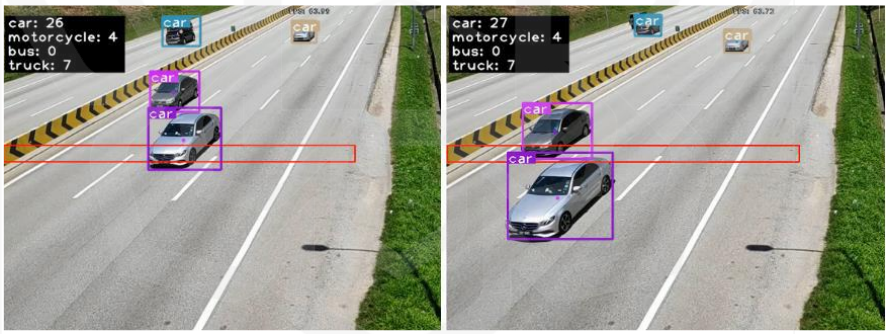


Figure 4.4 Sample Vehicle Counting Results

The experimental setup was tested by using varying angles, each capturing roughly 2 minutes of video data. A few of the live camera inferences obtained from the experimental setup can be viewed at the following link: https://tinyurl.com/yppcua3d. As observed from the video results, the frames per second for running of the inference was also calculated and displayed on the screen. The average speed obtained for running the inference on the 4GB Nvidia Jetson Nano was 15 FPS which is close to real-time.

Besides the live video inferences obtained from the deployment of the edge-computing system platform, several highway surveillance videos obtained from YouTube were also utilized as input for testing of our vehicle counting system. The videos consisted of three one-sided highway videos which comprised of various lengths varying between 0:12 and 01:35 durations which are freely available for researchers on YouTube. The Figure 4.5 shows snapshots of the final vehicle counting results obtained on the three separate highway scenes.

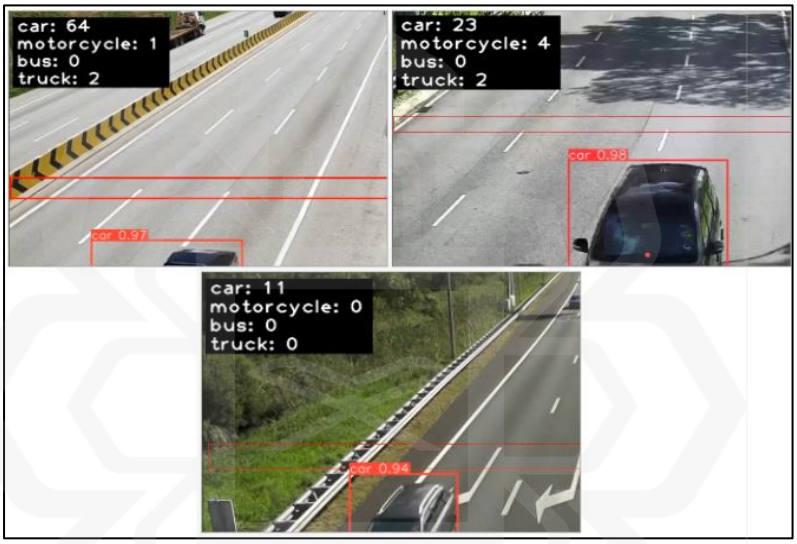


Figure 4.5 Snapshots of Vehicle Counting Results on Different Highway Scenes

Furthermore, the ground truth counts for each of the four trained vehicle classes were manually counted in order to obtain the accuracy of our vehicle counting method. The Table 4.3 summarizes the vehicle counting results obtained on a tabular format, displaying the duration of the open-source videos used for experiments, the ground truth counts of the vehicles observed for each of the four-vehicle classes and finally the counts made by our proposed vehicle counting method. The table further presents the average vehicle counting accuracy score, which was achieved by analyzing the video data encompassing all four trained vehicle classes, which obtained a score of 95.39%.

Table 4.3 Vehicle Counting Performance of the Proposed System

Video Name on YouTube	Duration (minutes)	Vehicle	Counting		Counting Error	Overall Accuracy
			Real	System		
Genting Highway Traffic Cars, Motorcycle, Trucks	01:35	Car Motorcycle Bus Truck	66 1 0 5	64 1 0 2	+2 0 0 +3	93.06%
Live Motorway Traffic in Aceh Indonesia	00:59	Car Motorcycle Bus Truck	24 4 0 1	23 4 0 2	+1 0 0 +1	93.10%
Sample Video Cars Traffic – 27260	00:12	Car Motorcycle Bus Truck	11 0 0 0	11 0 0 0	0 0 0 0	100%
Average Counting Accuracy						95.39%

Comparisons with other methods was difficult because most of the systems developed for the task of vehicle counting are able to obtain the total vehicle counts but are unable to obtain the individual vehicle counts for multiple individual classes of vehicles. The closest method was that developed by (Fachrie, 2020) which makes use of the YOLOv3 network and a virtual line counter with distance measurement for the counting of cars, motorcycles, buses and trucks similar to our system and which had achieved an average vehicle counting accuracy score of 92.20%. Unfortunately, the data used for testing of the following system has not been open-sourced and therefore accurate comparisons with our proposed system was not possible. Moreover, referring to the Table 4.3, our proposed system was not only able to obtain very accurate vehicle counting results, but it also consists of a lightweight vehicle detection model and a fast ReID model, enabling fast-processing time and deployment on edge-computing devices.

There were several challenges encountered when running the inference on the experimental setup and testing the performance of the system in real-world scenarios. The first problem was related to the vehicle detection, the algorithm struggled in detecting vehicles that were very close to each other or that were obstructed from the cameras Field of View (FoV). To solve the vehicle detection problems, identifying a suitable camera angle which had minimal to no blind spots was required. The second problem was related to the vehicle tracking where ID switches were very common, especially when the vehicles had similar model types and were travelling side by side on the road, making it difficult for the tracker to be able to differentiate between the distinct vehicles.

To solve the vehicle ID assignment problem, a unique vehicle counting method was implemented which does not rely solely on the vehicle tracking ID assignment for the counting and which resulted in accurate and efficient vehicle counting results. In summary, the comparison of our vehicle detection accuracy score was compared with other vehicle detection algorithms on the same validation set and the results are displayed on the Table 4.4.

Table 4.4 Vehicle Detection Accuracy Comparison

Algorithm Used	**Accuracy**	**Speed (FPS)**	**Reference**
Faster R-CNN	89.05%	0.5	(Ren et al., 2018)
YOLOv3 + ORB	87.88%	8.47	(Song H. et al., 2019)
YOLOv4	82.08%	11.2	(Bochkovskiy et al., 2020)
Original YOLOv5n	93.2%	16.41	(Jocher et al., 2020)
Proposed Algorithm	**96.1%**	**16.4**	Our proposed Approach

4.4 DISCUSSION

In this section, we provide an extensive analysis of the results obtained from the custom vehicle detection algorithm, tracking algorithm and counting method used in this study. The previous section provided a detailed account of the outcomes and performance metrics, while this section aims to provide a deeper interpretation and evaluation of these results. Additionally, we will also go through the outcomes for each of the three main subtopics in this section, highlighting the challenges we had encountered and the successes we had achieved. Through this discussion, we aim to provide a comprehensive understanding of the effectiveness and applicability of our proposed methods and shed light on the implications of the system in real world traffic analysis.

4.4.1 Vehicle Detection

Vehicle detection is the most crucial part of the vehicle counting system. In order to train our custom vehicle detector, 10,587 images were employed. Out of these, 8,237 images were used for the training of the model, meanwhile the remaining 2,350 were reserved for validation of the model's performance. Upon analyzing the loss function plots depicted in the Figure 4.1, we observe that the loss plots converged significantly fast when using our custom deep neural network architecture. By approximately 200 epochs, the model had already converged with a loss approximately 9.37% lower than that observed when using the default YOLOv5n model architecture. These results provide compelling evidence of the effectiveness of our proposed vehicle detection algorithm and its superiority over the default YOLOv5n model architecture.

Moreover, the mean average precision, mAP@.5 serves as a reliable indicator of the vehicle detection model's performance. A higher mAP@.5 score signifies greater

precision and accuracy in detecting vehicles. Figure 4.1 provides valuable insights into this aspect, revealing that the mAP@.5 score gradually stabilizes after 100 epochs of training, reaching approximately 96.02%. As the training progresses, the model achieves its highest mAP@.5 score of 96.1% at around 300 epochs. To enhance the model's performance, data augmentation techniques were applied to the training images as presented in the Table 4.1. This resulted in an improved recall and the model's ability to identify a majority of the vehicles correctly. However, the augmentation did also lead to a reduction in precision, consequently increasing the number of false positive detections.

The accuracy of our vehicle detection model was evaluated by comparing it with three existing models, serving as baseline benchmarks. Firstly, the original YOLOv5n model architecture, which achieved an average mAP@.5 score of 93.2% on the same validation set. Additionally, we assessed the performance of our model against another vehicle detection approach proposed by (Song & Liang, 2019). This approach utilized the YOLOv3 network in conjunction with an ORB algorithm for the detection of vehicles in highway scenes, yielding an average mAP@.5 score of 87.88%. Furthermore, we compared our model to the method introduced by (Bin Zuraimi & Kamaru Zaman, 2021), which employed the YOLOv4 network for vehicle detection, resulting in an average mAP@.5 score of 82.08%. Notably, our proposed architecture demonstrated superior performance in terms of mean average precision, surpassing all three of these comparative models.

4.4.2 Vehicle Tracking

Vehicle tracking plays a crucial role in our vehicle counting system. In our study, we employed the DeepSORT algorithm which combines the Kalman Filter for precise

estimation of vehicle locations and the Hungarian algorithm for frame-by-frame analysis of the video sequences. Additionally, we also incorporated the OSNet ReID model to extract feature vectors that enhanced the association of vehicles in highway scenes. To benchmark the performance of recent Re-Identification (ReID) architectures for Tracking by Detection (DBT) in Multi-Object Tracking (MOT) scenarios, we referred to the analysis conducted by (Ishikawa et al., 2021). Their research focused on heavily occluded scenes using the MOT20 Challenge dataset and multiple metrics, including MOTO, were employed to evaluate the performance of different models. Remarkably, the DeepSORT tracker combined with the OSNet ReID model demonstrated strong performance, achieving the second highest MOTO value among the evaluated models.

To maintain accurate vehicle tracking and ID assignment throughout the video sequence, the Hungarian algorithm was applied to associate and attribute a unique identification to each vehicle. Furthermore, because the DeepSORT incorporates a convolutional neural network (CNN) similar to that observed in our vehicle detection algorithm, the vehicle tracker exhibited robustness against object misses and occlusion, enabling seamless tracking in online and real-time scenarios. Moreover, the study adopted a Re-Identification (ReID) model, specifically the Omni-Scale Network (OSNet), originally trained on the Market 1501 dataset and had achieved an average mAP score of 84.9%. The OSNet ReID model was leveraged to extract feature vectors, allowing for improved association of detected vehicles on the highway scenes.

The combination of DeepSORT and OSNet ReID models have proven to be advantageous due to their lightweight nature, enabling fast inference times and deployment on edge-computing devices, which is crucial in our application. However,

occasionally, even while using the DeepSORT and OSNet tracking algorithms, ID switches still occurred from time to time during the tracking process, which was attributed to factors including occlusion, dynamic background, illumination variation and video noise. In order to solve this, the model was further trained on our custom vehicle dataset which significantly reduced the occurrence of ID switches and further enhanced the vehicle tracking performance.

4.4.3 Vehicle Counting

Once vehicle detection and tracking has taken place, vehicle counting serves as the concluding stage of the vehicle counting system. The vehicle counting method employed in this study utilizes a virtual polygon area for accurate counting of vehicles, achieving an average counting accuracy score of 95.39%. A comparative analysis was conducted with other counting methods, such as the YOLOv3 network combined with a virtual line counter in (Fachrie, 2020), which achieved an average counting accuracy score of 92.20% on cars, motorcycles, buses and trucks, similar to our system. Another method proposed by (Song & Liang, 2019) which made use of a virtual line counter for the counting of cars, busses and trucks had achieved an average vehicle counting accuracy of 93.20%.

Our innovative vehicle counting method stands out for its ability to accurately count vehicles without duplicate counts or missed objects. Furthermore, the system is also capable of counting vehicles moving in both inbound and outbound directions simultaneously, allowing for comprehensive traffic analysis. Moreover, the combination of a lightweight vehicle detection and tracking model, along with an effective counting technique using virtual polygon areas, enabled our system to achieve the highest accuracy score among the compared methods. This remarkable accuracy,

achieved in real-time scenarios, further underscores the effectiveness and reliability of our proposed counting approach, emphasizing its practical value and potential for widespread implementation.

Furthermore, it is also crucial to highlight that our proposed vehicle counting system not only delivers highly accurate counting results but also comprises of a lightweight vehicle detection model and a fast ReID model. The system runs at an impressive speed of 15 FPS on a low-computing Nvidia Jetson Nano edge device and an even faster 67 FPS on an RTX 3060 laptop GPU, ensuring its suitability for offline edge computing. This efficient performance makes our system well-suited for real-world applications where real-time counting and processing are essential.

4.5 SUMMARY

This chapter presented the results and evaluation of our proposed vehicle counting system. It highlighted on the main evaluation metrics that were used to determine the accuracy score of the vehicle detection model and analyzed its performance on various scenarios in order to determine its efficiency as well as ability to generalize to new environments. Furthermore, the chapter also discusses on the live vehicle counting experiments that have been conducted in real-world scenarios by utilizing the edge-computing system platform, the results that have been achieved and the challenges that were faced. Finally, the efficiency as well as accuracy of the vehicle detection and counting methods were benchmarked with other similar methods and a discussion was presented to elaborate on the findings. From the results observed and the experimental setup conducted, it is clear that our proposed real-time vehicle counting system was able to achieve very high vehicle detection and counting accuracy scores in highway scenes (above 95%) and that it is ready to be implemented in real world practical applications.

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.1 CONCLUSION

In conclusion, we were successful in generating a new highway vehicle dataset consisting of 11,982 images of the following four vehicle classes respectively; car, motorcycle, bus and truck each captured from a highway CCTV point-of-view. The vehicle dataset was cleaned, manually annotated and has now been published and made available for other researchers working on the same area.

Furthermore, a custom vehicle detection algorithm based on the YOLOv5n model architecture was successfully developed and used for the detection of vehicles in the highway-based scenes. Based on the qualitative analysis, our trained vehicle detection algorithm was able to obtain very high vehicle detection accuracy results (above 95%) and was able to successfully detect the small-scale vehicles observed in the highway scenes. Moreover, the generated vehicle detection model was also very light in size and was able to run in real-time on a compact edge computing device.

The trained vehicle detection model was successfully integrated with a DeepSORT vehicle tracking algorithm and a light-weight OSNet ReID model for the tracking of vehicles across the highway scenes. The vehicle tracking was performed with minimal ID switches and due to the use of light-weight Kalman Filters for the calculation, it was also able to run with high inference speeds and with high accuracy.

Finally, a unique “virtual polygon area” counting approach was also introduced and implemented which allowed for accurate and efficient vehicle counting results and

avoided duplicate counts. The entire system was deployed on an Nvidia Jetson Nano edge-computing platform and based on the qualitative results achieved, our proposed system was found to have achieved very high vehicle detection, tracking and counting results in the highway-based scenes. Furthermore, the proposed system was also light in weight and was deployed in real-world scenarios achieving high accuracy as well as inference speeds.

5.2 RECOMMENDATION

The proposed vehicle counting system makes use of the current state-of-the-art vehicle detection and tracking algorithms and had achieved significant improvements in the detection as well as counting accuracy of vehicles in highway scenes. However, there were some shortcomings that were experienced in which we aim to overcome in the future study. Occlusion and low light conditions created identity switches and difficulty for the tracker to accurately differentiate between distinct moving vehicles, in the future, a new vehicle ReID model will be trained by using type-specific vehicle sequence data for accurate tracking of vehicles, furthermore, improvements will be made to estimate vehicles speed, direction and trajectory for simultaneous counting in two-way traffics.

REFERENCES

Abdullah, A., & Oothariasamy, J. (2020). Vehicle counting using deep learning models: A comparative study. *International Journal of Advanced Computer Science and Applications*, *11*(7), 697–703.

Alpatov, B. A., Babayan, P. V., & Ershov, M. D. (2018). Vehicle detection and counting system for real-time traffic surveillance. *2018 7th Mediterranean Conference on Embedded Computing, MECO 2018 - Including ECYPS 2018, Proceedings*, *June*, 1–4.

Anka, A. (2020). *YOLO v4: Optimal Speed & Accuracy for object detection | by Andrej Anka | Towards Data Science*. 1–19. https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50

Barba-Guaman, L., Naranjo, J. E., & Ortiz, A. (2020). Deep learning framework for vehicle and pedestrian detection in rural roads on an embedded GPU. *Electronics (Switzerland)*, *9*(4), 1–17.

Barla, N. (2022). *The Complete Guide to Object Tracking [+V7 Tutorial]*. https://www.v7labs.com/blog/object-tracking-guide#object-tracking-stages

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *Proceedings - International Conference on Image Processing, ICIP*, *2016-Augus*, 3464–3468.

Bin Zuraimi, M. A., & Kamaru Zaman, F. H. (2021). Vehicle detection and tracking using YOLO and DeepSORT. *ISCAIE 2021 - IEEE 11th Symposium on Computer Applications and Industrial Electronics*, 23–29.

Bochkovskiy, A., Wang, C.-Y., & Liao, H.-Y. M. (2020). *YOLOv4: Optimal Speed and Accuracy of Object Detection*. http://arxiv.org/abs/2004.10934

Brownlee, J. (2019). A Gentle Introduction to Object Recognition With Deep Learning. *Machine Learning Mastery*, 1–31. https://machinelearningmastery.com/object-recognition-with-deep-learning/

Chaudhari, S., Malkan, N., Momin, A., & Bonde, M. (2020). Yolo Real Time Object Detection. *International Journal of Computer Trends and Technology*, *68*(6), 70–76.

Chen, Z., Ellis, T., & Velastin, S. A. (2012). Vehicle detection, tracking and classification in urban traffic. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 951–956.

D'costa, A. A. (2020). *Model Optimization For Edge Devices*. https://repository.lib.fit.edu/handle/11141/3134

Dendorfer, P., Rezatofighi, H., Milan, A., Shi, J., Cremers, D., Reid, I., Roth, S., Schindler, K., & Leal-Taixé, L. (2020). *MOT20: A benchmark for multi object tracking in crowded scenes*. 1–7. http://arxiv.org/abs/2003.09003

Dinh, D.-L., Nguyen, H.-N., Thai, T., & Le, K.-H. (2021). Towards AI-Based Traffic Counting System with Edge Computing. *Journal of Advanced Transportation*, *2021*, 1–15.

Fachrie, M. (2020). A Simple Vehicle Counting System Using Deep Learning with YOLOv3 Model. *Jurnal RESTI (Rekayasa Sistem Dan Teknologi Informasi)*, *4*(3), 462–468.

Gao, X., Shen, Z., & Yang, Y. (2022). Multi-object tracking with Siamese-RPN and adaptive matching strategy. *Signal, Image and Video Processing*, 1–9.

Girshick, R. (2015). Fast R-CNN. *Proceedings of the IEEE International Conference on Computer Vision*, *2015 Inter*, 1440–1448.

Girshick, R., Donahue, J., Darrell, T., & Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 580–587.

Gunjal, P. R., Gunjal, B. R., Shinde, H. A., Vanam, S. M., & Aher, S. S. (2018). Moving Object Tracking Using Kalman Filter. *2018 International Conference On Advances in Communication and Computing Technology, ICACCT 2018*, 544–547.

Hui, J. (2020). YOLOv4 (explained). *Medium*, 1–20. https://jonathan-hui.medium.com/yolov4-c9901eaa8e61

Industries, I. H., Computing, E., & Computing, E. (2019). *Edge computing changes everything*. 1–6.

Ishikawa, H., Hayashi, M., Phan, T. H., Yamamoto, K., Masuda, M., & Aoki, Y. (2021). Analysis of recent re-identification architectures for tracking-by-detection paradigm in multi-object tracking. *VISIGRAPP 2021 - Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications*, *5*(Visigrapp), 234–244.

Iyer, L. S. (2021). AI enabled applications towards intelligent transportation. *Transportation Engineering*, *5*, 100083.

Jakubiak, A., & Radomski, D. (2013). Expert systems in medical diagnosti. *Kwartalnik Elektroniki i Telekomunikacji*, *49*(4), 481–502.

Jocher, G., Chaurasia, A., Stoken, A., Borovec, J., Kwon, Y., Fang, J., Michael, K., Abhiram, V., Montes, D., Nadar, J., Skalski, P., Wang, Z., Hogan, A., Fati, C., Mammana, L., Patel, D., Yiwei, D., & You, F. (2020). *ultralytics / yolov5 : v6 . 1 - TensorRT , TensorFlow Edge TPU and OpenVINO Export and Inference*. 4–11.

Kalman, R. E. (1960). A New Approach to Linear Filtering and Prediction Problems. *SAE Technical Papers*.

Kim, Y., & Bang, H. (2019). Introduction to Kalman Filter and Its Applications. *Introduction and Implementations of the Kalman Filter*.

Lee, B. Y., Liew, L. H., Cheah, W. S., & Wang, Y. C. (2014). Occlusion handling in videos object tracking: A survey. *IOP Conference Series: Earth and Environmental Science*, *18*(1).

Li, H., & Zahr, M. (2012). Learning to Recognize Objects in Images. *Trends in Cognitive Sciences*, *3*(3), 1–5.

Li, Q., Li, R., Ji, K., & Dai, W. (2016). Kalman filter and its application. *Proceedings - 8th International Conference on Intelligent Networks and Intelligent Systems, ICINIS 2015*, *10*, 74–77.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016). SSD: Single shot multibox detector. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9905 LNCS*, 21–37.

Lohia, A. (2021). *DigitalCommons @ University of Nebraska - Lincoln Bibliometric Analysis of One-stage and Two-stage Object Detection Bibliometric Analysis of One-stage and Two-stage Object Detection*. *February*.

Luo, W., Xing, J., Milan, A., Zhang, X., Liu, W., & Kim, T. K. (2021). Multiple object tracking: A literature review. *Artificial Intelligence*, *293*, 1–49.

Luo, Z., Charron, F. B., Lemaire, C., Konrad, J., Li, S., Mishra, A., Eichel, J., & Jodoin, P. (2018). *Details of the MIO-TCD dataset dataset for vehicle MIO-TCD : A new benchmark classification and localization in press at IEEE Transactions on Image Classification challenge dataset Localization challenge dataset Click here to download the entire dataset :* 1–9.

Lyu, S., Li, R., Zhao, Y., Li, Z., Fan, R., & Liu, S. (2022). Green Citrus Detection and Counting in Orchards Based on YOLOv5-CS and AI Edge System. *Sensors*, *22*(2), 1–20.

MacHiraju, G. S. R., Kumari, K. A., & Sharif, S. K. (2021). Object Detection and Tracking for Community Surveillance using Transfer Learning. *Proceedings of the 6th International Conference on Inventive Computation Technologies, ICICT 2021*, 1035–1042.

Maiya, S. (2019). DeepSORT: Deep Learning to track custom objects in a video. *Nanonets*, 1–24. https://nanonets.com/blog/object-tracking-deepsort/

Mandal, V., & Adu-Gyamfi, Y. (2019). Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis. *2019 12th International Conference on Contemporary Computing, IC3 2019*, 1–6.

Matsuo, T., Kaneko, Y., & Matano, M. (1999). Introduction of intelligent vehicle detection sensors. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 709–713.

Meel, V. (2021). YOLOv5 Is Here! Is It Real or a Fake? *Viso.Ai*, *3*, 1–10. https://viso.ai/deep-learning/yolov5-controversy/

Monica, M. V., & Nigel, K. G. J. (2017). Object tracking based on Kalman filter and gait feature extraction. *Proceedings of the International Conference on Inventive Systems and Control, ICISC 2017*, 1–5.

NVidia Corporation. (2019). *Jetson NANO Module*. https://developer.nvidia.com/embedded/jetson-nano

Paik, C., & Kim, H. J. (2022). *Improving Object Detection, Multi-object Tracking, and Re-Identification for Disaster Response Drones*. http://arxiv.org/abs/2201.01494

Rakai, L., Song, H., Sun, S., Zhang, W., & Yang, Y. (2022). Data association in multiple object tracking: A survey of recent techniques. *Expert Systems with Applications*, *192*, 116300.

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 779–788.

Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, *2017-Janua*, 6517–6525.

Redmon, J., & Farhadi, A. (2018). YOLO v.3. *Tech Report*, 1–6. https://pjreddie.com/media/files/papers/YOLOv3.pdf

Ren, S., He, K., Girshick, R., & Sun, J. (2017). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(6), 1137–1149.

Rhodes, A. D. (2018). *A Overview of Computer Vision Tasks , including Multiple-Object Detection (MOT)*.

Roehrig, C., & Müller, M. (2009). *Indoor location tracking in non-line-of-sight environments using a IEEE 802.15.4a wireless network*. 552–557.

Safiullin, R., Kerimov, M., Afanasyev, A., & Marusin, A. (2018). A model for justification of the number of traffic enforcement facilities in the region. *Transportation Research Procedia*, *36*(SPbOTSIC), 493–499.

Sakagami, Y., Watanabe, R., Aoyama, C., Matsunaga, S., Higaki, N., & Fujimura, K. (2002). The intelligent ASIMO: System overview and integration. *IEEE International Conference on Intelligent Robots and Systems*, *3*(October), 2478–2483.

Samadi, S., Rad, A., mohammad kazemi, F., & Jafarian, H. (2012). Performance Evaluation of Intelligent Adaptive Traffic Control Systems: A Case Study. *Journal of Transportation Technologies*, *02*, 248–259.

Santos, A. M., Bastos-Filho, C. J. A., MacIel, A. M. A., & Lima, E. (2020). Counting Vehicle with High-Precision in Brazilian Roads Using YOLOv3 and Deep SORT. *Proceedings - 2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images, SIBGRAPI 2020*, 69–76.

Shehu, D. (2010). *Using Real Time Computer Vision Algorithms in Automatic Attendanc Management Systems*. 2–9.

Skalski, P. (2019). *Make Sense*.

Song, H., & Liang, H. (2019). *Vision-based vehicle detection and counting system using deep learning in highway scenes*. *4*.

Sreenu, G., & Saleem Durai, M. A. (2019). Intelligent video surveillance: a review through deep learning techniques for crowd analysis. *Journal of Big Data*, *6*(1), 1–27.

Stauffer, C., & Grimson, W. E. L. (2000). Learning patterns of activity using real-time tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *22*(8), 747–757.

Thuan, D. (2021). *Evolution of Yolo Algorithm and Yolov5: the State-of-the-Art Object Detection Algorithm*. 61.

Tian, B., Yao, Q., Gu, Y., Wang, K., & Li, Y. (2011). Video processing techniques for traffic flow monitoring: A survey. *IEEE Conference on Intelligent Transportation Systems, Proceedings, ITSC*, 1103–1108.

Toropov, E., Gui, L., Zhang, S., Kottur, S., & Moura, J. M. F. (2015). Traffic flow from a low frame rate city camera. *Proceedings - International Conference on Image Processing, ICIP*, *2015-Decem*, 3802–3806.

Tsai, C., & Yeh, Z. (2013). *via Adaptive Frame Differencing Method*. *1*, 1–11.

Valencia, I. J., Dadios, E., Fillone, A., Puno, J. C., Baldovino, R., & Billones, R. K. (2021). *Vision-based Crowd Counting and Social Distancing Monitoring using Tiny-YOLOv4 and DeepSORT*. 1–7.

Verani, E., & Pitsiava-Latinopoulou, M. (2019). *Traffic management integrated in the urban development of an area towards sustainability*.

Vision, I. C. (2019). *Computer Vision for Multi-Object Tracking — Live Example*. 1–17.

Weijie, X., Feihong, Y., & Shuaiqi, L. (2022). Real-Time Multi-Class Disturbance Detection for OTDR Based on YOLO Algorithm. *Sensors*.

Whang, S. E., & Lee, J.-G. (2020). Data collection and quality challenges for deep learning. *Proceedings of the VLDB Endowment*, *13*(12), 3429–3432.

Wojke, N., Bewley, A., & Paulus, D. (2018). Simple online and realtime tracking with a deep association metric. *Proceedings - International Conference on Image Processing, ICIP*, *2017-Septe*, 3645–3649.

Xiang, X., Zhai, M., Lv, N., & El Saddik, A. (2018). Vehicle counting based on vehicle detection and tracking from aerial videos. *Sensors (Switzerland)*, *18*(8), 1–17.

Zahaby, M., Gaonjur, P., & Farajian, S. (2009). *Location tracking in GPS using Kalman Filter through SMS*. 1707–1711.

Zhang, Q., Sun, H., Wu, X., & Zhong, H. (2019). Edge video analytics for public safety: A review. *Proceedings of the IEEE*, *107*(8), 1675–1696.

APPENDIX I

STEPS TO INSTALL

The following scripts were tested on an Nvidia Jetson Nano with **Jetpack 4.5** image flashed. To download and flash the Jetpack 4.5, refer to the following link.

1. **SWAP the Jetson Nano to Free Space**

- Run the following commands one by one on the Jetson Nano to increase the swap file size by 4GB.

```
$ sudo systemctl disable nvzramconfig
$ sudo fallocate -l 4G /mnt/4GB.swap
$ sudo chmod 600 /mnt/4GB.swap
$ sudo mkswap /mnt/4GB.swap
$ sudo su
$ echo "/mnt/4GB.swap swap swap defaults 0 0" >> /etc/fstab
$ exit

REBOOT!
```

2. **Install Torch and Torchvision libraries**

- Create a shell script with the following commands and name it “install_torch.sh” then run the following command: ./install_torch.sh

```
#!/bin/bash
# install pytorch 1.8
wget https://nvidia.box.com/shared/static/p57jwntv436lfrd78inwl7iml6p13fzh.whl -O torch-1.8.0-cp36-cp36m-linux_aarch64.whl
sudo apt-get install python3-pip libopenblas-base libopenmpi-dev
pip3 install Cython
pip3 install numpy torch-1.8.0-cp36-cp36m-linux_aarch64.whl

# install torch vision 0.9.0
sudo apt-get install libjpeg-dev zlib1g-dev libpython3-dev libavcodec-dev libavformat-dev libswscale-dev
git clone --branch release/0.9 https://github.com/pytorch/vision torchvision
cd torchvision
export BUILD_VERSION=0.9.0
python3 setup.py install --user
cd ../
pip3 install 'pillow<7'
```

3. Install OpenCV on Jetson Nano

- Run the following commands one by one on the Jetson Nano in order to install OpenCV 4.5.5.

```
$ wget https://github.com/Qengineering/Install-OpenCV-Jetson-Nano/raw/main/OpenCV-4-5-5.sh
$ sudo chmod 755 ./OpenCV-4-5-5.sh
$ ./OpenCV-4-5-5.sh
```

4. Install the requirements

- Create a text file with the following commands and name it “requirements.txt”, then run the following command: pip3 install -r requirements.txt

```
# base ----------------------------------------
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.2
Pillow>=7.1.2
PyYAML==5.4.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.41.0

# plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# deep_sort -----------------------------------
easydict

# torchreid
Cython
h5py
six
tb-nightly
future
yacs
gdown
flake8
yapf
isort==4.3.21
imageio
```

APPENDIX II

THE CODES

Google Colab is a product from Google Research that allows you to link to a personal Google Drive account and execute arbitrary Python codes from the browser.

1. Preparing the Dataset for Training

- The Highway Vehicle Dataset that has been collected and used in the following study has been published on Google Drive and has the below folder structure.

```
Vehicles (
|
└──train
|   |
|   └──images
|       |   file111.jpg
|       |   file101.jpg
|       |   ...
|   └──labels
|       |   file111.txt
|       |   file101.txt
|       |   ...
|
└──valid
|   |
|   └──images
|       |   file1111.jpg
|       |   file1011.jpg
|       |   ...
|
|   └──labels
|       |   file1111.txt
|       |   file1011.txt
|       |   ...
|
└──test
|   |
|   └──images
|       |   file1112.jpg
|       |   file1012.jpg
|       |   ...
|
|   └──labels
|       |   file1112.txt
|       |   file1012.txt
|       |   ...
)
```

2. **Create the data.yaml file**

- The YOLOv5 algorithm access the dataset images using a yaml file containing summary information about the dataset. The following is our data.yaml file.

```
train: ../MyDrive/YOLOv5/Vehicles/train/images
val: ../MyDrive/YOLOv5/Vehicles/valid/images

nc: 4
names: ['bus', 'car', 'motorcycle', 'truck']
```

3. **Mount the Google Drive to Colab**

- Once the dataset has been prepared, we need to mount the Google Drive to Colab to be able to read the dataset. This is done by using the following command:

```
[1] from google.colab import drive
    drive.mount('/content/gdrive')

    Mounted at /content/gdrive
```

4. **Clone the following repo and install all of the dependencies**

- Clone the following repository which has been published and open-sourced in this study and install all of the required dependencies.

```
[3] !git clone https://github.com/abuelgasimsaadeldin/yolov5-deepsort-vehicle-counter.git  # clone the repo
    %cd yolov5-deepsort-vehicle-counter
    !pip install -qr yolov5-deepsort-vehicle-counter/requirements.txt  # install dependencies

    import torch
    from yolov5 import utils
    display = utils.notebook_init()  # checks

    YOLOv5 🚀 v6.1-243-g7cef03d Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla T4, 15110MiB)
    Setup complete ✅ (2 CPUs, 12.7 GB RAM, 38.7/78.2 GB disk)
    Setup complete. Using torch 1.11.0+cu113 _CudaDeviceProperties(name='Tesla T4', major=7, minor=5, total_memory=15109MB, multi_processor_count=40)
```

5. Let's have a look at our data.yaml file

- Now let's visualize our data.yaml file to verify that our train and valid dataset locations are all right and the number of classes are also correct.

```
[5]  %cd /content/gdrive/MyDrive/YOLOv5/Vehicles
     %cat data.yaml

     /content/gdrive/MyDrive/YOLOv5/Vehicles
     train: ../gdrive/MyDrive/YOLOv5/Vehicles/train/images
     val: ../gdrive/MyDrive/YOLOv5/Vehicles/valid/images

     nc: 4
     names: ['bus', 'car', 'motorcycle', 'truck']
```

6. Train the model on the custom dataset using pre-trained weights

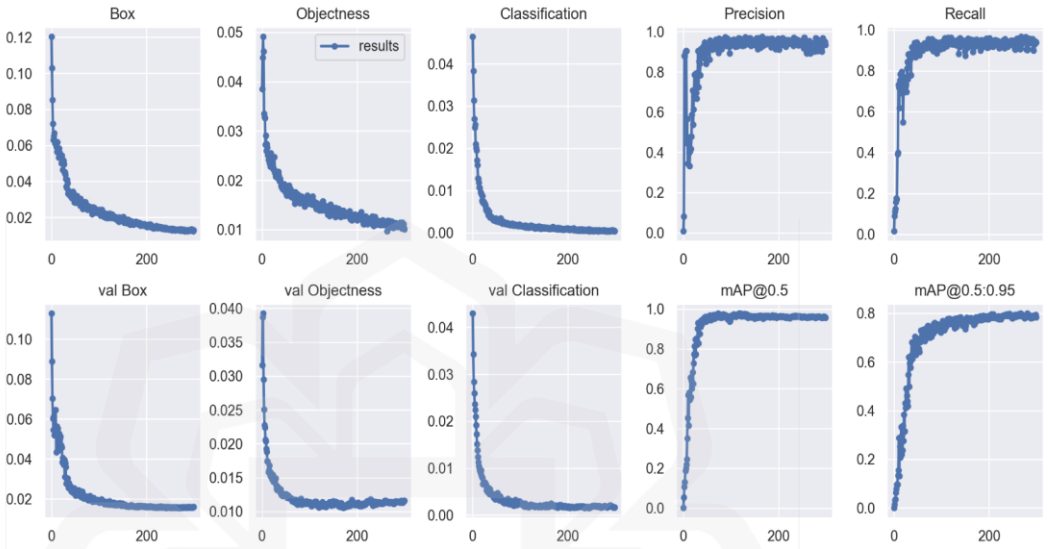
- Train the model on the custom dataset to detect and classify four vehicle classes: car, motorcycle, bus and truck using transfer learning techniques.

```
[6] %%time
    %cd /content/yolov5-deepsort-vehicle-counter
    !python train.py --img 416 --batch 8 --epochs 300 --data '/content/gdrive/MyDrive/YOLOv5/Vehicles/data.yaml' --weights yolov5n.pt --name vehicle_model --workers 1

    [Errno 2] No such file or directory: '/content/yolov5-deepsort-vehicle-counter'
    /content/yolov5
    train: weights=yolov5n.pt, cfg=, data=/content/gdrive/MyDrive/YOLOv5/Vehicles/data.yaml, hyp=data/hyps/hyp.scratch-low.yaml, epochs=300, batch_size=8, imgsz=416, rec
    github: up to date with https://github.com/ultralytics/yolov5 ✅
    YOLOv5 🚀 v6.1-243-g7cef03d Python-3.7.13 torch-1.11.0+cu113 CUDA:0 (Tesla T4, 15110MiB)

    hyperparameters: lr0=0.01, lrf=0.01, momentum=0.937, weight_decay=0.0005, warmup_epochs=3.0, warmup_momentum=0.8, warmup_bias_lr=0.1, box=0.05, cls=0.5, cls_pw=1.0,
    Weights & Biases: run 'pip install wandb' to automatically track and visualize YOLOv5 🚀 runs (RECOMMENDED)
    TensorBoard: Start with 'tensorboard --logdir runs/train', view at http://localhost:6006/
    Overriding model.yaml nc=80 with nc=4

                    from  n    params  module                                  arguments
      0                -1  1      1760  models.common.Conv                      [3, 16, 6, 2, 2]
      1                -1  1      4672  models.common.Conv                      [16, 32, 3, 2]
      2                -1  1      4800  models.common.C3                        [32, 32, 1]
      3                -1  1     18560  models.common.Conv                      [32, 64, 3, 2]
      4                -1  2     29184  models.common.C3                        [64, 64, 2]
      5                -1  1     73984  models.common.Conv                      [64, 128, 3, 2]
      6                -1  3    156928  models.common.C3                        [128, 128, 3]
      7                -1  1    295424  models.common.Conv                      [128, 256, 3, 2]
      8                -1  1    296448  models.common.C3                        [256, 256, 1]
      9                -1  1    164608  models.common.SPPF                      [256, 256, 5]
     10                -1  1     33024  models.common.Conv                      [256, 128, 1, 1]
     11                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
     12          [-1, 6]  1         0  models.common.Concat                    [1]
     13                -1  1     90880  models.common.C3                        [256, 128, 1, False]
     14                -1  1      8320  models.common.Conv                      [128, 64, 1, 1]
     15                -1  1         0  torch.nn.modules.upsampling.Upsample    [None, 2, 'nearest']
     16          [-1, 4]  1         0  models.common.Concat                    [1]
     17                -1  1     22912  models.common.C3                        [128, 64, 1, False]
     18                -1  1     36992  models.common.Conv                      [64, 64, 3, 2]
     19          [-1, 14]  1         0  models.common.Concat                    [1]
     20                -1  1     74496  models.common.C3                        [128, 128, 1, False]
     21                -1  1    147712  models.common.Conv                      [128, 128, 3, 2]
     22          [-1, 10]  1         0  models.common.Concat                    [1]
     23                -1  1    296448  models.common.C3                        [256, 256, 1, False]
     24      [17, 20, 23]  1     12177  models.yolo.Detect                      [4, [[10, 13, 16, 30, 33, 23], [30, 61, 62, 45, 59, 119], [116, 90, 156, 198, 373, 326]],
    Model summary: 270 layers, 1769329 parameters, 1769329 gradients

    Transferred 343/349 items from yolov5n.pt
    AMP: checks passed ✅
    Scaled weight_decay = 0.0005
    optimizer: SGD with parameter groups 57 weight (no decay), 60 weight, 60 bias
```

7. Evaluate the Vehicle Detector performance

- Once the training had completed, the performance metrics are saved to Tensor board and can be visualized as a PNG file and plotted as follows:

```
[7] # we can also output some older school graphs if the tensor board isn't working for whatever reason...
    from IPython.display import Image
    Image(filename='/content/gdrive/MyDrive/Vehicle_Counter/Results/results.png', width=1000)
```

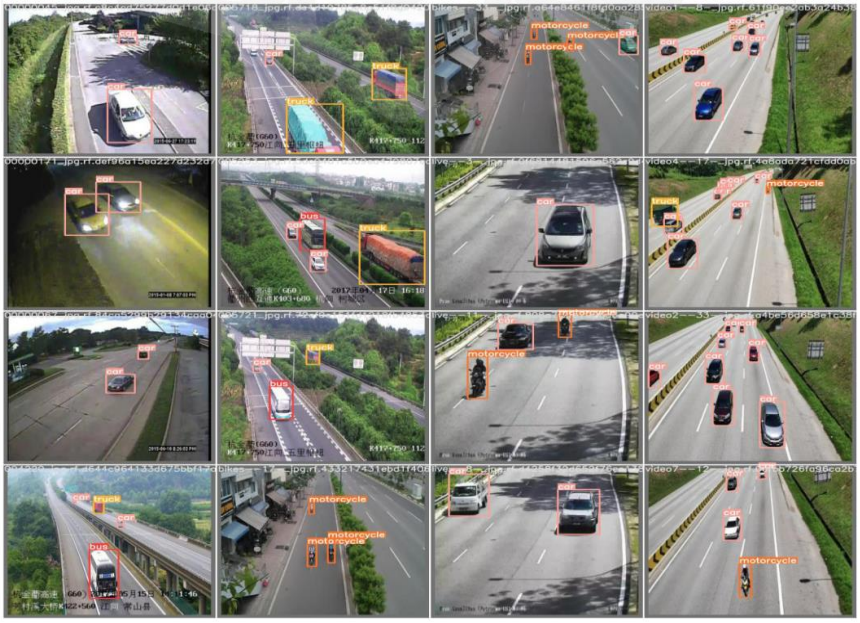


8. Visualize Training Data with Labels

- Now that our model has been successfully trained, out of curiosity let's visualize some of our training images along with their respective labels.

```
[11] print("GROUND TRUTH TRAINING DATA:")
     Image(filename='/content/gdrive/MyDrive/Vehicle_Counter/Results/ground_truth.png', width=600)
```

GROUND TRUTH TRAINING DATA:



9. Visualize Model Results on Validation Set

- Now let's run inference of our trained model on the same set of images to see how well our model was able to perform.

```
[12] print("PREDICTION ON TRAINING DATA:")
     Image(filename='/content/gdrive/MyDrive/Vehicle_Counter/Results/model_prediction.jpg', width=600)
```

```
PREDICTION ON TRAINING DATA:
```

10. Run live vehicle counting inference

- Once all of the requirements have been installed and model weights have been trained, let's test the performance of the vehicle counter on a live camera feed.

```
$ python track.py --source 0 --weights vehicle_model.pt
```