Ollscoil Teicneolaíochta an Atlantaigh

Atlantic Technological University

Computer Vision in Traffic Analysis using EfficientDet and YOLOv11

By
Chayapol Hongsrimuang

September 1, 2025

Masters Thesis

Submitted in partial fulfillment for the award of **Master of Science in Computing** to the Department of Computer Science & Applied Physics, Atlantic Technological University (ATU), Galway.

M.Sc. in Computing

Contents

1 Introduction **1**
- 1.1 Objectives . . . 2
- 1.2 Structure . . . 3

2 Research Methodology **4**
- 2.1 Development . . . 4
- 2.2 Research . . . 5

3 Literature Review **7**
- 3.1 Object Detection Models . . . 7
- 3.2 Mobile and Edge Deployment . . . 8
- 3.3 Research Gaps . . . 9

4 System Design **10**
- 4.1 Principles . . . 10
- 4.2 System Overview . . . 11
- 4.3 Object Detection . . . 12
  - 4.3.1 Model Selections . . . 12
  - 4.3.2 Tracking . . . 15
  - 4.3.3 Optimisations . . . 16
- 4.4 Web Application Development . . . 18
  - 4.4.1 Middleware . . . 18
  - 4.4.2 Database . . . 18
  - 4.4.3 Web Application . . . 19
- 4.5 Deployment . . . 19

5 System Evaluation **21**
- 5.1 Object Detection . . . 21
- 5.2 Web Application and Database . . . 23
- 5.3 Model Comparisons . . . 25

6 Conclusion **27**

7 Appendix **29**

List of Figures

4.1 System Architecture of the project 11
4.2 EfficientDet Architecture [1] 14
4.3 Web application overview 20

5.1 YOLOv11 Detection 22
5.2 EfficientDet-like Model Head Using Transfer Learning 23
5.3 EfficientDet Model Head 23
5.4 EfficientDet Detection 24

List of Tables

4.1 YOLOv11 Performance [2] 13
4.2 EfficientDet Performance [1] 14
4.3 EfficientDet (LiteRT version) mAP Values [3] 15

5.1 YOLOv11 v EfficientDet Performance Metrics 25

Abstract

This thesis discussed the usage of computer vision in traffic analysis with two different object detection models of EfficientDet and YOLOv11, focusing on the rise of using mobile devices for computer vision and machine learning. This thesis aimed to implement these models in different ways, with an Android application for EfficientDet and a Python script for YOLOv11, as well as comparing the two models together. This paper also aimed to produce web application using NextJS, with a stacked bar chart display per time period and details about each detection.

The project was separated into two different parts for object detection and web application. The object detection part included the development of the Android application and Python script, as well as tracking algorithm for the objects themselves. An attempt was also made for a transfer learning process of the EfficientDet. However, this failed due to complications of making a detection head for predictions as well as complexity of the EfficientNet model used in the process. The web application part included the development of the NextJS web application, as well as a middleware, gathering all data from the object detection part and distributing data for the web application. Accompanying the middleware is also a MongoDB database, storing all detections recorded.

The results from the findings concluded that there is a possibility of more fine-tuning of using computer vision in mobile devices, with a web application accompanying the results display. YOLOv11 also performed better than EfficientDet, prompting other approaches that can be used to implement it in mobile devices, such as a video streaming approach to a YOLOv11 prediction model.

Acknowledgments

I would like to express a personal gratitude to the supervisor of the project, Dr Brian McGinley, for his guidance of the project as well as providing feedback towards the completion of the research thesis project. I would also would like to thank the technicians for supporting the project through various means, including providing platforms for development of the project and the creation of this paper, as well as resources aiding this project.

Another special recognition as well to the lecturers and staff members behind the Masters programme for Atlantic Technological University, as well as the prior Bachelors programme in Software Development, for their development of the programmes as well as providing resources and materials for modules supporting the programme and the years prior.

And finally, I would like to thank colleagues, friends, and family back in Thailand for their support throughout the project, the Masters programme itself and the past Bachelors programme, both financially and personally. Without them, this project would not be possible.

Chapter 1

Introduction

Recently, there has been an increase in machine learning and computer vision when it comes to their uses in real life, including gesture recognition and transportation analysis. One of the key fields in computer vision would concern traffic analysis as well as identification of each public transport modes, including buses, taxis, and more. With this, there are also related concerns in regards to public transport as well, especially in buses, where there can be countless times that the buses would not arrive on-time, and would arrive either late or early, which does not correspond to what it's indicated in the timetables of those buses.

With that concern in mind, there are numerous ways to tackle this issue. One of which is via identification of those buses or other public transport modes. This can help users and commuters know whether a vehicle arrives at that specific point or not. This can be used for journey planning as well as further analysis on whether the area needs more buses or not. While there are systems in place over the past couple of years, such as GPS-based tracking or manual reporting, another way to tackle this issue is by using computer vision to identify whether a bus would arrive in a specific place or not. With the rise of mobile phones, this technology can also be adapted to be used in this way. This can make it easy to setup and does not require a need of other external devices.

In addition to computer vision, its capability of object detection and tracking are also beneficial as well, determining the direction of the traffic itself and potentially estimating arrivals of a particular targeted vehicle. Numerous approaches have been done in the past, including You Only Look Once's ID-based object tracking or centroid object tracking. Centroid tracking involves calculating the centre point of each detected object across frames and using Euclidean distance to associate detections over time. However, there are also numerous other ways as well, including tracking with sizes of bounding boxes and more.

With those in consideration, the project would explore the feasibility of using computer vision in identification and tracking of objects, particularly vehicles and

traffic, including mobile and lightweight implementation of computer vision as well as reporting of data gathered from computer vision itself.

In terms of the structure of the project, it is separated into two parts. First part being the computer vision itself, using machine learning methods and models that are popularly and suitable for the objective in mind. These include You Only Look Once (YOLO) and EfficientDet. Both of which are capable for object detection as well as producing bounding boxes and identify the category of each of those bounding boxes. These models worked in different ways, with YOLO being a single-shot convolutional network, predicting bounding boxes for each grid cell in the network, while EfficientDet uses a weighted bi-directional feature pyramid network (BIFPN) to fuse different feature levels together to form a prediction.

These models are implemented in two further methods: application on a mobile device using the EfficientDet model and a Python script using the YOLO model. These two methods would produce the same output of the category and direction (of where it is going towards) of the object as well as the timestamp and location of the device.

The second part of the project concerns the reporting of the output produced from the computer vision part. This includes database storage of those output and displaying results in a web application, hosted in a third-party service.

1.1 Objectives

In this project, three main objectives were proposed, concerning the development of the application and analysing the effectiveness of the models used:

- Develop a software application and a script that detect vehicles, including cars, cyclists, and buses, or objects, to be used in either a mobile device or a video streaming device (web camera)
- Develop and deploy a web application that visualise the results in a suitable manner, including a backend database to support the application.
- Compare the effectiveness and methods between using an EfficientDet model in a mobile device and using a YOLO model in a web camera device.

For the first objective, there were two software applications and scripts developed with the intention of object and vehicle detection. The first of which concerns a mobile application that made use of the EfficientDet model and the React Native backend, in which the EfficientDet model would be fine-tuned for the use of vehicle detection. The second of which concerns a Python script that made use of the YOLO technology and model.

The second objective concerned the web application output, which made use of a React framework (Next.JS) with a non-relational database (MongoDB) attached to the web application via a middleware (Node server with Express library) connecting between the database and the web application, as well as the database and the object detection portion of the project. These were also deployed on a cloud service as to enable the data exchange for the mobile device.

As described earlier, the third objective concerned the comparisons between the fine-tuned EfficientDet model as well as the YOLO model, in which the two models were compared based on their confidence level as well as their accuracy in identifying objects. Any trade-offs were also considered in the comparisons as well, including the performance of the models themselves.

1.2 Structure

This thesis is separated into five further chapters. Chapter 2 describes the methodologies used for the development of the project, including the development of the components of the project and research methods used accompanying the project itself. Chapter 3 details the research and literature review conducted for the project, including any other approaches of the subject matter and related work to the project. Chapter 4 details the system design of the project, including the principles behind the project, descriptions of each component of the project and their justifications. Chapter 5 summarised the results of the development of the project, compared to the three objectives set earlier as well as describing any challenges faced throughout the course of the project. And finally, Chapter 6 would conclude and summarise the project itself.

Chapter 2

Research Methodology

This chapter follows the methodologies in conducting this project, separated into two methodologies: development and researching.

2.1 Development

With the objectives considered, this research project is concentrated on the software development side of the project itself. With this, a modified software development lifecycle methodology is used, with a pre-planning process in the beginning, listing down all the requirements of the project itself, and separating into two distinct milestones or components:

1. Object detection software and script = this milestone or component focused mainly on developing an Android application for the project as well as a Python script as well. This also includes fine-tuning of the models used behind each subcomponents

2. Web application and database = this milestone or component focused on developing a full-stack web application, displaying the output from the Android application and the Python script, including the database behind the web application, the middleware, and the deployment of all the subcomponents in the milestone.

These two milestones or components were flexible in time length, due to varying difficulty levels of the projects, including trail and error aspect of the first milestone, in which different technologies were used to see what would be the best approach of the project. These milestones or components may also overlap or have its requirements as deemed fit for a better version of vision of the components themselves, allowing flexibility when it comes to changes to be made.

For the first milestone, this involved the creation and development of said two applications as well as potential training the models as well, particularly with the EfficientDet-like model itself. This model was trained using real-life traffic footage from a mobile phone. For the milestone to be completed, applications would have to be working as intended and suitable for comparisons between the two approaches. These comparisons would then be made via each of the model's mean confidence score as well as their framerates while predicting the same video stream. This video stream would be live for the EfficientDet-like model, but recorded for the YOLO model.

For the second milestone, this involved the web application development for reporting of data. Considerations in the web application would be done in terms of modularity, accurate report of data as well as some customisations and options to display the data in a certain way, such as viewing the data separated in certain time period.

2.2 Research

The research component for this project consisted of two different methods:

- Internet resources research - this involved documentations for each technology used in the project as well as guidance towards completing the development side of the project
- Literature research - this involved researches and studies done in regards to subject matter, including approaches of computer vision and object detection

In detail, internet resources research are done throughout the course of the development of the project, in getting guidance for creating and developing components needed to fulfil the requirements and objectives of the project. This can involve reading through extensive documentation of libraries and technology used as well as third-party guides and artificial intelligence in regards to subject problem matter. Some documentation would also be referenced in the project, if they provide substantial content and data to the subject matter, including metrics and analysis.

In terms of the literature review, this included conducting literature review, going through past papers in regards to computer vision and object detection as well as any supporting evidence and justifications on certain subjects. The literature search is done in two different databases of *IEEE Xplore* (`https://ieeexplore.ieee.org`) and *Google Scholar* (`https://scholar.google.com`), in which *Google Scholar* would also lead to other scientific journals website as well, including *ScienceDirect*, *arXiV*, and more. Filters and boolean operators of AND, OR, and NOT are also used.

Based on the subject matter of the project, some keywords are deduced, with each search query being a combination of one or more keywords: “computer vision“, “object detection“, “YOLO”, “EfficientDet”, “vehicle detection”, “traffic analytics”, “mobile computer vision“, and “phone computer vision“. The keywords for the web application part is not included, but reporting of data for each paper is considered.

The papers from the conducted literature review are also selected based on other inclusion and exclusion criteria:

- The papers should be written in English
- The papers should cover the subject matter described in the project and in the keywords described earlier. Any papers reporting quantitative metrics are heavily considered to be included.
- The papers should be released between 2015 to include recent attempts of computer vision and object detection. Exceptions would be made if the work is substantial for the subject matter.
- The papers would not be considered, if they are not related to the topic of this project or lacking substantial evaluation of the subject of the particular paper.

These papers would then be screened and skimmed, first by the title, the abstract, the introduction, the conclusion and then the entire content. If any of the papers do not pass any of the sections, then the paper would not be considered. The entire content would then be read-through to provide justifications and directions of the project itself.

Chapter 3

Literature Review

Based on the literature search, nine papers were selected, discussing the themes and topics behind the project, including object detection models used, deployment strategies for said models, and various adaptations of both EfficentDet and various versions of YOLO.

3.1 Object Detection Models

In terms of object detection models, papers selected involved either using the EfficientDet model or using the YOLO technology, or using a combination of both.

In a paper from Lin *et al.*[4], it discussed the implementation of various EfficentDet models with a comparison to YOLOv3 model, using a CCTV-like camera footage to analyse traffic on a top-down view of a road. This also included a proposed real-time speed calculations and traffic scene flow statistics, as well as going in-depth on the Bi-Directional Feature Pyramid Network (BiFPN) backbone technology. This paper also showcased that the mean accuracy precision (mAP) of EfficientDet would also increase, in a trade-off of a decrease in FPS. However, it also has shown that YOLOv3 would suffer in FPS metric for predictions and tracking. While this has shown that EfficientDet can be efficient for this scenario of traffic analysis, there were no other showcase on other factors such as with the proposed traffic analysis metrics mentioned earlier.

Another paper from Amulya *et al.*[5], the paper discussed using YOLOv3 for traffic density estimation, going through the logic and proposed architecture on applying YOLOv3 to the detection of a vehicle and counting them. The paper also showcased the metrics collected from the number of vehicles detected in a 15-minute time period and the average speed of each vehicle in that same time period. The paper also suggested on how YOLOv3 is more superior than EfficientDet due to its ability to detect fast moving vehicles. However, this paper lacks analysis in

terms of accuracy and precision from the model itself, as well as more discussions on other time periods, such as 1-minute or 5-minute time periods, as mentioned on the paper itself.

A comparison has also been made between the two models in a paper from Rajendran & Ganapathy.[6] It discussed the architecture behind YOLOv4 and EfficientDet, including the backbone network, neck, and prediction head of each of the models. The paper also did testing for each of the model types of the two models, concluding that the YOLOv4 is the one with the most mean average precision and a good framerate during runtime, and therefore the best suited model for their testing in traffic sign analysis. However, the paper dismissed considertaions in terms of deployment in other devices such as in mobile devices or a micro-processor.

In terms of the combination of two models, there are some papers proposing a used combination of the two models. One paper from Rajan *et al.*[7] proposed an integration of EfficientDet and YOLOv4 instead of using either one, which resulted in a slight improved mAP value of 0.85 out of using standalone ones (0.78 for YOLOv4 and 0.82 for EfficientDet). While the integration can also be considered, but this would require the two models to be ran simulateneously, which can increase computation power needed in a single device.

Another paper from Lv *et al.*, has discussed another combination of the two models on a Forward-Looking Infrared (FLIR) dataset, where the two predictions from both models are combined using a non-maximum supression (NMS) algorithm. From that algorithm, the maximum confidence score is kept as the final prediction. The results from the paper itself showcased that the ensemble of EfficientDet and YOLOv8 performed better in terms of all metrics of recall, precision, F1-Score, and mAP, which is similar to the paper from Rajan *et al.* However, concerns were still raised about the computational complexity of using two models together.

There is also a hybrid approach in a paper from Kar and El-Sharkawv,[8] where a visual transformer is used as the backbone alongside the BiFPN component from the EfficientDet model. This resulted in a better mAP value of 86.3, the most out of any other approaches on the paper, including YOLOv3, Faster R-CNN, SSD, and more. However, this paper did not discuss the implementation in limited computation scenarios, such as on mobile devices or microprocessors.

3.2 Mobile and Edge Deployment

In terms of the deployment, more specifically the object detection, there were some implementations of object detection on both microprocessors (Raspberry Pi) and mobile devices.

In a paper from Munteanu *et al.*[9], the paper used a combination of EfficientDet for underwater mine detection and YOLOv5/SSD for floating mine detection. These were deployed on microprocessors of Raspberry Pi, in which it took around 2 seconds per frame for each detection. With Raspberry Pi's computing capabilities, this paper did not went through some other approaches in regards to making those three models ran in a low-computing device such as one used. However, the mAP results were satisfactory in a range of 0.55-0.87 across the three models.

Another approach was done in a paper from Mohamedon *et al.*,[10] in which TensorFlow Lite (now known as LiteRT) models are used instead. The model is then deployed to an Android application on a mobile device. The application itself is also made use of another native development approach of Java development, instead of Kotlin or Expo React Native. With the application and the model used, it resulted in an accuracy of 98.25%. However, the model's prediction is made from taking a picture from a camera, which can skewed the results to be better on accuracy, rather than efficiency in a live video setting.

3.3 Research Gaps

The gaps presented throughout the literature review were presented, including the comparisons in terms of object detection deployment as well as object tracking itself in a traffic scenario.

Firstly, throughout the papers mentioned, there have been attempts of implementing computer vision within video streams of a CCTV/fixed camera or a microprocessor such as Raspberry Pi. However, only one has concerned using a mobile phone for computer vision, and that it only analyses still images instead of a video stream. With this, there's a possibility of an analysis on implementing computer vision on both a mobile device as well as in a higher performing device such as a laptop linking up with a Python script for computer vision. While it can be expected that the mobile device would perform worse in terms of accuracy and latency, but the field itself can be something to explore in the project.

Secondly, not too many papers mentioned about object tracking, especially with the direction of where the vehicles or objects are going. One paper from Amulya *et al.*[5], come close to analysis in terms of traffic density, but did not cover the direction of where the traffic would go. Instead, it is focused on just one specific portion of the road itself. With this, object tracking would be something to explore on, in order to enhance the computer vision side of the project and enable more potential future analysis of data.

Chapter 4

System Design

4.1 Principles

For this project, there were some key principles that were used in regards to the design of this project.

One of the key principles that were maintained throughout was the modularity of the project itself, promptly due to the separation of two distinct components of object detection and web application, as well as subsequent components, including a middleware to connect the two parts together and the connection to the database of the project. This made the debugging of the project easier and focused on fixing individual components themselves, rather than combining and mashing all the components together.

Another principle is the cross-platform nature of the project itself, with the web application itself using the framework of React NextJS. This framework allows multiple versions of the web application itself to be displayed amongst individual devices. The object detection component is also capable to work in multiple mobile platforms as well. Although not demonstrated in this project, the Expo React Native framework used can promote usage of the application in Android and iOS devices. However, there was a limitation to the object detection component on the Expo application due to the usage of LiteRT and its corresponding library, where object detection would not work on browsers, but only on mobile devices.

With mobile devices in mind for object detection, one principle is also exercised of efficiency, where lightweight models are utilised, such as EfficientDet However, this came with a trade-off of accuracy as well, which could be improved by transfer learning to focus on detecting vehicles themselves. YOLOv11 is also used for the Python script portion of object detection which is another lightweight model used in the project, but with more accuracy and optimised for the purpose. These two models' lightweight nature helped with the real-time reporting of data, which is

crucial for the web application portion of the project. This also helped to give users real-time data for the purpose of traffic and bus analysis.

4.2 System Overview

As mentioned, this project is divided into several components, predominantly two sections of the project. Those being the object detection side of the project and the full-stack web application development side of the project. These are showcased in Figure 4.1 with the left side of the diagram demonstrating the object detection side and the right side of the project demonstrating the web application and database development side.

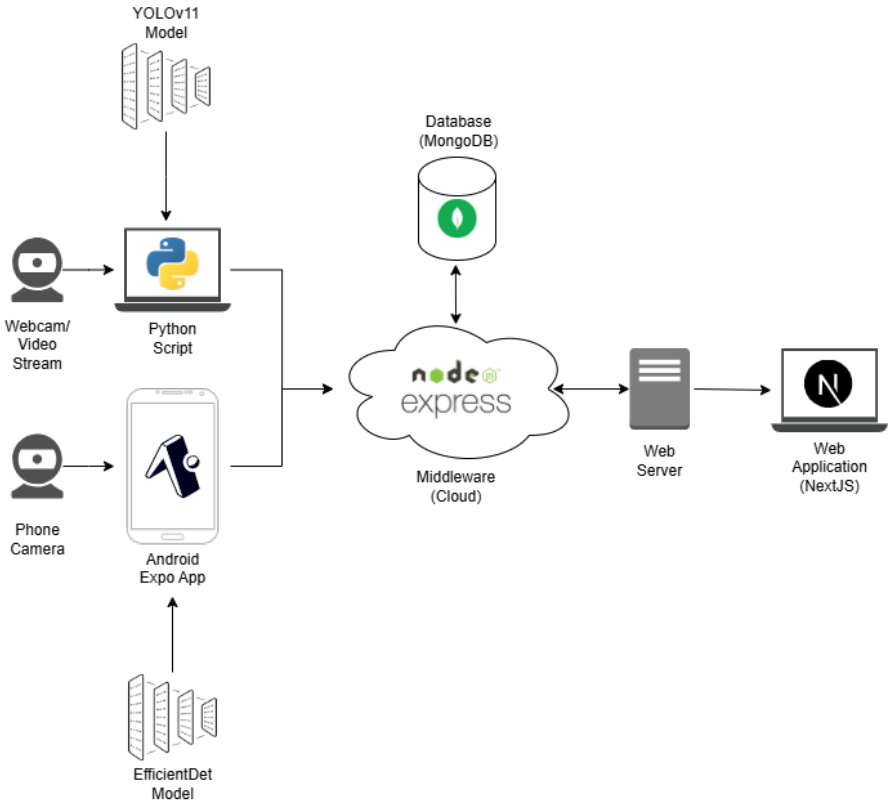

Figure 4.1: System Architecture of the project

In terms of the object detection side, it can be divided into two different sub-components, each representing a technology and model used for object detection. The first, being the YOLOv11 script itself, using Python as the back-end. This script would make use of an external web camera to help detect objects in real-time. For this to be adapted for vehicle detection, the webcam can be situated near

the roadside, or more preferably, using a recorded video stream of said traffic. This helps demonstrate the common technology used in object detection and tracking in the literatures reviewed.

The second sub-component of object detection represented the mobile implementation of object detection. This makes use of an Expo React Native framework, which help translates a web-based React project into an Android application, without a need of learning technologies behind creating an Android application, including Kotlin and the Android infrastructure itself.

For the full-stack web application development side of the project, there are three further sub-components that are implemented. The first of which is the middleware, connecting the object detection technologies to the database, and the web server to the database. These makes use of HTTP requests, in a form of API requests to the database itself. This is to allow a de-coupling and reliance on direct connection to the database, and helps maintain security within the database itself. This makes use of a NodeJS Express server connected to all the components mentioned.

The second part of the side is the database of the project where it makes use of a non-relational database of MongoDB. The document that the database stores concerned the category, timestamp, location, and direction of each object, sent from the object detection side. This is only connected to the middleware itself, for its own security of data.

The third part of the side concerned the web application. This web application served as the output of the data obtained in the database, including a stacked bar chart representation of the data collected in a specific time period. This will showcase on how many vehicles or objects exist within a time period and the direction that they are going on the camera feed or video stream. This made use of a React framework of NextJS, which included extensive debugging tools and customisations via technologies such as TailwindCSS.

4.3 Object Detection

4.3.1 Model Selections

As mentioned, two models are used in the project. Firstly, the You Only Look Once (YOLO) model, which made use of a region-based convolutional neural network that looked through an image only once in its forward propagation. This factor, alongside its incremental improvements, made the model as one of the common object detection methods in computer vision.

In addition, five models are also produced in this version including: `n`, `s`, `m`, `l`, and `x`, with increasing parameter count and accuracy for each model, in a trade-off

for speed. For this project, the `n` model is utilised as it has the fastest speed out of any models as well as a validation score that can be close to the EfficientDet model, for comparable results. These values are shown in Table 4.1 for each of the models, including the amount of parameters used, the latency in Open Neural Network Exchange's CPU (ONNX) on a validation set of COCO images as well as their validation scores on the same set.[2]

Model	Parameters / M	Latency (CPU ONNX) / ms	Mean Average Precision (mAP) Validation values
YOLO11n	2.6	56.1 ± 0.8	39.5
YOLO11s	9.4	90.0 ± 1.2	47.0
YOLO11m	20.1	183.2 ± 2.0	51.5
YOLO11l	25.3	238.6 ± 1.4	53.4
YOLO11x	56.9	462.8 ± 6.7	54.7

Table 4.1: YOLOv11 Performance [2]

The model itself accepts images and frames in a 640x640 image size as the default. However, the model can be adapted to be used in any resolution, depending on the video streams from the web camera or a video stream, as they would be downsized or upsized to be used in the model.

Another model that is used in the project was an EfficientDet model, which is based from the EfficientNet model, including the bi-direction feature pyramid network, featuring fusions between each feature level (with each being half of the original resolution)[1]. Another step also included for the model was a detection head to produce class scores and bounding boxes of the objects, in a similar sense to the YOLO model. This workflow is showcased in Figure 4.2. With its conversion to LiteRT (formerly known as TensorFlow Lite) model, this model is also This model would then be converted to a LiteRT (formerly known as TensorFlow Lite) model, optimised for mobile devices in terms of latency and size of the model.

Considerations and an approach of transfer learning for the EfficientDet-like model was also taken due to performance in object detection for a standardly trained EfficientDet model, as it is trained for the original purpose of general object detection. This can be disadvantageous as it doesn't the specific approach of traffic analysis of this project, thus producing inaccuracies for each detection. To accomplish this, additional training data was collected and passed through a higher accuracy model (YOLOv11) and annotated to be used in training the EfficientDet-like model. This would also provide an equal ground of detection between the two models used in the project itself.

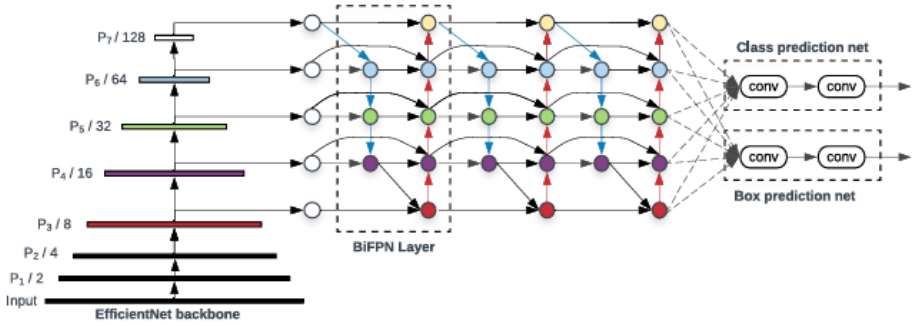

Figure 4.2: EfficientDet Architecture [1]

The base model, EfficientNet, had a similar setup with multiple models based on the number of parameters used, ranging from B0 to B7, with 5.3 million parameters for B0 to 66 million parameters for B7. With this, the base architecture chosen in the transfer learning attempt is the `B4` architecture of the model, as has a middle ground of trading off accuracy of the model and the efficiency of the model, basing on EfficientDet's validation scores for D0 to D7, where the number represented the corresponding checkpoint number in EfficientNet. These scores and their parameters used are shown in Table 4.2, noting that the D7 model has more multiply-adds than D6 despite the same number of parameters. However, the validation scores can be lower due to their usage in mobile devices, trading off for better performance.

Model	Parameters / M	AP Validation values
EfficientDet-D0	3.9	34.3
EfficientDet-D1	6.6	40.2
EfficientDet-D2	8.1	43.5
EfficientDet-D3	12	46.8
EfficientDet-D4	21	49.3
EfficientDet-D5	34	51.3
EfficientDet-D6	52	52.2
EfficientDet-D7	52	53.4

Table 4.2: EfficientDet Performance [1]

The transfer learning model itself would accept 512x512 image size as with the normal EfficientNet model, accepting a similar image size. These image sizes would then be broken down in several feature layers in a BiFPN block, which then fused into an output, containing the class and the bounding box of the objects.

However, problems arose during the development, due to the original EfficientNet model and a different training architecture compared to the available ready-to-use EfficentDet model. With this, the original EfficientDet LiteRT model is used instead, with its `lite3` variant. The mAP values and 1 CPU latency (from Pixel 4 phone) for each LiteRT variant are displayed in Table 4.3. These issues would be discussed in the next section.

Model	mAP / %	Latency (1 CPU) / ms
lite0	25.69	49.92
lite1	30.55	91.87
lite2	33.97	144.24
lite3	37.70	264.39
lite3x	40.98	455.17
lite4	41.96	605.8

Table 4.3: EfficientDet (LiteRT version) mAP Values [3]

From these two models, the models are used in different demonstrations, with YOLOv11 model being used in a Python script and the EfficientDet model being used in an Android Expo application. For YOLOv11, it is very optimal to be used in a higher computation device than mobile phones, hence making use of a simple web camera and laptop setup, as well as support for local video streams as well for testing purposes. For EfficientDet model, with the conversion to a LiteRT model, this allowed for an optimal implementation of object detection in mobile devices, which in itself required less computation than the original YOLOv11 model. However, a significant trade-off in accuracy was accounted as well due to the performance and computation capabilities of mobile devices themselves.

4.3.2 Tracking

The two models exhibit different tracking methods. For YOLOv11, this is done by tracking via association of the ByteTrack algorithm, using bounding box overlap and motion prediction using Kalman filters, fusing an actual measurement of the object and the prediction of the object. The object being tracked is then assigned a unique ID for persistence as long as the object is on the screen. This algorithm is optimal and built-in to the library of where the model exists, which made object detection and tracking easier.

For the EfficientDet model, tracking is done with a custom algorithm that measures the differences between the bounding boxes of each frame, where the object in between two frames would be the same if the latter frame has a 30% height difference between itself and the former frame. After determining if the

object in each frame is the same, the direction of the object in the latter frame is determined depending on the difference between the left side of each box and the difference between the right side of each box. The actual direction would then be deduced based on each subsequent frames. After which, the objects are compared together to determined if they are disappeared or not, with a 10-frame fallback for the same object. The algorithm determining this is written in Listing 1.

This custom algorithm can be deemed as complicated due to its reliance on both the bounding box of the objects in each frame as well as the frame rate of the camera itself, as the speed of each object may not be able to detect if the object is on the screen or not. This would also be dependent on the accuracy of each detection as well, as one object detected in the previous frame might not be able to be detected in the next frame. This is where YOLOv11 would have an advantage on in terms of predicting the direction of where the object is going, but comes at a cost of performance and efficiency of the model, especially with a computer vision model running on a mobile device.

4.3.3 Optimisations

When it comes to the optimisations of the implementation of computer vision, in terms for EfficientDet model, Expo React Native is used with a purpose of EfficientDet's mobile implementation and its capability of cross-platform development. While native development using Kotlin can be done for this subject matter, but the familiarity of web-based syntaxes can outweigh that development approach. This also sped up the development process rather than starting fresh in an unfamiliar framework.

In terms of Expo React Native, several libraries help to achieve object detection and tracking in the application, including `react-native-fast-tflite`, which makes use of a LiteRT (formerly TensorFlow Lite) model to be used in machine learning in mobile device. While this would not allow for the application to be run on browsers, but they are really optimised and useful for the targeted approach on mobile machine learning and object detection.

Another considerations are also taken in terms of the format of the model between the described LiteRT model and TensorFlow.js library using a JavaScript TensorFlow model. Despite its capability of usage in a web application via JavaScript syntaxes, LiteRT is optimised solely for usage in mobile devices, which is the aim of the mobile application itself. Including that, running the model (via TensorFlow.js) in web applications such as *Google Chrome* or *Samsung Internet Browser* on *Samsung* devices, can pose a challenge in balancing between accuracy and speed of the model itself, as web resources may not be optimised for web browsers in mobile devices. Difficulties in integrating with mobile's camera are also not optimal for the usage of the model via TensorFlow.js. Hence, the direction of developing an

```
Let HEIGHT_TOLERANCE = 0.3
Let FRAME_FALLBACK = 10
For each f:
    Get the objects detected in f as curr
    Get the objects detected in f-1 as prev
    Let checked as blank array
    If objects in f-1 is more than 0:
        For each obj_curr in curr:
            Initialise obj_curr frame fallback to 0
            For each obj_prev in prev:
                Get heights in obj_curr and obj_prev
                Get difference percentage in heights
                If difference less than or equal to HEIGHT_TOLERANCE:
                    Set obj_curr checked to TRUE
                    Set obj_prev frame fallback to 0
                    Get difference in left side box in obj_curr and obj_prev
                    Get difference in right side box in obj_curr and obj_prev
                    If left difference and right difference are less than 0:
                        obj_prev LEFT direction count added by 1
                    If left difference and right difference are more than 0:
                        obj_prev RIGHT direction count added by 1
                    Else:
                        obj_prev STILL direction count added by 1
                    obj_prev is pushed to checked
                    Break
            If obj_curr checked is FALSE:
                obj_curr is pushed to checked
        Filter prev that is not in curr
        If filtered prev is more than 0:
            For each obj_dis in filtered prev:
                If obj_dis frame fallback is less than or equal to
                ↪ FRAME_FALLBACK:
                    obj_dis frame fallback count added by 1
                    obj_dis pushed to checked
                Else:
                    Get final direction from direction count
                    Send data for obj_dis
    Else:
        Set prev to curr
    prev is set to checked for the next f
```

Listing 1: Custom tracking algorithm for EfficientDet model

Android application using LiteRT models, but with a web-like Expo React Native framework, rather than native development.

4.4 Web Application Development

4.4.1 Middleware

As a connection between the object detection part of the project and the database, as well as a connection between the web application development and the database, a middleware service is created via a simple Node Express server. This server handles JavaScript Object Notation (JSON) HTTP requests from both parts in connection with the database of the project. Two HTTP requests were created:

- POST request - each component of the object detection part would send a POST request containing the object's timestamp, location (latitude and longitude), category, ID, and direction of where the object is going.
- GET request - returning all instances of the objects inside the database, containing all the information above

The separation of database connection into a middleware helped with the modularity principle of the project, where the database itself can be switched out anytime or its interaction mechanics (via the requests) changed at anytime, without changing the structure of the other parts of the architecture.

However, this comes with some security concerns such as with the sensitivity of the data itself, especially with the location of where the object is detected, and the accessibility of the database itself by those request methods. These can be promptly fixed by introducing authentication to the server and encryption of location data. However, these are not explored and out-of-scope of the project, with the main focus of the project being the object detection part and the display of such data.

4.4.2 Database

In terms of the database used for this project, a non-relational database of MongoDB is used. This database is described as a document-oriented database, allowing for flexible data models to be stored in each database. These can either be the same structure as with a table in relational database or a loose structure with each document (or entry) having different models or fields. This promptly allowed for flexible development for changing requirements of the project itself, as it evolved throughout the course of the project.

While relational databases such as PostgreSQL or MySQL can be used in this project, but the requirements of the project would be changing overtime, even if this project was to be picked up again in the future. Hence, the need of flexibility of the database itself, in terms of what are stored and the usage of itself. Including

that, relational databases also are complicated in terms of its joining mechanisms and foreign keys across each table in the database, which can be complicated compared to a document model of MongoDB. The document model can be easier to interpret due to its structure similarity with JSON, and thus can be friendlier to web application developers. MongoDB's ease of access as well as flexibility heavily influenced the choice of database in the project. However, there can be some concerns in regards to its fail-safe mechanisms such as with backup managements and rollbacks, but this is not a concern for a smaller-sized project.

4.4.3 Web Application

The web application of the project is made use of React NextJS framework. The React NextJS architecture itself is a component-based architecture, allowing separation of each module of the web application. This separation itself exercised the modularity principle of the project itself, isolating components for easier and targeted debugging.

The modules themselves include the stacked bar chart, displaying the amount of detections per time period (including categories detected) as well as a table listing all the entries, depending on a selected bar in the chart. These information allow the users to inspect on how many objects exist in a time period, hence also the traffic that can be reported on the web application. The modules on the web application are displayed in Figure 4.3, displaying example entries from testing.

Another web application framework is also considered of Angular as well. While it is preferred to use Angular for modularity in web applications, React NextJS allow for more easier integration with other external libraries such as Chart.js, which displays the stacked bar chart or TailwindCSS for styling, compared to Angular's templating system. NextJS also allowed for server-side rendering which is optimal for connection with the middleware (containing the connection to the database) on the same cloud service, and only loading necessary components to the user. This also exercised the dashboard and real-time data reporting aspects of the application as well, as it decreased the load time needed to report the data from object detection, despite a set 15-second delay.

4.5 Deployment

All of the components in the full-stack web application development side of the project, including the Node server for the middleware and the NextJS web application, are deployed into an Amazon Web Service (AWS) EC2 server. This deployment helped link the end-devices of the two object detection approaches together and allow for cloud-based testing and usage for the Android object de-

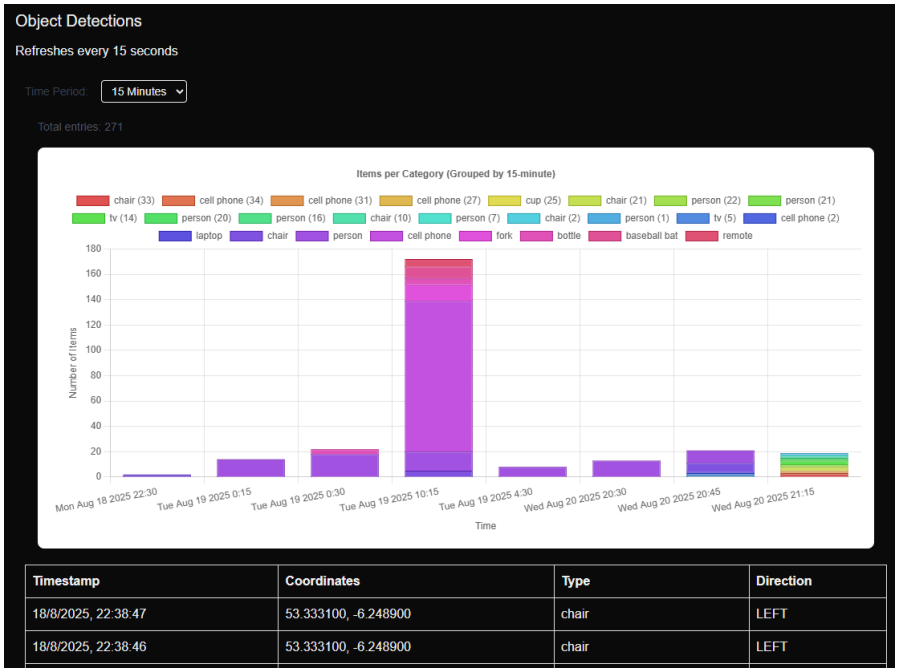

Figure 4.3: Web application overview

tection application. This was due to the application being deployed to a mobile device and thus is separated from the host device when testing locally.

Some concerns that were considered were about security issues of the application. This can include the sensitivity of the location data in the database as well as connections with the middleware, the database and the web server. However, these are not considered for the project due to the nature of demonstrating the object detection technology and display of data from said technology.

Chapter 5

System Evaluation

This section evaluates the development and researched conducted during the course of the project, based on the three objectives given at the beginning. Those three being the development of object detection systems, the development of a reporting web application, and the comparisons of the two models of YOLOv11 and EfficientDet.

5.1 Object Detection

The implementation and execution of the two object detection approaches of YOLOv11 and EfficientDet had some success, in regards to detecting objects and vehicles that appear on screen as with the initial intention.

For YOLOv11 Python script, with the nature of the model being adaptable for local and live video streaming, the model was able to detect, predict, and track the objects on those streams, with adjustments as to how to detect if the object is off the screen and tracking the direction of the object using centroid tracking. Figure 5.1 showcased on how YOLOv11 works with a local video stream.

However, with its high accuracy, the model and technology itself would not work in the Expo React Native app, as it required a lot more computation to run the model with similar results as to one that was displayed on the Python's script version. One way to potentially circumvent this could possibly be transmitting the video data into the cloud service instead of doing the detections on the mobile device. This might help offload computation needed to run the mobile application.

This approach would come with some trade-offs in terms of the amount of bandwidth required to send to the cloud service and receive in the cloud service. This may incur some additional costs to the cloud service itself if the application is running for a very long time, compared to only sending in the prediction data. Another trade-off would also be on the bounding boxes displayed on the video

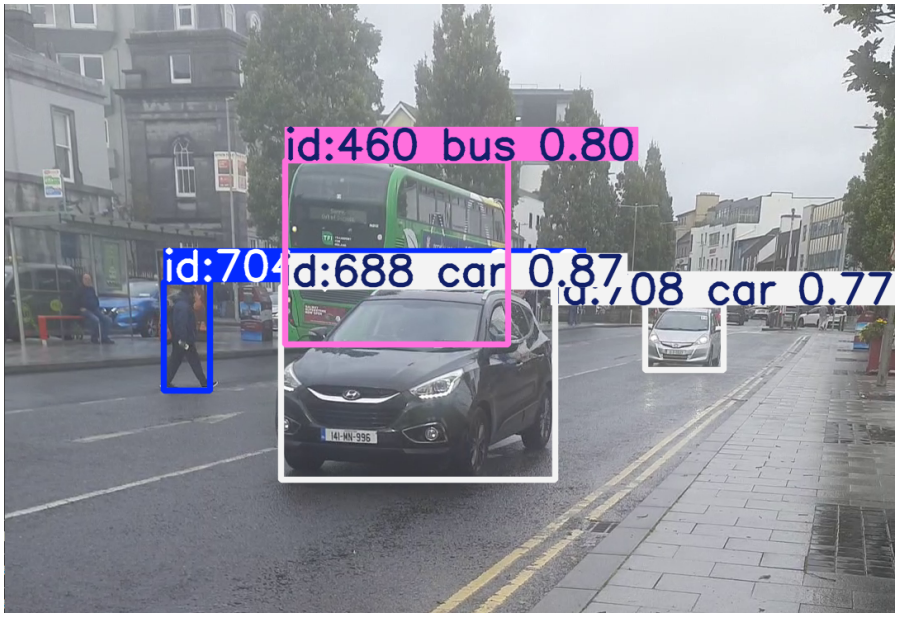

Figure 5.1: YOLOv11 Detection

stream, as either the application would not be able to display them or risk having latency by retrieving those bounding boxes back to the application. In this case, this would really depend on the connection speed and signal of the internet connection itself, hence potentially increasing the latency and speed of prediction.

In terms of the EfficientDet model training, an attempt was made in using transfer learning to fine-tune the model and use the exact same principle in EfficientDet in an EfficientNet model. This was done by retrieving the EfficientNet model, and adding in features of a BiFDN technology and detection head, printing out the exact same parameters such as ones in provided in Kaggle.[3] However, the attempt was made to be convoluted in implementing and running the resulting model itself. Including this, the resulting model's detection head from the transfer learning method is also different than the one pre-made in Kaggle, seen in both Figures 5.2 and 5.3. The transfer learning model also performed worse in terms of performance and detection in general as well, hence justifying the use of the original pre-made model.

Nevertheless, the Expo React Native application handling the EfficientDet model also worked as intended, in terms of displaying the frames per second for the video stream, the mean precision score, and the bounding boxes. This showcased that object detection and tracking can be implemented in a web syntax-like framework such as Expo, rather than tackling on native development, which can be difficult for people who are used to web-based syntaxes. With this, there is a

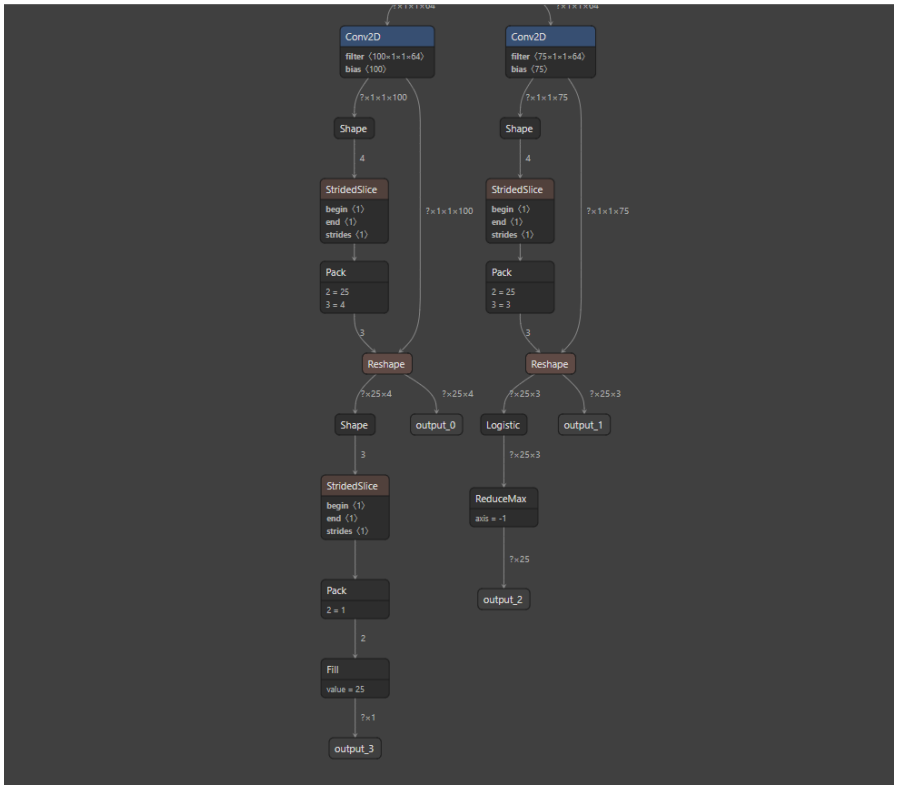

Figure 5.2: EfficientDet-like Model Head Using Transfer Learning

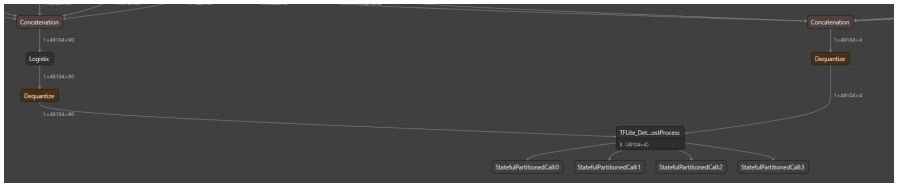

Figure 5.3: EfficientDet Model Head

potential to explore more on object detection and tracking in mobile devices using Expo, apart from what is showcased in the project itself. Figure 5.4 showcased the application itself detecting objects on a mobile phone camera.

5.2 Web Application and Database

With the middleware and the database setup of the project, this turned out to be a good choice in terms of modularity of the project itself, especially with a potential of having various other sources of data. However, this came with security concerns.

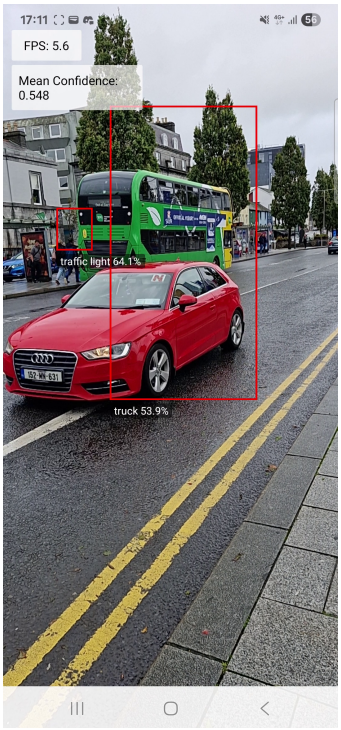

Figure 5.4: EfficientDet Detection

These can include the sources and validity of data itself, whether it is from a legit source from a traffic site or not. Another concerns can include the sensitivity of data when gathering the data from the database via the middleware, as well as the exposed IP address for HTTP requests. Methods such as authentication or encryption can help in maintaining security of the data themselves as well as the database itself.

In terms of the main web application reporting itself, the web application displayed the data as intended with the bars showcasing how many vehicles or objects appear in a specific time period as well as its location and direction when inspecting each one of them. However, more steps can be taken to display the data in a wider range, such as a map with the location of the object pinned, or having a heat map of where a spot contained the most number of detections. (which in itself, can make the results more interpretable) Despite this, the web application itself proved to be able to showcase real-time data (with a 15-second refresh delay) of objects detected from the object detection part of the project. This could realistically be used in informing users as to where a spot is crowded or not. Traffic density calculations can also be used as well, in integration with this approach.

For the deployment of the middleware, database, and web application, AWS is deemed to be a good choice, especially with its Elastic Compute Cloud service (EC2) with lower costs to maintain a simple server. While other services such as Google Cloud Service or Microsoft Azure can be used, AWS can provide the same features from the other two realistically. Apart from that, the three services were setup to be ran without any restarts, and no scripts to maintain them. While this approach is enough for the project itself, this may not suffice in a production scenario, as restart procedures and error mitigation have to be done to maintain those services up. To achieve this, other technologies from AWS can be used, such as Lambda for accepting inputs or HTTP requests, or DynamoDB for the database service, rather than MongoDB.

5.3 Model Comparisons

When it comes to comparing the models of YOLOv11 and EfficientDet, the metrics of frames per second (during prediction) as well as the mean confidence for each method are measured. The models' predictions were based on the same exact footage, with EfficientDet's predictions being done in real-time and YOLOv11 being done based on the recorded footage. Table 5.1 summarised the metrics of each of the model.

Model	FPS	Mean Confidence
YOLO11n	8.84	0.7514
EfficientDet-lite3	5.5	0.514

Table 5.1: YOLOv11 v EfficientDet Performance Metrics

From the results themselves, it displayed that the YOLOv11 model performed better in both metrics. This can be possibly due to its one-shot approach of detecting objects in its convolutional neural network, which can make the the model more efficient in that way. However, this model is ran through a higher-efficiency device of a laptop, in which the model may suffer from performance issues when being done on mobile devices like in EfficientDet. As with EfficientDet itself, the results were worse than YOLOv11 due to its focus on the size and efficiency of the model itself, being able to be ran on a mobile device, and barely runs on the device with 5.5 frames per second.

When it comes to the objects detected themselves, the YOLOv11 model did significantly better in terms of the number of objects that the model can detect as well as the accuracy in terms of what are being detected or tracked, including objects outside the traffic itself. In contrast, the EfficientDet model did worse when it comes to detecting passing objects on the screen, and produced a significantly

low number of detections compared to YOLOv11. The tracking of the objects themselves were also inverted, especially when it comes to the bounding boxes produced for each object. Including that, the objects detected were also inaccurate, as objects such as "truck" were detected instead of a car. All of these issues can be presumed due to EfficientDet's low mAP values for LiteRT models as they are trained for efficiency to be ran on mobile devices.

Some considerations, however, could have improved this testing workflow better, including restricting the objects detected to be solely based on categories relating to the traffic, as to see how the models would perform in a case of traffic analysis. Another consideration would also be the "grace period" for each object detected. This grace period determines the number of frames the object can disappear on the screen without being considered as "off the screen". With this, the grace period was set to 10 frames, which can be low for the YOLOv11, hence a potential to experiment around different values of grace period to find a suitable value. Another approach can be to keep track of the IDs that have appeared on the screen. However, this may incur some space complexity in terms of storing the IDs that are already detected, but can potentially be resolved with a sliding window approach of the range of IDs being tracked.

Chapter 6

Conclusion

This paper has discussed the approach of using object detection models of YOLOv11 and EfficientDet to report and track traffic in real-time, as well as reporting of them in a web application. With this, it can be deduced that there is a potential of using a model such as EfficientDet in a mobile device to help with traffic analysis, barring computational issues from the limitations of the device as well as inaccuracies and low precision from the EfficientDet model itself. More work on transfer learning may be able to help lift the barrier up for targeted detection of traffic itself.

Otherwise, in comparison within the two models, YOLOv11 is deemed to be a more precise model compared to EfficientDet, and would be a preferable model to be used in computer vision in this scenario. However, adjustments have to be made to accompany the YOLOv11 model to be used, including another approach of streaming the video taken from the camera to the model in the cloud.

With the web application itself, the architecture behind the web application as well as the middleware and database indicate a suitable approach in reporting the data to users interested in traffic analysis, especially with the modularity of them, isolating parts of each service. This would make the service more easy to maintain. However, the basic setup itself can pose security risk in terms of sensitivity of data itself. Other methods of displaying of data can also be explored as well, such as with maps, rather than the stacked bar charts. Nevertheless, the stacked bar charts itself can indicate a lot about how much traffic is at specific locations.

All combined, this project has some future explorable research that can be done in the area. One of which concerns other models that can be used, rather than EfficientDet or YOLOv11, such as Faster R-CNN or SSD, in which future research can be done to determine if other models may be more suitable for traffic detection and tracking on mobile itself. Another considerations can also be taken into using real-time object detection and tracking in a mobile device, rather than using microprocessors or solely from a web camera or CCTV camera. This approach also does not limit to just traffic analysis itself as well.

In terms of traffic analysis itself, other researches can be done to reporting them in multiple locations, as indicated already with the position gathering in the project itself, such as implementing them in various sections of the street and comparing them in real-time. This approach can lead to a lower-cost setup of traffic analysis, rather than spending on CCTV cameras for analysis itself, if the interested party also wants a visual response to the detections on each video stream as well.

Otherwise, this project explored the potential of using computer vision in traffic analysis, using existing object detection and tracking technologies that can be equipped in a mobile device. While the technology itself is constantly developing to become more accurate, this mobile device approach can help shape the computer vision and machine learning side of artificial intelligence to become more concerned in efficiency of the model itself. The implementation of said technology can also lead to easier access of analysis such as with the subject matter, without investment in any other devices other than a mobile device.

Chapter 7

Appendix

This project is developed in a GitHub repository, linked here: `https://github.com/duechayapolgmit/traffic-analyser` The repository contains four parts of the project, with each folder representing each of the component:

- `python` - containing Python scripts for the YOLO implementation and an attempt at transfer learning.
- `server` - containing the middleware server, using Node Express, connecting object detection methods and the web application to the database via an API.
- `traffic-analyser` - containing the Android application for the EfficientDet implementation, using Expo React Native.
- `web-app` - containing the web application of the reporting of data, using NextJS.

All setup instructions and further information can be found in the main README file of the repository as well as the README files in each of the folders. A screencast link is also linked in the repository.

Bibliography

[1] Mingxing Tan, Ruoming Pang, and Quoc V. Le. Efficientdet: Scalable and efficient object detection. *CoRR*, abs/1911.09070, 2019.

[2] Glenn Jocher and Jing Qiu. Ultralytics yolo11, 2024.

[3] Tensorflow | efficientdet (kaggle website).

[4] Kaihao Lin, Junyan Chen, Aoge Chen, and Hu Huang. Application of the EfficientDet Algorithm in Traffic Flow Statistics. In *2020 16th International Conference on Computational Intelligence and Security (CIS)*, pages 140–143, November 2020.

[5] Kotla Amulya, Mandarapu Sahithya, Mulka Trisha, Shanmugasundaram Hariharan, A V Senthil Kumar, and R. Karthikeyan. Efficient Traffic Density Calculation for Diverse Road Traffic using Yolov8 Algorithm. In *2025 IEEE 14th International Conference on Communication Systems and Network Technologies (CSNT)*, pages 535–539, March 2025. ISSN: 2473-5655.

[6] Shehan P. Rajendran and Sreelatha Ganapathy. Comparative Analysis of YOLOv4 and EfficientDet based models for Traffic Sign Detection in Autonomous Vehicles. In *2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT)*, pages 1–8, July 2023. ISSN: 2473-7674.

[7] V. Aravinda Rajan, Sridevi Sakhamuri, A Periya Nayaki, Swathi Agarwal, Anurag Aeron, and M. Lawanyashri. Optimizing Object Detection Efficiency for Autonomous Vehicles through the Integration of YOLOv4 and EfficientDet Algorithms. In *2024 International Conference on Trends in Quantum Computing and Emerging Business Technologies*, pages 1–5, March 2024.

[8] Shreyanil Kar and Mohamed El-Sharkawv. Object Detection Using Vision Transformed EfficientDet. In *NAECON 2023 - IEEE National Aerospace and Electronics Conference*, pages 214–220, August 2023. ISSN: 2379-2027.

[9] Dan Munteanu, Diana Moina, Cristina Gabriela Zamfir, Ștefan Mihai Petrea, Dragos Sebastian Cristea, and Nicoleta Munteanu. Sea Mine Detection Framework Using YOLO, SSD and EfficientDet Deep Learning Models. *Sensors*, 22(23):9536, January 2022. Publisher: Multidisciplinary Digital Publishing Institute.

[10] Muhammad Farhan Mohamedon, Faridah Abd Rahman, Sarah Yasmin Mohamad, and Othman Omran Khalifa. Banana Ripeness Classification Using Computer Vision-based Mobile Application. In *2021 8th International Conference on Computer and Communication Engineering (ICCCE)*, pages 335–338, June 2021.