الجامعة الإسلامية العالمية ماليزيا
INTERNATIONAL ISLAMIC UNIVERSITY MALAYSIA
Garden of Knowledge and Virtue
LEADING THE WAY
KHALĪFAH • AMĀNAH • IQRA' • RAHMATAN LIL 'ĀLAMĪN
LEADING THE WORLD
INTERNATIONAL MULTI-AWARD WINNING INSTITUTION FOR SUSTAINABILITY
KICT
KULLIYYAH OF INFORMATION AND COMMUNICATION TECHNOLOGY

Advancement in ICT: Exploring Innovative Solutions (AdICT)

Series 1/2024

KICT Publishing

ADVANCEMENT IN ICT: EXPLORING INNOVATIVE SOLUTIONS (AdICT) SERIES 1/2024

Editors
Noor Azura Zakaria
Dini Oktarina Dwi Handayani
Elin Eliana Abdul Rahim
Ahmad Fatzilah Misman

KICT Publishing

ADVANCEMENT IN ICT: EXPLORING INNOVATIVE SOLUTIONS (AdICT) SERIES 1/2024

First Publication 2024

Published by

KICT Publishing
International Islamic University Malaysia
53100 Kuala Lumpur, Selangor, Malaysia

e ISBN 978-629-99388-0-4

9 786299 938804

KICT Publishing

(Online)

Preface

Advancement in ICT: Exploring Innovative Solutions (AdICT) Series 1/2024 is an e-book showcases the collective achievements of Final Year Project (FYP) in Kulliyyah of Information and Communication Technology (KICT). This compilation represents evidence to the technical passion and academic skills of our students before they venture into the professional realm.

FYP is a journey that demands creativity, critical thinking, and perseverance. This book encapsulates the diverse range of projects undertaken by our students, each a unique exploration into the vast landscape of Information and Communication Technology (ICT). From cutting-edge software applications to groundbreaking research, these projects not only demonstrate technical proficiency but also the ability to address real-world challenges.

In this comprehensive collection, the topics covered span a spectrum from cutting-edge software development, cybersecurity, artificial intelligence and multimedia technologies reflecting the breadth and depth of our academic program. This offers a curated journey through the diverse landscape of final year ICT projects to the readers while appreciating the impact these projects can have on the wider community.

This e-book carries significant benefits and impact whereby it serves as a valuable knowledge repository, offering a diverse audience—from students and educators to industry professionals—a comprehensive view of the latest innovations and technological solutions in ICT. Moreover, the book fosters a culture of knowledge sharing and collaboration, as each project represents a unique contribution to the broader technological landscape.

"When the human being dies, his deeds end except for three: ongoing charity, beneficial knowledge, or a righteous child who prays for him" *– Sahih Muslim*

Editors

Noor Azura Zakaria
Dini Oktarina Dwi Handayani
Elin Eliana Abdul Rahim
Ahmad Fatzilah Misman

TABLE OF CONTENTS

A Machine Learning-Based Automated Vehicle Classification Implementation on Toll System in Malaysia: A Preliminary Study

Raini Hassan
Department of Computer Science
Kulliyyah of ICT, International Islamic University Malaysia
Kuala Lumpur, Malaysia
hrai@iium.edu.my

Aisyah Afiqah Mohd Ridzal
Department of Computer Science
Kulliyyah of ICT, International Islamic University Malaysia
Kuala Lumpur, Malaysia
aisyah.ridzal@live.iium.edu.my

Nur Zulfah Insyirah Fadzleey
Department of Computer Science
Kulliyyah of ICT, International Islamic University Malaysia
Kuala Lumpur, Malaysia
insyirah.fadzleey@live.iium.edu.my

***Abstract*— Congestion in toll plazas has prompted the exploration of various solutions, from infrastructure improvements to advanced technologies. Enhancing toll plaza infrastructure, such as constructing additional tollbooths and widening lanes while implementing electronic toll collection systems, has had some positive impacts. However, these existing measures have faced limitations in effectively addressing congestion. The use of mixed-mode lanes at the leftmost toll lanes still applied manual vehicle classification, which relies on human operators, but it has yet to sufficiently overcome congestion, given the diverse vehicle types and toll rates. This situation leads to human error and affects traffic flow. Although RFID (Radio frequency identification) technology has been widely adopted at only a few toll lanes, challenges in implementation have led to congestion issues due to insufficient infrastructure and reliability problems. Therefore, the outcome of this project is to develop the best model detector of automated real-time multiclass vehicle classification for all lanes in the toll plaza. This model input is extracted from a pre-trained 800 images, which consist of 6 classes of vehicles and their annotated XML file, respectively, for one stage detector: Faster Region-Convolutional Neural Network (Faster R-CNN), ResNet-50 and two-stage detectors; You Only Look Once (YOLO), YOLOv8 Darknet-53. The classification model performs well in YOLOv8 architecture with the highest mean average precision (MAP-50) of 95.0% and has a good performance measurement on loss function compared to Faster R-CNN architecture.**

Keywords—Vehicle Classification, Toll System, MLFF, RFID system, You Only Look Once, Faster R-CNN, Machine Learning.

I. INTRODUCTION

A. Current Toll System

The current toll collection system in Malaysia heavily relies on manual processes conducted by human operators to set toll rates for each vehicle passing through the leftmost lane's booth. This manual approach is susceptible to human errors and can lead to congestion at toll booths, resulting in inefficient highway traffic flow. Furthermore, an automated vehicle classification system must be revised to differentiate between various vehicle types accurately, leading to potential misclassification and inaccurate toll rates. These issues significantly hinder the toll collection system's efficiency, accuracy, and overall performance, negatively impacting road service providers and users. According to statistical data from the Malaysian government, traffic congestion during peak hours has been a persistent problem, causing substantial economic losses. For instance, in 2021, traffic congestion in the Greater Kuala Lumpur area alone resulted in an estimated economic loss of over RM 4.3 billion (approx. USD 1 billion) due to wasted time and fuel consumption [1]. Additionally, manual toll collection processes contribute to increased travel times, fuel consumption, and vehicle emissions, adversely affecting air quality and exacerbating environmental concerns.

B. Project Overview

The project aims to develop an automated vehicle classification system for toll collection in Malaysia that can operate accurately and efficiently in real-time while contributing to smoother traffic flow on highways. This project proposes the implementation of a computer vision-based solution using the Region-of-Interest (ROI) method for vehicle detection and extraction from video footage obtained from toll booths, combined with machine learning techniques which are Faster R-CNN and YOLOv8 algorithms for vehicle classification. Combining the ROI method and the Faster R-CNN or YOLO algorithm shows promising potential for accurately detecting and classifying vehicles passing through toll booths. Upon complete integration, the system can substantially enhance toll collection efficiency and precision, curtail operational expenses, and elevate the smoothness of traffic movement along highways. The project aligns with Goal 11 of the Sustainable Development Goals (SDGs), Sustainable Cities and Communities. Efficient transportation and toll collection systems are vital for sustainable urban development, as highlighted in SDG Goal 11. By accurately classifying vehicles based on their characteristics, such as size, number of axles, and type, the toll collection system can incentivize using cleaner and more fuel-efficient vehicles, reduce traffic congestion, and improve air quality. Thus, the project also contributes to other related SDGs such as Goal 13 (Climate Action), Goal 3 (Good Health and Well-being), and Goal 9 (Industry, Innovation, and Infrastructure) by promoting technological innovation and investment in infrastructure. Overall, the project intends to achieve an advanced and efficient toll system in Malaysia, bringing it in line with international standards.

C. Project Objectives

As stated, this project focuses on implementing machine learning to solve the problem statements. The main objectives are as follows:

1) Develop an automated image-based vehicle classification system for toll collection in Malaysia by implementing computer vision techniques, such as the Region-of-Interest (ROI) method, to accurately detect and extract vehicles from toll booth video footage as well as utilizing machine learning algorithms like Faster R-CNN or YOLO for efficient, and real-time vehicle classification.

2) Build a robust system that accurately classifies various vehicle classes based on their attributes, aiming to create an effective toll collection system that minimizes human errors in toll rate settings and reduces congestion at toll booths.

3) Align Malaysia's toll system with international standards and best practices to enhance the overall user experience by implementing a smoother and more efficient toll collection process.

4) Contribute to sustainable urban development and the accomplishment of SDG 11 (Sustainable Cities and Communities) by incentivizing the use of cleaner and more fuel-efficient vehicles through the implementation of an automated vehicle classification system, supporting sustainable mobility practices.

D. Significances of Project

The significance of this project is that the evolution of Malaysia in tolling technology to aim for Multi-Lane free flow (MLFF) since 2009 can be done practically by providing one of the best models to detect and localize vehicle classes by capturing real-time objects with high accuracy and speed performance. In addition, the system improves the current tolling system by adding new features and variables, such as an alert system when unclassed vehicles enter the toll. Meanwhile, different latest models of Convolutional Neural Networks-based approach, FRCNN and YOLOv8 (version 2023) performance are provided to compare which model performs well, and the ideal results in a detection model will perform well as it trains explicitly and validates Malaysia's vehicle datasets of different toll classes. Lastly, this project contributes to the development of the Machine Learning approach in Malaysia. We observed one of the current real Malaysia problems and were able to implement it with the computational science area that allows the enhancement of features in the toll system using an image classification for multiple Malaysia vehicle classes.

This paper is organized as follows: Section 2, Literature Reviews, discusses similar and previous papers as well as re-highlights the concepts of Machine Learning used in this paper. Section 3 Methodology presents the overall method adopted and implemented in order to execute this project. Section 4, Project Development, elaborates on the approaches from the beginning of the development process until model findings and enhancements for future projects.

II. Literature Reviews

RFID technology has been widely adopted in toll collection systems worldwide, including in Malaysia. RFID (Radio Frequency Identification) technology is designed to make toll collection more efficient and faster. It allows vehicles to pass through toll gates without stopping to make a payment manually. Instead, the toll fee is automatically deducted from the driver's prepaid RFID account. However, there have been reports of congestion at some toll gates in Malaysia due to the implementation of RFID technology. One of the main reasons for this congestion is the need for proper infrastructure and equipment to handle the increased traffic flow caused by RFID. Additionally, there have been issues with the reliability of the RFID system, such as faulty RFID readers and insufficient funds in the drivers' RFID accounts. These issues can lead to delays and frustration for drivers, further exacerbating traffic congestion. RFID technology is designed to read and collect information from RFID tags or stickers attached to the vehicle, even when moving at high speed. However, certain factors can affect the readability of the RFID sticker, such as the distance between the reader and the sticker and the angle at which the sticker is positioned. In some cases, if the RFID reader is not calibrated correctly or positioned, it may not be able to read the RFID sticker when the vehicle moves quickly. Hence, it can cause delays and inconvenience for drivers, especially during peak hours when heavy traffic is heavy. RFID systems depend highly on technology and can be vulnerable to disruptions and malfunctions. For example, if a vehicle's tag is damaged, lost, or stolen, the system may be unable to detect it. In addition, the RFID signals can be interfered with by other electronic devices or materials, which can cause inaccuracies in data collection. Additionally, one of the primary disadvantages of RFID technology is the cost of implementing and maintaining the system. It includes the cost of purchasing and installing the RFID readers, as well as the cost of replacing and upgrading the technology as it becomes outdated. Furthermore, RFID systems require regular maintenance to ensure they function correctly. It can include cleaning and repairing the equipment and updating software and firmware to keep the system up to date with the latest security protocols and features [2].

One of the computer vision techniques is object detection. It requires identifying and localizing images or videos by tracking the object's location. The procedure initiates by directly extracting image characteristics from the unaltered image. These features are then systematically relayed through successive layers, enabling the accumulation of intricate high-dimensional insights inherent to the image. This accomplishment stands as a significant triumph within the realm of computer vision. Efficient vehicle image classification is imperative, demanding both rapidity and a commendably elevated level of accuracy. This urgency arises from the need to discern the vehicle's category while it is moving along roadways. Simultaneously, it facilitates the identification of numerous classifiable vehicles and road-related entities within a single scene, allocating accurate labels to the bounding boxes encompassing these entities. Noteworthy contemporary models for target detection encompass R-CNN, SPP-Net, Fast-RCNN, Faster-RCNN, SSD, YOLO, and ResNet [3]. You Only Look Once (YOLO) is one of the best algorithms to pass the real-time object to a fully connected neural network and is recognized as a regression problem. YOLO is a one-stage detector approach based on deep neural networks without a specific region proposal step that uses the whole picture as the network's input and goes forward from image pixels to bounding box coordinates and class probabilities [4]. A single instance of

forward propagation through the network allows this algorithm to generate predictions, producing outcomes that encompass identified objects, corresponding confidence level scores, and bounding box specifications.

Several studies discuss comparing Faster-RCNN, YOLO, and SSD for real-time vehicle type recognition [5]. Some works elaborated on the real-time detection of traffic participants using the YOLO algorithm [6]. The multitask loss function is used to ze and realize single-stage object detection, allowing all network layers to be updated in the model training without needing disc storage to cache the features [7]. To compare Faster-RCNN, YOLO, and SSD, the author trained 1447 vehicle image datasets named car, mini_van, big_van, mini_truck, truck, and compact for each model. YOLO v4 was used to improve the performance with a more prominent solution. The data's Region of Interest (ROI) was extracted from the vehicle's front window to the bumper to keep the features. As a result, the accuracy of CNN is relatively high, but the speed is significantly slower than other models. Despite its faster processing speed compared to YOLO and CNN, the SSD model exhibits diminished accuracy due to its reliance on the lightweight MobileNet architecture. This showed that it sometimes failed to recognize a vehicle, while YOLO had a low accuracy but better precise value for the number of vehicles that can be detected in each frame of video [5]. As a result, the studies indicate that the YOLO approach is the best model among the trained object detection models.

Several previous works have implemented the Faster R-CNN algorithm for vehicle image classification, demonstrating its effectiveness in addressing similar challenges to the proposed project. These works provide valuable insights into the strengths and weaknesses of applying Faster R-CNN in vehicle classification. One recent research study by [8] focused on vehicle classification using Faster R-CNN and transfer learning techniques. The study utilized a large-scale dataset of vehicle images and trained the Faster R-CNN model with a pre-trained convolutional neural network (CNN). The results showed high accuracy in classifying various vehicle types, indicating the effectiveness of the Faster R-CNN algorithm in this domain. However, one limitation of the study was the reliance on a pre-trained CNN, which may affect the model's adaptability to new or unseen vehicle classes. Another relevant work by Wen Li [9] explored the application of Faster R-CNN for vehicle detection and classification in urban traffic scenarios. The study employed a region proposal network (RPN) to generate candidate regions and utilized the Faster R-CNN framework for classification. The results demonstrated the robustness of the approach in accurately detecting and classifying vehicles under challenging conditions, such as occlusions and varying scales. However, the study acknowledged that the computational requirements of Faster R-CNN could be demanding, particularly for real-time applications, which may limit its practical implementation in resource-constrained environments. In a separate research endeavor, Jiani Xi [10] investigated using Faster R-CNN for vehicle attribute recognition. The study aimed to classify vehicles based on color, type, and brand attributes. The Faster R-CNN model was trained on a diverse dataset comprising various vehicle attributes. The findings highlighted the model's ability to accurately recognize vehicle attributes, providing valuable information for traffic surveillance and law enforcement applications. However, the study acknowledged that occlusions and variations in lighting conditions can still pose challenges to the accuracy of attribute recognition.

In conclusion, the reviewed works demonstrate that Faster R-CNN remains relevant and effective for vehicle image classification tasks. It offers high accuracy in detecting and classifying vehicles, even in complex urban traffic scenarios. However, limitations such as reliance on pre-trained models, computational requirements, and challenges in handling occlusions and variations in lighting conditions should be considered in implementing Faster R-CNN for real-world applications. Further research and development efforts are needed to enhance the algorithm's adaptability, efficiency, and robustness to ensure its practical viability in automated toll-collection systems.

The realm of computer vision has transformed by integrating machine learning techniques, ushering in an era of practical and precise object detection and classification. These techniques have been extensively studied and applied in various domains, including toll-collection systems [11]. By leveraging machine learning algorithms, toll collection systems can accurately classify and differentiate various vehicle classes based on their attributes, improving efficiency and accuracy in toll collection processes. The reviewed literature highlights the effectiveness of object detection algorithms such as YOLO (You Only Look Once) and Faster R-CNN in the context of toll collection systems. These algorithms have been chosen over other models due to their unique advantages and proven performance. For instance, YOLO stands out in its ability to achieve real-time processing through its approach of treating object detection as a regression challenge. It directly predicts bounding box coordinates and class probabilities, eliminating the need for a separate region proposal step. This approach improves speed and efficiency, making it highly suitable for real-time vehicle classification in toll-collection scenarios.

On the other hand, Faster R-CNN is a two-stage detector known for its accuracy and robustness. It utilizes a region proposal network (RPN) to generate candidate regions, which are then classified and refined. Despite its computational demands, Faster R-CNN demonstrates high accuracy in detecting and classifying vehicles, even in complex urban traffic scenarios. Its ability to handle occlusions and variations in lighting conditions makes it a reliable choice for accurate vehicle classification [12].

The extensive research and successful application of YOLO and Faster R-CNN in automated toll collection systems reinforce their effectiveness and establish them as preferred choices in the reviewed literature. These algorithms offer real-time performance, accuracy, and robustness, making them well-suited for the challenges and requirements of toll collection systems. By leveraging the capabilities of these machine learning algorithms, toll collection systems can achieve efficient, accurate, and real-time vehicle classification, leading to improved congestion management, reduced human errors, and enhanced overall system performance. Moreover, the continued relevance of YOLO and Faster R-CNN in computer vision and object detection is evident through ongoing research and projects utilizing these models. Researchers and practitioners continue to explore and refine these algorithms, incorporating advancements and improvements to address specific challenges in toll collection systems and other related domains. The consistent utilization of YOLO and Faster R-CNN in recent studies indicates their

effectiveness and reliability, reinforcing their position as preferred choices for vehicle classification and object detection tasks [12].

Despite implementing RFID technology in Malaysia's toll collection systems, these problems have affected its effectiveness in reducing congestion and improving the overall toll collection process. Therefore, developing an automated image-based vehicle classification system using YOLO and Faster R-CNN, such as the one proposed, can provide an alternative solution to address these issues and ensure accurate toll rates, smoother traffic flow, and reduced operational costs. Overall, this system ensures that all lanes will be available for all vehicles except motors while expecting that the RFID system will be improved in its effectiveness; if RFID is not possible, any advanced system will suffice.

III. Project Schedules

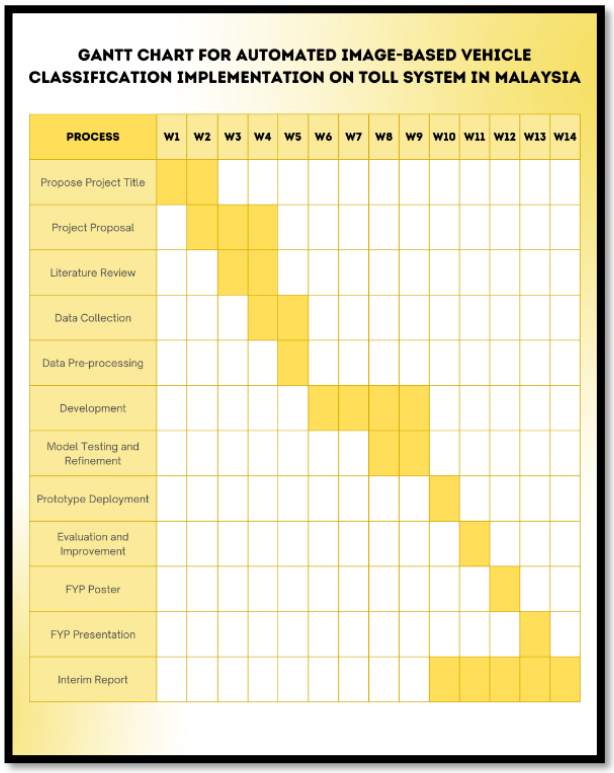

Fig. 1. Phase 1 Gannt Chart.

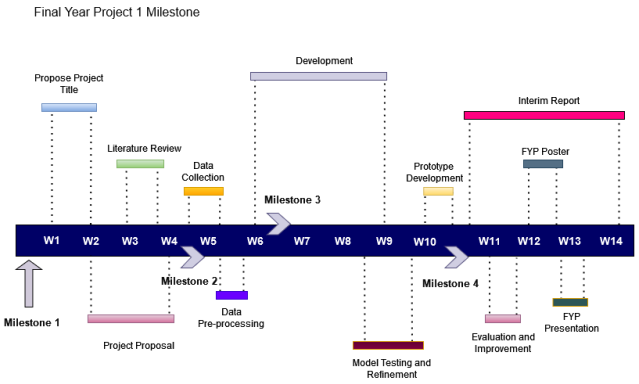

Fig. 2. Phase 1 Milestone.

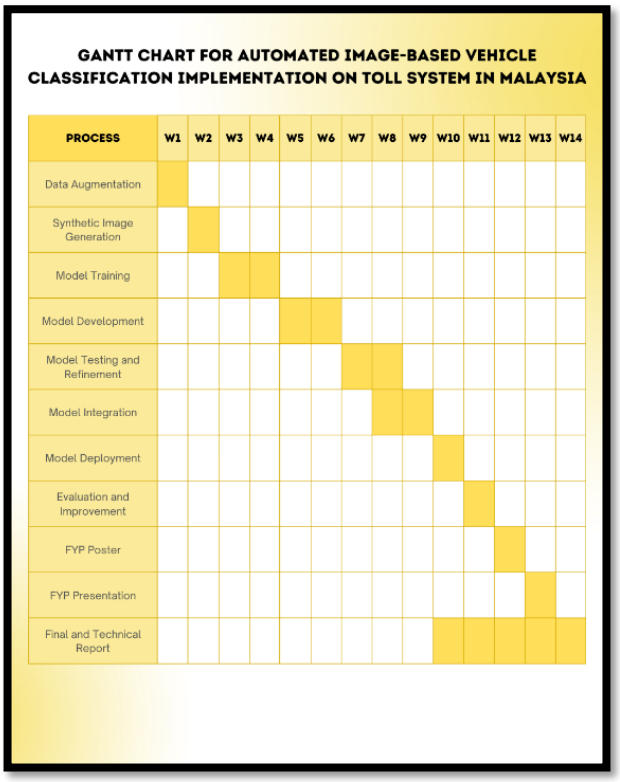

Fig. 3. Phase 2 Gannt Chart.

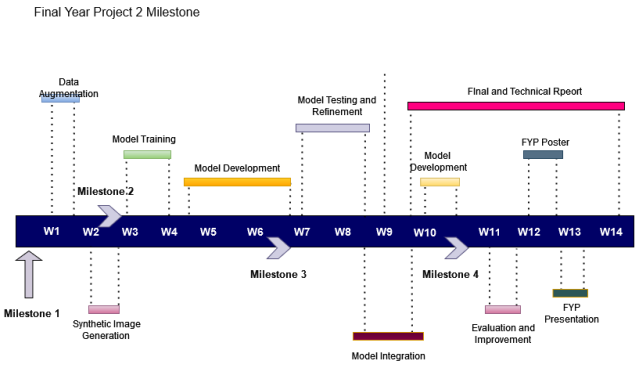

Fig. 4. Phase 2 Milestone.

IV. METHODOLOGY

A. Implementation

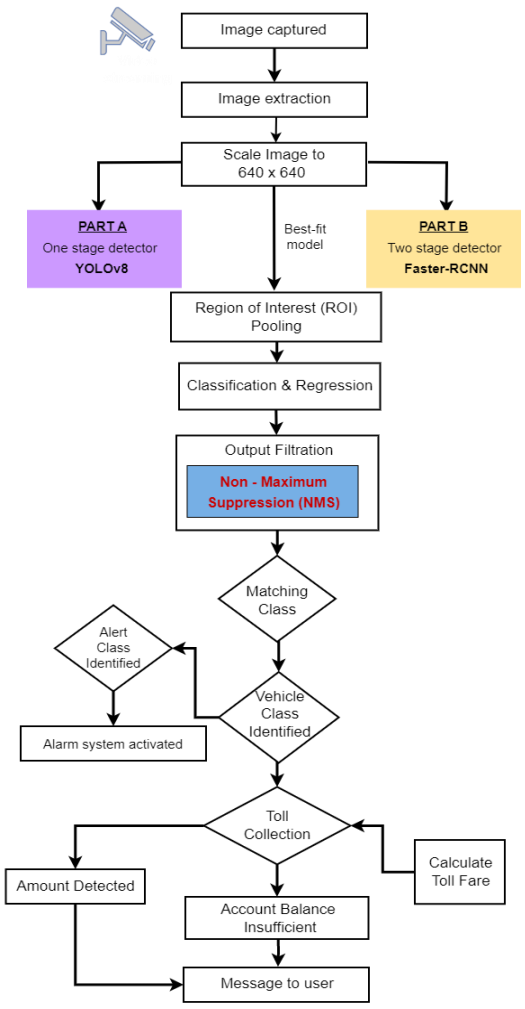

Fig. 5. Flowchart Implementation Machine Learning Approach.

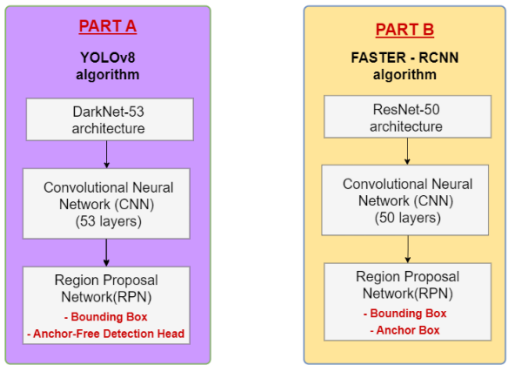

Fig. 6. Algorithms Approaches.

B. Data Collection

Malaysia vehicle datasets were collected manually from various trusted website online platforms such as mytruck, WapCar, Flickr, Dreamstime, BusOnlineTicket, redbus, caricarz, mytruck, and carousell. We also obtained some images from social media such as Facebook and Twitter. Triangulation techniques were used to gather images of vehicles from multiple sources and collect images from various angles. We ensure all the images are in the same digital format, jpg. Different image formats will be converted using an online image converter before being categorized into distinct types of vehicle classes. All saved images with the same format are divided into classes accordingly to ensure the total number of images for each class is balanced. One thousand four hundred vehicle images have been collected and uploaded to Google Drive for easy access.

C. Software, Features, and Packages Required

The following table shows the functions, features, and packages utilized from selected software.

TABLE I. SOFTWARE, FEATURES AND PACKAGES UTILIZED

Software/Features/ Packages	Functions
OpenCV	Library of programming functions mainly for real-time computer vision
ultralytics	Ultralytics YOLOV8 for SOTA object detection, multi-object tracking, instance segmentation, pose estimation and image classification.
Roboflow	Software to manage image data, annotate and label datasets, apply preprocessing and augmentations, convert annotation file formats
LabelImg	A graphical image annotation tool and label object bounding boxes in images.
Pycharm	An integrated development environment used for programming in Python
Google Colab	The cloud-based development environment is used for running the Python code.
Jupyter Notebook	Used to split image folders and rescale images.
TensorFlow 2.0	The deep learning framework is used for importing, utilizing the pre-trained model, training, and inference. Also, using TensorFlow Object Detection API.
Pandas	A Python library is used for data manipulation and analysis.
PIL (Python Imaging Library)	A Python library is used for image processing and manipulation.
Matplotlib	A plotting library is used for visualizing images and results.
NumPy	A Python library is used for numerical operations and array manipulation.
Visual Studio Code	Create yaml file for Yolov8 configure files.

D. Develop Algorithm

In object detection models, a dichotomy exists between single-stage and two-stage variants. Single-shot detectors, also called one-stage models, are designed to identify objects in a single traversal of the input image. These models employ a singular neural network to predict object positions and class designations simultaneously. In contrast, two-stage object detection models follow a bipartite process. The initial phase employs a distinct network, known as a Region Proposal Network (RPN), to generate a set of potential object locations, termed region proposals. Subsequently, the second stage engages another network to classify these regional proposals and refine their positional accuracy. Consequently, this project introduces two distinct models: a one-stage object detection model represented by YOLO and a two-stage object detection model exemplified by Faster R-CNN.

a) YOLOv8 (2023)

The algorithm used uses a convolutional neural network (CNN) approach to analyze an entire picture in a single forward pass. YOLO's primary feature is its single-stage detection, which is meant to find objects in real-time with high accuracy and performance. YOLO processes differ from two-stage detection models such as RCNN because it processes the entire picture in a single pass, making it quicker and more efficient than two-stage detectors that need to indicate regions of interest and classify the object's region. Regarding network design, new features, and applications, Yolov8 is the most recent version of YOLO. Since it does not deal with complicated pipelines, YOLO is highly efficient in speed, as it has a processing speed of 45 frames per second (FPS) [14].

This model comes equipped with several different pre-trained models: instance segmentation, image classification, and object identification. The annotation format employed by YOLOv8 corresponds to the YOLOv5 PyTorch TXT annotation format, which itself is a variation of the Darknet annotation format. The process of YOLOv8 is the most recent version of the YOLO object detection model, intending to improve accuracy and efficiency over prior versions. Key improvements encompass an enhanced and finely tuned network architecture, a revised design for anchor boxes, and a modified loss function tailored to greater accuracy [13]. As our project performs custom datasets to specific Malaysia vehicles only, YOLOv8 will fine-tune custom datasets to boost their accuracy for specific object identification applications.

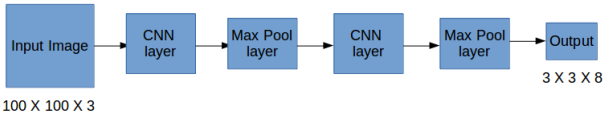

Fig. 7. YOLOv8 layers.

The original YOLO design incorporates as many as 24 convolutional layers, succeeded by two fully connected layers. In contrast, YOLOv8 introduces Darknet-53 as a novel backbone network, which substantially enhances both speed and accuracy compared to the preceding backbone of YOLOv7. DarkNet-53 is a convolutional neural network consisting of 53 layers and can classify images across a spectrum of 1000 distinct object categories. This model also adopts an anchor-free detection methodology, wherein the object detection prediction involves determining the object's centre rather than referencing an offset from a predefined anchor box, as explained by Encord [13]. Previously, anchor boxes were predefined rectangles with specific dimensions to identify object classes with suitable scales and aspect ratios. The innovation in YOLOv8 lies in its elimination of manual anchor box specification, a feature that augments flexibility and cost-effectiveness. As seen in prior YOLO versions like v1 and v2, manual anchor selection often posed challenges leading to suboptimal outcomes.

Furthermore, the network generates several attributes, including background, IoU, and offsets, for each gridded box. These attributes contribute to the adaptation of anchor boxes. While traditional anchor strategies entail numerous predefined starting points for bounding box predictions, YOLOv8 departs from this approach, resulting in fewer predictions per image. This strategic shift optimizes inference time without compromising accuracy.

The operational flow of this algorithm commences with the prediction of numerous bounding boxes per grid cell; however, solely those bounding boxes exhibiting the highest Intersection Over Union (IOU) with the ground truth are retained, a procedure commonly referred to as Non-Maxima Suppression (NMS). In its initial steps, the YOLO algorithm extracts an individual image from the video stream to serve as input. What sets YOLO apart is its methodology of segmenting images into grid cells. Each image input is divided into a grid of dimensions S x S, with every grid cell making predictions for three distinct bounding boxes [14]. Distinguishing itself further, the algorithm executes a singular forward propagation pass through the network to formulate predictions and categorize either an entire image or an object presented within the image into one of its predefined classes or categories.

After applying non-maximum suppression, the system generates identified entities alongside corresponding bounding boxes. The non-maximum suppression mechanism entails comparing the bounding box with the highest probability score and all other bounding boxes, assessing their intersection sequentially. Those classes whose Intersection over Union (IoU) value exceeds 0.5 are excluded from consideration. IoU calculates the overlap region between predicted and actual bounding boxes, assigning a quantitative score to evaluate the extent of alignment between predicted bounding boxes and ground truth. Predicted bounding boxes that exhibit substantial overlap with the actual objects are awarded a higher score. An intersection score surpassing 0.5 signifies a robust prediction. YOLO employs a singular regression module to define the characteristics of these bounding boxes. The final vector representation for each bounding box is denoted as Y = [pc, bx, by, bh, bw, c1, c2] in the format described [14].

b) Faster R-CNN ResNet50

Faster R-CNN stands as a renowned architectural solution for object detection. This cutting-edge algorithm, Faster R-CNN, ingeniously merges the deep convolutional neural networks with the two-stage framework for object detection. The initial stage involves a region proposal network (RPN) that generates a collection of potential object regions. Subsequently, these regions enter the second stage, encompassing a Fast RCNN network responsible for assigning class labels to the regions and enhancing the precision of bounding box positioning. These two stages collaborate to recognize objects within an image proficiently by suggesting numerous prospective regions. These regions are then meticulously classified and precisely located using the Fast RCNN network. Notably, the critical strength of Faster RCNN lies in its end-to-end training approach. This strategy empowers the algorithm to learn object detection directly from image data without being dependent on subjective rules or manually crafted characteristics [15]. The proposed model for Faster RCNN, employing ResNet50 for the identification and categorization of vehicle objects, can be segmented into the subsequent modules:

ResNet50 Backbone: Within this component lies the ResNet50 network, which is pre-trained and serves as the feature extractor. It generates a corresponding feature map as output by operating on an image input. In the context of Faster RCNN, the ResNet-50 backbone assumes the feature extraction network's role, condensing the input image's essential attributes. Subsequently, this compact representation journeys through the convolutional neural network, constituting the backbone. This process entails resizing the input image and preserving the aspect ratio by capping the longer side at 1000 pixels while adjusting the shorter side proportionally. This manipulation leads to the creation of the feature, as mentioned earlier, map through the backbone network.

Consequently, these feature maps are harnessed to fuel the Fast RCNN network, facilitating tasks like classification and fine-tuning bounding box coordinates. Notably, Faster RCNN leverages the insights gleaned from extensive picture classification endeavors by leveraging a pre-trained ResNet50 network as its feature extraction foundation. This strategic integration significantly bolsters its prowess in object detection tasks [15].

Furthermore, utilizing ResNet50 as a backbone presents an opportunity for transfer learning. This practice permits fine-tuning the feature extractor to cater to the specific demands of the object detection task, even with a more limited dataset. The ResNet architecture brought about a breakthrough by introducing the concept of Residual Networks, primarily aimed at mitigating the challenge of exploding gradients. This predicament is effectively addressed through the implementation of a technique referred to as a skip connection. Renowned for its adeptness with skip connections, the ResNet architecture incorporates these shortcut links to combat the issue of gradients vanishing within profoundly deep neural networks. The mechanics of these skip connections within ResNet involve allowing the network to circumvent one or multiple layers, thereby facilitating the direct backpropagation of gradients to preceding layers, as illustrated in the diagram. This strategic approach safeguards the information inherent to the original input, thus enhancing the network's capacity to comprehend and enhance its performance. The network's learning process is enriched and refined by upholding the integrity of the original input information [15].

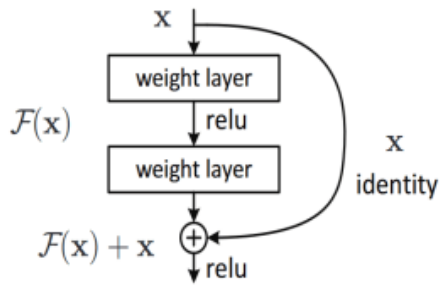

Fig. 8. FRCNN layers.

In the absence of employing skip connections, the input 'x' undergoes multiplication with the layer's weights, succeeded by the addition of a bias term:

$$H(x) = f(wx + b) \quad (1)$$

or

$$H(x) = f(x) \quad (2)$$

With the integration of the skip connection mechanism, the layer's output transforms to

$$H(x) = f(x) + x \quad (3)$$

Like most deep learning architectures, ResNet50 employs a categorical cross-entropy loss function. This specific loss function is commonly utilized when dealing with multiclass classification challenges. Its primary role is to measure the disparity between the predicted probability distribution of classes and the actual class label. To illustrate, here is an equation delineating calculating the categorical cross-entropy loss.

$$Cross - entrophy = -\frac{1}{N}\sum_{i=1}^{N} logP_{model}[y_i \in C_{y_i}] \quad (4)$$

Region Proposal Network (RPN): Plays a pivotal role within the Faster RCNN object detection framework. Its primary responsibility revolves around formulating a collection of region proposals sourced from the candidate object regions present in the input image. The central outcome of a regional proposal network is the generation of multiple proposals, each encapsulating a distinct region. These generated proposals subsequently undergo detection via the identification network. The Region Proposal Network (RPN) encompasses three fundamental components: the anchor window, the loss function, and the set of region proposals. To execute its operations, the RPN adopts a sliding window approach. This technique involves subjecting a compact sub-network to an exhaustive 3x3 sliding window traversal. Consequently, the RPN adeptly utilizes the Intersection over Union (IoU) ratios and the ground-truth bounding boxes to create an extensive array of anchors, optimizing the proposal process.

The Region Proposal Network (RPN) employs anchor boxes, predefined shapes for bounding boxes, to steer the process of generating region proposals. Subsequently, the outcomes from the network are combined with these anchor boxes to yield the ultimate collection of regional proposals. The sequence of actions unfolds as outlined below: (i) The process of RPN involves sliding a window across the feature map for each individual region. (ii) At each location, k (where k=9) anchor boxes are employed. These anchors possess scales of 128, 256, and 512, along with aspect ratios of 1:1, 1:2, and 2:1, effectively constituting the foundation for generating region proposals. (iii) The CLS layer is responsible for generating 2k scores corresponding to k boxes, irrespective of whether an object is detected or not. (iv) Conversely, the reg layer contributes by producing 4k values that denote the center coordinates, width, and height of the k boxes. (v) In totality, the count of anchors sums up to WHk, correlating with the dimensions of the WH feature map.

The multitask loss function computes the RPN's total loss. The formula for calculation is:

$$\frac{1}{N_{cls}}\sum_i L_{cls}\,(p_i, p_i^*) + L(\{p_i\},\{t_i\}) = \lambda\frac{1}{N_{reg}}\sum_i p_i^*\, L_{reg}(t_i, t_1^*) \quad (5)$$

Here, let N_{cls} Denote the count of training data within the batch, N_{reg} Signify the count of anchors, λ stands for the balancing weight. The notation $L_{cls}(p_i, p_i^*)$ Pertains to the logarithmic loss function, characterized as follows:

$$L_{cls}(p_i, p_i^*) = -\log\,[p_i^* p_i + (1 - p_i^*)(1 - p_i)] \quad (6)$$

The regression loss, denoted as $p_i^* L_{reg}(t_i, t_i^*)$, is evaluated through the application of the subsequent Smooth L1 function:

$$L_{reg}(t_i, t_i^*) = \begin{cases} 0.5\,(t_i - t_i^*)^2, & |x| < 1 \\ |t_i - t_i^*| - 0.5, & otherwise \end{cases} \quad (7)$$

Here, p_i denotes the likelihood assigned to the anchor being identified as the intended target, and p_i^* is the truth value of the prediction outcome: if the anchor is predicted as a positive sample, the value of tag p_i^* is 1; otherwise, the value is 0; $t_i = \{t_x, t_y, t_w, t_h\}$ is the location of the predicted detection box; and t_i^* is the ground truth coordinate.

As a result, the RPN network must determine which location contains the object ahead of time. Subsequently, the detection network incorporates the provided coordinates and bounding boxes to execute object class recognition and furnish the corresponding object bounding box (Ren et al., 2015) [15].

RoI Pooling Layer: The RoI (Region of Interest) pooling mechanism is a pivotal component within Faster RCNN, meticulously handling the region proposals that emerge from the preliminary RPN phase. Nestled within the Fast RCNN network, the RoI pooling layer interfaces with two primary inputs: the feature map engendered by the ResNet-50 backbone and a collection of region proposals. Its core functionality revolves around resizing each proposed region to a standardized dimension, irrespective of the original size or aspect ratio. Subsequently, this layer amalgamates the features within each designated region into a concise feature portrayal. This strategic maneuver equips the Fast RCNN network to engage in object classification and regression, emancipating from object dimensions or aspect ratio constraints. The essence of the RoI pooling process lies in its pivotal role, enabling the Faster RCNN architecture to effectively discern and categorize objects possessing varying scales and aspect ratios within a given image. Furthermore, the RoI pooling layer bestows the Fast RCNN network with a consistent input size, streamlining the training and optimization process. This attribute facilitates adept handling of diverse object sizes within the image, rendering the network more manageable in practice while retaining its capacity to accommodate dimensional variability [15].

Fast RCNN Classifier and Bounding Box Regressor: Following the RoI pooling stage, the features extracted from the proposed regions are channeled into the classifier and bounding box regressor components within the Faster RCNN architecture. The fully connected layer classifier undertakes the object classification task by making predictions regarding the likelihood of each region proposal being associated with distinct object classes. In essence, the classifier computes a score for each combination of region proposal and class, signifying the probability of the presence of an object from that specific class within the region. On the other hand, the bounding box regressor, also implemented as a fully connected layer, specializes in refining bounding box positions. It takes the feature representation extracted from the region proposals as its input and subsequently generates outputs that denote adjustments to the positions of these region proposals. These adjustments enhance the proposals' alignment with the actual objects depicted in the image. The Fast RCNN network is comprised of the classifier and bounding box regressor components. This amalgamation effectively identifies and categorizes objects within images, leveraging a fusion of insights from region proposals, classifier scores, and refined bounding box coordinates.

In this project, we implement Faster R-CNN to replicate a specific general detector within this undertaking. Our approach entails harnessing object proposals trained through an RPN (Region Proposal Network) and the corresponding features derived from a ResNet50 CNN architecture. This amalgamation furnishes an effective strategy for identifying and categorizing various vehicle classes. By harmoniously integrating RPN and Fast R-CNN, we consolidate their convolutional capabilities by utilizing the contemporary neural network framework. The blueprint of our proposed methodology encompasses three profound networks: the feature network, RPN, and detection network. The Faster R-CNN methodology adopts a bounding box strategy that empowers the operator to define potential regions for submission to the RPN. Employing the devised technique, we set a CNN model into motion on our vehicle dataset. Following a meticulous analysis of the input image, a selective search procedure comes into play to pinpoint a region of interest (RoI). This delineated RoI is subjected to a refinement process, classifying candidates within the nearest raster frames. This fine-tuning is done by leveraging the intricate model generated through deep learning methodologies [15].

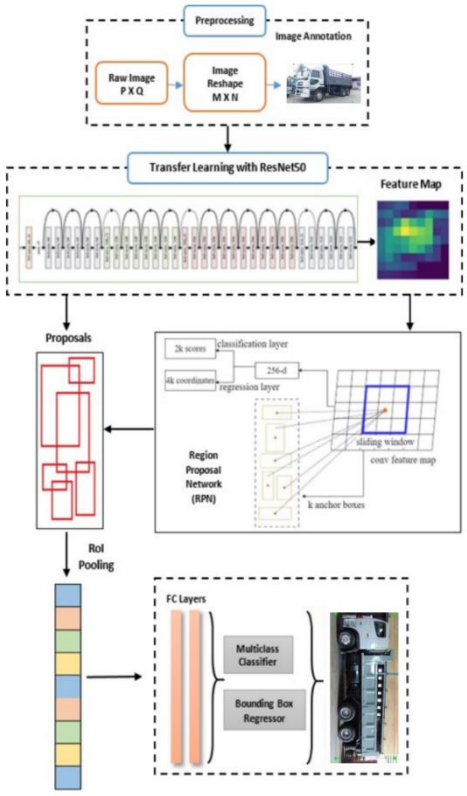

Fig. 9. Region of Interest (ROI) layers.

E. Performance Evaluation

Several metrics, such as Mean Average Precision (mAP), precision, F1 score, and recall, can be used to evaluate the performance of a vehicle detection and classification model. These metrics provide insights into the model's performance and help assess its effectiveness in detecting and classifying vehicles accurately [16].

V. PRECISION

$$Precision = \frac{TP}{TP+FP} \times 100 \quad (8)$$

Precision quantifies the fraction of accurately anticipated positive occurrences (in this case, vehicles) within the entirety of instances projected as positive by the model. Specifically concerning the realm of vehicle detection, precision serves as an indicator of the model's effectiveness in accurately recognizing vehicles amidst the identified objects. A heightened precision score signifies the model's capability to minimize false positives, denoting infrequent misclassification of non-vehicle entities as vehicles.

VI. RECALL

$$Recall = \frac{TP}{TP+FN} \times 100 \quad (9)$$

The concept of recall, alternatively termed sensitivity or the true positive rate, quantifies the ratio of accurately predicted positive instances (referring to vehicles) relative to the total actual positive instances within the dataset. In the realm of vehicle detection, recall serves as an indicator of the model's efficacy in encompassing all the vehicles existing within the images. A substantial recall value implies that the model exhibits minimal false negatives, signifying infrequent instances where it overlooks or neglects to identify vehicles.

VII. F1-SCORE

$$F1\ score = \frac{2 \times Precision \times Recall}{Precision+Recall} \quad (10)$$

The F1-score finds its essence in being the harmonic average of precision and recall, culminating in a well-rounded gauge of the model's efficacy, encompassing both erroneous positives and negatives. Within vehicle detection, the F1-score mirrors the model's proficiency in precisely identifying vehicles while mitigating the instances of mistaken positives and negatives. Elevating the F1-score signifies an augmented equilibrium between precision and recall, increasing accuracy in detection and classification endeavors.

VIII. MEAN AVERAGE PRECISION (MAP)

$$Average\ Precision\ (AP) = \sum_{k=0}^{k=n-1} [Recall(k) - Recall(k+1)] * Precision(k) \quad (11)$$

$$Mean\ Average\ Precision\ (mAP) = \frac{1}{n} \sum_{k=1}^{k=n} AP_k \quad (12)$$

mAP is a commonly used metric in object detection tasks. It evaluates the overall mean Average Precision (mAP), which is a prevalent metric within object detection undertakings. Its role encompasses assessing a model's efficacy comprehensively, factoring in precision and recall at varying confidence thresholds. The mAP computation involves arranging the model's output following the assigned confidence scores for each detected object. This arrangement facilitates the creation of a precision-recall curve, the enclosed area of which signifies the mAP value. By encapsulating precision and recall dynamics across multiple confidence levels, mAP offers an inclusive gauge of the model's competence in diligently recognizing objects.

A. Testing Prototypes

We propose to evaluate the performance of the prototype real-time moving vehicle toll collecting system using a database, Microsoft SQL Server Management Studio, linked to the program through ODBC Drivers. The database will include the vehicle toll classes, owner's vehicle class detected, the toll fare, and the owner's remaining wallet balance. The prototypes are as follows :

Fig. 10. Situation 1: Toll users of Class 1 that have a sufficient credit balance.

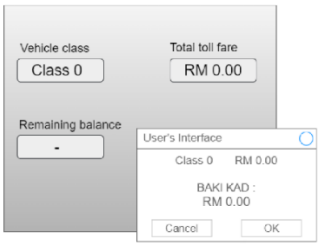

Fig. 11. Situation 2: Toll user of Class 0 with zero toll fare deduction.

Fig. 12. Situation 3: Toll users of Class 4 that have insufficient credit balance.

IX. Project Development

A. *Data Pre-processing*

X. Images Rescaling

In this process, a code snippet was implemented to rescale a collection of vehicle images stored in a specified folder. This process aimed to standardize the images to a specific target size while preserving their aspect ratio. In machine learning, it is essential to maintain consistency in the input data to ensure fair comparisons and reliable model performance evaluation. By rescaling all the images in the folder to the same target size, any image dimensions or aspect ratio variations are standardized, providing a consistent input for the machine learning model. The code utilized the OpenCV library in Python to perform image manipulation tasks. The rescaling procedure involved iterating through each image in the folder, loading it using OpenCV, and applying the resizing transformation to match the target dimensions. A black canvas of the target size was created to ensure the rescaled images were centred, and the resized image was pasted onto it with the appropriate padding. The original images vary in pixels. In this project, a target size of (800, 600) was chosen as it is commonly used for some object detection models.

```python
# Define the target size
target_size = (800, 600)

def rescale_image(image, target_size):
    # Get the original dimensions of the image
    (h, w) = image.shape[:2]

    # Resize the image to the target size while maintaining the aspect ratio
    (target_w, target_h) = target_size
    ratio = min(target_w / w, target_h / h)
    new_dimensions = (int(w * ratio), int(h * ratio))
    resized_image = cv2.resize(image, new_dimensions, interpolation=cv2.INTER_AREA)

    # Create a black canvas of the target size
    canvas = np.zeros((target_h, target_w, 3), dtype=np.uint8)

    # Compute the padding needed to center the resized image on the canvas
    pad_x = (target_w - new_dimensions[0]) // 2
    pad_y = (target_h - new_dimensions[1]) // 2

    # Paste the resized image onto the canvas
    canvas[pad_y:pad_y + new_dimensions[1], pad_x:pad_x + new_dimensions[0]] = resized_image

    return canvas
```

Fig. 13. Rescale Image user-defined function

XI. Splitting Folders for Train, Test, and Validation Dataset

The dataset is split into train, test, and validation phases where in each phase, there are seven classes of image, which are Alert, Class 0, Class 1, Class 2, Class 3, Class 4, and Class 5. The splitting of folders is being done on the rescaled images. The split folders library was employed to split the input images. The split was performed using a specified ratio of 70% for training, 10% for validation, and 20% for testing. The process ensured consistent and standardized data division for subsequent stages of the project, such as model training, validation, and testing.

```python
import splitfolders  # or import split_folders

input_folder = 'split/'

# Split with a ratio.

# Train, val, test
splitfolders.ratio(input_folder, output="First Split",
                   seed=42, ratio=(.7, .1, .2),
                   group_prefix=None) # default values
```

Fig. 14. Data Splitting Folders.

XII. Images Labeling

The vehicle image dataset acquired was unlabeled, so it had to be labeled manually. To annotate the dataset for two different object detection models, YOLO and Faster R-CNN, the open-source tool called "labelImg" tool was utilized. The tool supports generating annotations in two different formats: YOLO (TXT file) and Faster R-CNN (XML file). The command ! python labelImg/labelImg.py was run to start the "labelImg" tool.

```
! python labelImg/labelImg.py
```

Fig. 15. Label image command

This opened the graphical user interface (GUI) of "labelImg" where images from the dataset could be loaded, and annotations could be created and saved as below:

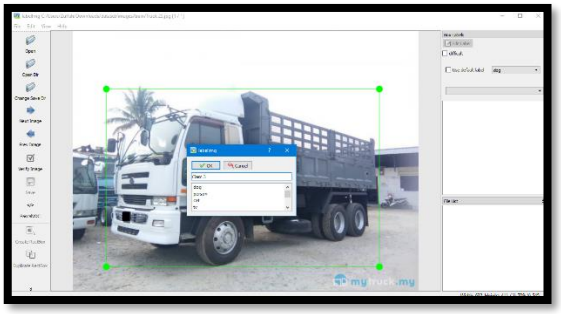

Fig. 16. Labelimg tool interface.

This process facilitated efficient and accurate labeling of the dataset by providing an intuitive graphical interface for annotating objects of interest in the images. The vehicle image dataset is labeled with their corresponding predefined classes: Alert, Class 0, Class 1, Class 2, Class 3, Class 4, or Class 5.

For YOLO annotation, each image in the dataset was loaded into the "labelImg" tool's graphical user interface (GUI). Bounding boxes were manually drawn around the objects of interest in the image, specifying their class labels. The tool then generated a TXT file for each annotated image containing the coordinates and class labels of the bounding boxes. This format is compatible with the YOLO model's training requirements.

For Faster R-CNN annotation, a similar process was followed. The images were loaded into the "labelImg" tool, and bounding boxes were created around the objects. Class labels were assigned to the bounding boxes. However, instead of generating text files, the tool produced XML files following the PASCAL VOC format. These XML files contained the bounding box coordinates, class labels, and additional information required by the Faster R-CNN model.

By using "labelImg" and its support for both YOLO and Faster R-CNN annotation formats, the dataset was effectively labeled to cater to the requirements of both models. This facilitated seamless training and evaluation processes for each model, ensuring accurate and consistent annotations for their respective object detection tasks.

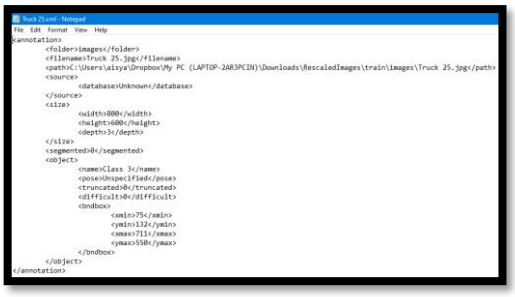

Fig. 17. XML annotation.

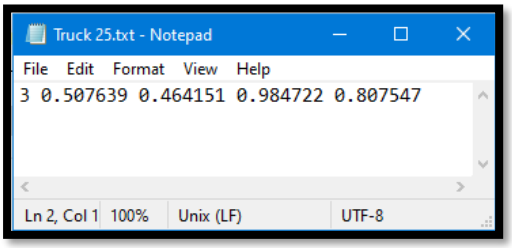

Fig. 18. TXT annotation.

A. *Build Model*

XIII. YOLO

1) Accessing the datasets and importing necessary packages in Google Colab

The Yolo algorithm used an annotation format in the TXT file that is extracted from software named Roboflow. This software provides a workspace for uploading images and other annotation formats that need to be converted into the format that Yolo requires [17]. Firstly, we created a workspace for random images in each folder, train, test, and validation before uploading all the random split images and XML files. The software then converts the image format into a text file. This software also includes some advanced features to ensure the images do not contain null files and duplicate images. Before extracting the text file, we set the format as autorotation to remove any bias and improve performance during training. The images and its text files were kept in Google Drive accordingly, located in path (..content/MyDrive/FYP/Yolo) to access to Google Collab.

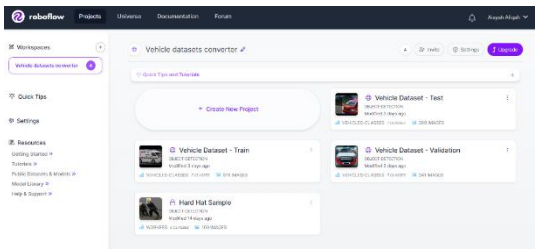

Fig. 19. Roboflow Vehicle Datasets Workspace.

In Google Collab, we changed the runtime type for the hardware accelerator to GPU type T4 to increase the iteration speed. The iteration starts by mounting the drive to access the content of MyDrive. Yolov8 required ultralytics packages to perform multi-image classification and object detection [19]. The YOLO function is imported after ultralytics packages have successfully been executed to continue the prediction and training tasks.

2) Detect object with yolov8l.pt model

We first test the yolo model named yolov8l.pt by performing predict model to the image named vehicles.jpg. The task's objectives are to ensure the Yolo model is located in the workspace and can successfully execute to detect any object in the frame. As a result, the Yolo model detected 3 persons, 12 cars, 2 motorcycles, 2 buses, and 4 trucks in this

image and saved the detection results as predict3 in the runs folder.

Fig. 20. Pre-test image for yolov8l.pt model.

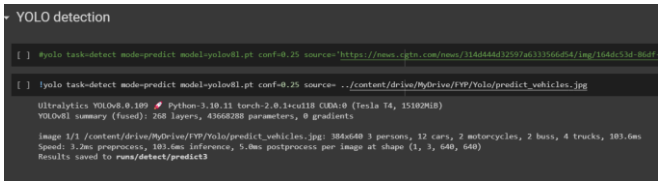

Fig. 21. Result of Pre-test image.

3) Create human-readable data serialization language

A folder named 'data.yaml' was created in Visual Code Studio before training the model. This file indicates where the train and validation images are located, the total number of classes, and the list of predefined classes in a single array. YAML is a popular programming language because it is designed to be easy to read and understand.

4) Custom Data Training

All the images are being extracted from 'data. yaml' path file with the yolo model with 50 epochs and in train mode.

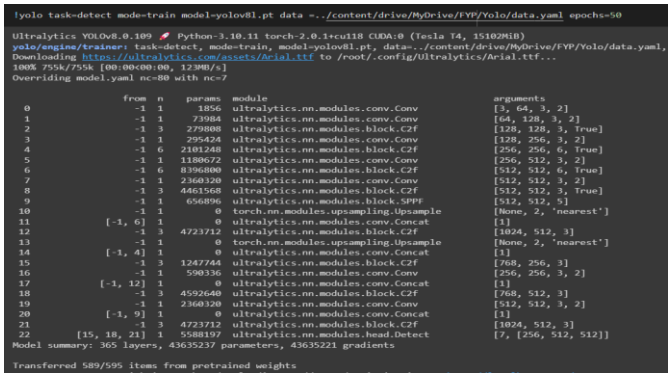

Fig. 22. Execution of Train mode to data.yaml file.

XIV. Faster R-CNN ResNet50

1) Import Libraries

The code begins by importing the required libraries, including os, glob, XML.etree. ElementTree as ET, pandas as pd, and tensorflow as tf, to facilitate subsequent steps in the project. The version of TensorFlow being used is printed to the console, 2.12.0.

2) Clone the TensorFlow Models Git Repository and install the TensorFlow Object Detection API.

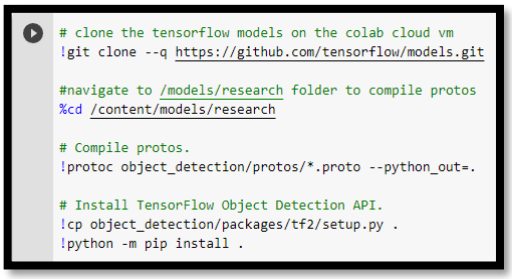

Fig. 23. Cloning and Installing TensorFlow.

The TensorFlow Object Detection API is installed, which is essential for implementing the vehicle detection model.

3) Test the Model Builder

The model builder module is responsible for constructing the object detection model architecture and configuring its components, such as the backbone network, feature extractors, and region proposal networks. By performing this step, we can ensure that the model builder is correctly implemented and that there are no major issues that might hinder the subsequent steps of your project, such as training the model on the vehicle dataset.

4) Create the CSV Files and the Label Map File

This step involves creating CSV files and a label map file.

a) CSV files

Fig. 24. Converting XML format to CSV format.

- In object detection tasks, it is expected to use CSV (Comma-Separated Values) files to store the information about the annotated objects in the images.
- The xml_to_csv function is responsible for converting the XML files containing annotations into CSV format.
- The function iterates over the XML files in the specified train_labels and test_labels directories, extracts relevant information such as image filenames, object classes, and

bounding box coordinates, and stores them in a DataFrame.

- The resulting DataFrame is then saved as a CSV file (train_labels.csv and test_labels.csv) using the to_csv function.
- The CSV files serve as input for the subsequent steps in the object detection pipeline, such as model training and evaluation.

	A	B	C	D	E	F	G	H
1	filename	width	height	class	xmin	ymin	xmax	ymax
2	Ambulan	800	600	Class 0	14	102	788	485
3	Ambulan	800	600	Class 0	42	213	470	511
4	Ambulan	800	600	Class 0	415	240	652	420
5	Ambulan	800	600	Class 0	329	36	624	479
6	Ambulan	800	600	Class 0	197	68	444	312
7	Ambulan	800	600	Alert	177	226	195	264
8	Ambulan	800	600	Class 0	40	48	746	529
9	Ambulan	800	600	Class 0	15	28	780	598
10	Ambulan	800	600	Class 0	50	11	734	592
11	Ambulan	800	600	Class 0	150	60	799	527

Fig. 25. Generated train_labels.csv.

	A	B	C	D	E	F	G	H
1	filename	width	height	class	xmin	ymin	xmax	ymax
2	Ambulan	800	600	Class 0	6	113	791	519
3	Ambulan	800	600	Class 0	11	91	539	401
4	Ambulan	800	600	Class 0	23	121	521	457
5	Ambulan	800	600	Class 0	124	117	736	529
6	Ambulan	800	600	Class 0	1	204	337	428
7	Ambulan	800	600	Class 0	213	203	561	342
8	Ambulan	800	600	Class 0	1	49	750	536
9	Ambulan	800	600	Class 0	415	155	800	547
10	Ambulan	800	600	Class 0	91	164	505	491
11	Ambulan	800	600	Class 0	488	240	570	330

Fig. 26. Generated test_labels.csv.

b) Label Map File

```
label_map_path = os.path.join("label_map.pbtxt")
pbtxt_content = ""

for i, class_name in enumerate(classes):
    pbtxt_content = (
        pbtxt_content
        + "item {{\n    id: {0}\n    name: '{1}'\n}}\n\n".format(i + 1, class_name)
    )
pbtxt_content = pbtxt_content.strip()
with open(label_map_path, "w") as f:
    f.write(pbtxt_content)
    print('Successfully created label_map.pbtxt ')

Successfully converted train_labels xml to csv.
Successfully converted test_labels xml to csv.
Successfully created label_map.pbtxt
```

Fig. 27. Generate Label Map.

- The label map file is used to map the class names to class IDs in the object detection model.
- The xml_to_csv function retrieves the unique class names from the XML annotations and sorts them.
- Then, the script iterates over the class names and assigns a numerical ID to each class.
- The class names and their corresponding IDs are formatted in the Protocol Buffers (protobuf) syntax and stored in the pbtxt_content variable.
- Finally, the pbtxt_content is written to the label map file.
- The label map file is required during model training and inference to map the predicted class IDs to their corresponding human-readable class names.

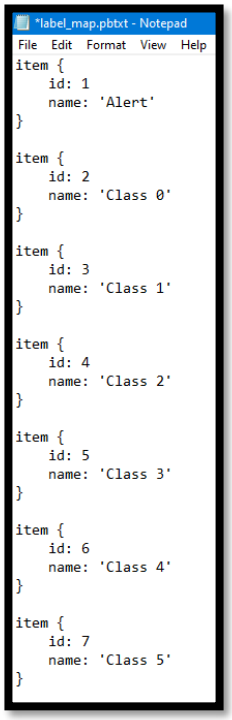

Fig. 28. Label Map Output.

By creating the CSV files and the label map file, it organizes and provides the necessary data and metadata for the subsequent steps in the object detection pipeline. The CSV files contain the annotations, while the label map file provides the mapping between class names and IDs, ensuring proper identification and labeling of objects during training and inference.

5) Create train.record & test.record Files

In this step, the generate_tfrecord.py script is used to create TFRecord files, which are the required data formats for training the object detection model. TFRecord is a binary file format that efficiently stores large amounts of data. The script takes as input the CSV files (train_labels.csv and test_labels.csv) generated in the previous step, along with the label_map.pbtxt file and the path to the image's directory. It then generates TFRecord files for training and testing purposes.

By creating the train.record, and test.record files in the TFRecord format, the data is efficiently stored and ready to be fed into the object detection model for training and evaluation. These files contain the image data, along with their corresponding annotations and class labels, in a format that TensorFlow can process effectively during the training process.

6) Download Pre-trained Model Checkpoint

In this step, a pre-trained model checkpoint is downloaded and extracted. The pre-trained model checkpoint serves as a starting point for transfer learning, where the already trained weights and parameters of the model are used as a foundation

for training on a new dataset, enabling faster convergence and better performance on a specific task.

7) Edit the Model Pipeline Config File

We obtain the model pipeline configuration file, modify it according to the requirements, and save it in the data folder. The model pipeline configuration file contains various settings and parameters that define the architecture and behavior of the object detection model during training. The changes that have been made includes:

- Change num_classes to 7:
  This reflects the number of classes in our vehicle classification task.
- Update paths for test.record, train.record, and labelmap:
  This will update the paths to match the paths of the corresponding files we generated in earlier steps.
- Change fine_tune_checkpoint and fine_tune_checkpoint_type:
  This will update the fine_tune_checkpoint parameter with the path to the directory where we extracted the downloaded checkpoint in a previous step. This should be the directory containing the pre-trained model checkpoint files.
  Set the fine_tune_checkpoint_type to "detection" to indicate that the checkpoint is used for object detection tasks.
- Adjust batch_size and num_steps:
  Modify the batch_size parameter based on the capability of our GPU. In this step, we set it to 1 due to limited GPU memory.
  Update the num_steps parameter to the desired number of training steps we want the object detector to go through. In this step, we set it to 25000 for the training.
- Save the modified model pipeline config file:
  After making the necessary changes, we save the modified model pipeline config file.
- Put the modified config file in the data folder:
  We move and copy the modified model pipeline configuration file to the data folder where we store our project data. This ensures that the training script can access the updated configuration during the training process.

By modifying the model pipeline config file, we customize the behavior of the object detection model according to your specific requirements. The changes we made include adjusting the number of classes, updating file paths, specifying the checkpoint directory, and setting batch size and training steps. These modifications ensure that the model is trained on our dataset and tuned to perform vehicle classification with the desired settings and parameters.

8) Load Tensorboard

```
[ ]  #load tensorboard

     %load_ext tensorboard
     %tensorboard --logdir '/content/gdrive/MyDrive/customTF2/training'
     #%reload_ext tensorboard
```

Fig. 29. Load Tensorboard.

In this step, it will load TensorBoard, a web-based visualization tool provided by TensorFlow, to monitor and analyze the training progress of researchers' object detection model. TensorBoard allows us to visualize various aspects of our model's training, such as loss curves, accuracy metrics, and other valuable statistics. It helps us gain insights into the performance, identify potential issues, and make informed decisions during training.

9) Train the Model

We train our object detection model with the prepared vehicle data using the TensorFlow 2 Object Detection API. This step involves feeding the dataset to the model and iteratively adjusting the model's weights to improve its performance. Running this step allows the model to learn from the labeled vehicle images and improve its ability to classify vehicles accurately. It took about 1 hour to complete the training with the help of GPU memory to speed up the process.

10) Evaluate the Model

We then evaluate the performance of our trained object detection model using the TensorFlow 2 Object Detection API. By running this step, we can evaluate the effectiveness of our trained object detection model and assess its performance metrics, which can help us understand how well it can classify vehicles in unseen data.

11) Export Inference Graph

In this step, we export the trained model as an inference graph, which can be used to predict new images or videos. The export graph will contain all the necessary information and parameters for the trained model. The trained model is exported as saved_model.pb.

12) Test the Trained Object Detection Model on New Data

```
image_path = "/content/gdrive/MyDrive/RescaledImages/VehicleDatasets/test/images/polis 218.jpg"
#print('Running inference for {}... '.format(image_path), end='')

image_np = load_image_into_numpy_array(image_path)

# The input needs to be a tensor, convert it using `tf.convert_to_tensor`.
input_tensor = tf.convert_to_tensor(image_np)
# The model expects a batch of images, so add an axis with `tf.newaxis`.
input_tensor = input_tensor[tf.newaxis, ...]

detections = detect_fn(input_tensor)

# All outputs are batches tensors.
# Convert to numpy arrays, and take index [0] to remove the batch dimension.
# We're only interested in the first num_detections.
num_detections = int(detections.pop('num_detections'))
detections = {key: value[0, :num_detections].numpy()
                 for key, value in detections.items()}
detections['num_detections'] = num_detections

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

image_np_with_detections = image_np.copy()
```

Fig. 30. Model Testing on New Data

In this step, we test the trained object detection model on a new image to perform inference and visualize the detected objects. The code provided loads the saved model, loads the label map, processes an input image, runs the inference, and visualizes the detected objects on the image.

In the figure below, we test the trained object detection model on "polis 218.jpg" and the detected vehicle result is classified as Class 0 with a confidence score of 86%.

Fig. 31. Result of Detection for "polis 218.jpg".

A. Results and Discussion

In this section, we present and analyze the results of two popular object detection models: YOLOv8 and Faster R-CNN, with ResNet50 as the backbone. These models were evaluated on a dataset of vehicle images, aiming to detect various classes of vehicles accurately. The primary focus of this section is to examine and compare the performance of these models based on their evaluation metrics and provide insights into their strengths and limitations.

a. YOLOv8

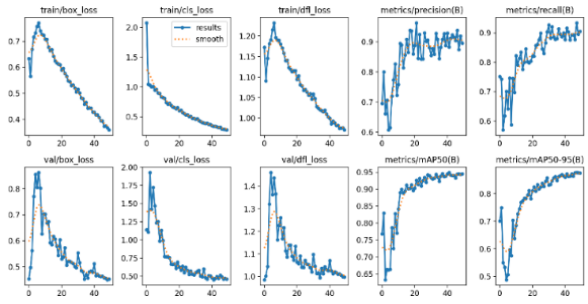

Fig. 32. Summary of Line Plot Performance Representation

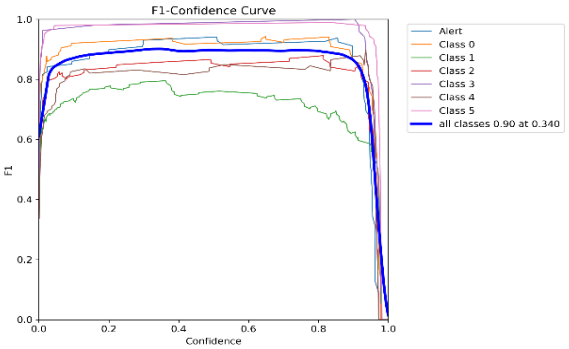

Fig. 33. F1-Confidence Curve in Each Vehicle Classes.

TABLE II. SUMMARY OF YOLOv8 PERFORMANCE RESULTS

Class	Images	Instances	Precision	Recall	mAP 50	mAP 50-95
All	241	337	0.874	0.934	0.945	0.877
Alert	241	43	0.934	0.930	0.946	0.826
Class 0	241	54	0.906	0.963	0.983	0.929
Class 1	241	86	0.774	0.814	0.836	0.716
Class 2	241	47	0.822	0.894	0.915	0.856
Class 3	241	26	0.971	1.000	0.995	0.934
Class 4	241	33	0.725	0.958	0.958	0.905
Class 5	241	48	0.987	0.979	0.982	0.970

The evaluation of the YOLOv8 model revealed promising results. It successfully detected a total of 337 instances across 241 images, resulting in a high average precision (MAP) of 0.945. The model demonstrated a strong capability to identify bounding boxes accurately, as indicated by a precision score of 0.874 and a recall score of 0.934. The MAP@0.50-0.95 value of 0.877 further highlights the model's proficiency in handling objects with varying levels of overlap and positioning. We also assessed the performance of YOLOv8 for individual vehicle classes, such as Alert, Class 0, Class 1, Class 2, Class 3, Class 4, and Class 5. The results showcased varying precision, recall, and average precision levels for each class, offering insights into the model's performance across different vehicle types.

YOLOv8 performs exceptionally well in detecting the 'Alert' class, achieving high precision, recall, and mAP scores. The precision of 0.934 indicates that when the model predicts this class, it is correct 93.4% of the time. The recall of 0.930 indicates that the model captures 93.0% of all actual instances of 'Alert' in the dataset. The mAP values are also notably high, demonstrating the model's effectiveness in recognizing this class accurately.

For 'Class 0,' YOLOv8 exhibits a high precision of 0.906, signifying a solid ability to correctly identify instances of this class. The recall of 0.963 indicates that it effectively captures the majority of actual instances of 'Class 0.' The mAP values are also notably high, demonstrating the model's accuracy in detecting this class.

'Class 1' exhibits a lower precision and recall compared to the previous classes. The precision of 0.774 suggests that the model's predictions for this class may include some false positives. The recall of 0.814 indicates that it captures around 81.4% of actual instances of 'Class 1.' The mAP values, while lower, still indicate a reasonable level of detection performance for this class.

YOLOv8 demonstrates a good balance of precision and recall for 'Class 2.' The precision of 0.822 indicates that it maintains a relatively low rate of false positives. The recall of 0.894 shows that it captures a significant portion of actual instances of 'Class 2.' The mAP values further validate its effectiveness in detecting this class.

YOLOv8 excels in 'Class 3' detection, achieving near-perfect precision and recall scores. The precision of 0.971 indicates an extremely low rate of false positives. The recall of 1.000 signifies that it captures all instances of 'Class 3' in

the dataset. The mAP values reinforce its outstanding performance for this class.

While 'Class 4' exhibits a lower precision of 0.725, indicating some false positives, it compensates with a high recall of 0.958, capturing the majority of actual instances. The mAP values also suggest a good overall performance in detecting this class.

YOLOv8 performs exceptionally well in detecting 'Class 5,' achieving high precision, recall, and mAP scores. The precision of 0.987 indicates a very low rate of false positives, and the recall of 0.979 signifies its ability to capture a significant portion of actual instances of 'Class 5.'

On the other hand, YOLOv8 demonstrates strong individual class performance across most classes, with particularly impressive results for 'Alert' and 'Class 3.' While some classes exhibit lower precision, the model generally maintains a balanced trade-off between precision and recall, resulting in solid mAP scores for most classes. Nevertheless, these results collectively underscore YOLOv8's proficiency in object detection tasks, offering a robust and balanced performance across diverse vehicle classes in our dataset.

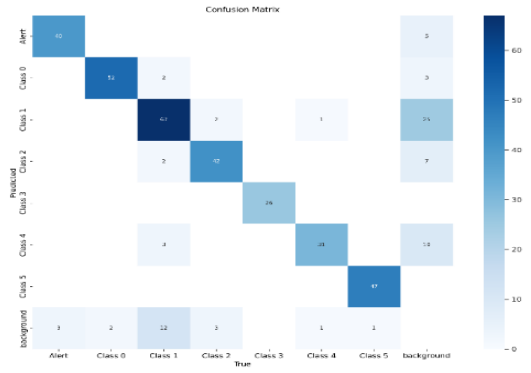

Fig. 34. Confusion Matrix in Each Vehicle Classes.

In the confusion matrix provided for YOLOv8, the true positive results for each vehicle class offer valuable insights into the model's performance. Notably, the model correctly identified 40 instances in the' Alert' class, showcasing its ability to effectively detect this class. Similarly, for 'Class 0,' YOLOv8 demonstrated precision by correctly identifying 52 instances. 'Class 1' yielded 67 true positive results, signifying the model's capability to accurately detect this class. In 'Class 2,' the model achieved a true positive count of 42, indicating its proficiency in recognizing instances of this category. In the 'Class 3' category, YOLOv8 correctly identified 26 instances as true positives, underscoring its competence in accurately detecting this specific class. Furthermore, 'Class 4' demonstrated the model's strong detection capabilities with 31 true positives. Lastly, 'Class 5' yielded an impressive 47 true positive results, underlining the model's excellence in identifying this class. These true positive outcomes across various vehicle classes emphasize YOLOv8's effectiveness in correctly detecting and categorizing instances, thereby contributing to its overall robust performance in object detection tasks.

b. Faster R-CNN

```
INFO:tensorflow:Eval metrics at step 25000
I0527 15:54:47.057358 140391009699648 model_lib_v2.py:1015] Eval metrics at step 25000
INFO:tensorflow:         + DetectionBoxes_Precision/mAP: 0.553335
I0527 15:54:47.079287 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP: 0.553335
INFO:tensorflow:         + DetectionBoxes_Precision/mAP@.50IOU: 0.749452
I0527 15:54:47.081755 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP@.50IOU: 0.749452
INFO:tensorflow:         + DetectionBoxes_Precision/mAP@.75IOU: 0.630845
I0527 15:54:47.083630 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP@.75IOU: 0.630845
INFO:tensorflow:         + DetectionBoxes_Precision/mAP (small): -1.000000
I0527 15:54:47.085588 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow:         + DetectionBoxes_Precision/mAP (medium): 0.121264
I0527 15:54:47.087370 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP (medium): 0.121264
INFO:tensorflow:         + DetectionBoxes_Precision/mAP (large): 0.576085
I0527 15:54:47.089090 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Precision/mAP (large): 0.576085
INFO:tensorflow:         + DetectionBoxes_Recall/AR@1: 0.619587
I0527 15:54:47.090809 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Recall/AR@1: 0.619587
INFO:tensorflow:         + DetectionBoxes_Recall/AR@10: 0.725867
I0527 15:54:47.092524 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Recall/AR@10: 0.725867
INFO:tensorflow:         + DetectionBoxes_Recall/AR@100: 0.732505
I0527 15:54:47.094240 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Recall/AR@100: 0.732505
INFO:tensorflow:         + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0527 15:54:47.095745 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow:         + DetectionBoxes_Recall/AR@100 (medium): 0.487619
I0527 15:54:47.097468 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Recall/AR@100 (medium): 0.487619
INFO:tensorflow:         + DetectionBoxes_Recall/AR@100 (large): 0.747177
I0527 15:54:47.099346 140391009699648 model_lib_v2.py:1018]     + DetectionBoxes_Recall/AR@100 (large): 0.747177
INFO:tensorflow:         + Loss/RPNLoss/localization_loss: 0.310282
I0527 15:54:47.100760 140391009699648 model_lib_v2.py:1018]     + Loss/RPNLoss/localization_loss: 0.310282
INFO:tensorflow:         + Loss/RPNLoss/objectness_loss: 0.056886
I0527 15:54:47.102188 140391009699648 model_lib_v2.py:1018]     + Loss/RPNLoss/objectness_loss: 0.056886
```

Fig. 35. Evaluation Metrics with Mean Average Precision (mAP) values

```
DONE (t=1.68s).
Accumulating evaluation results...
DONE (t=1.11s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.553
 Average Precision  (AP) @[ IoU=0.50       | area=   all | maxDets=100 ] = 0.749
 Average Precision  (AP) @[ IoU=0.75       | area=   all | maxDets=100 ] = 0.631
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.121
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.576
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.620
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.726
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.733
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = 0.488
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.747
```

Fig. 36. Average Precision (AP) and Average Recall (AR) values

TABLE III. SUMMARY OF FASTER R-CNN PERFORMANCE RESULTS

Metrics (area = all, maxDets = 100)	Performance
mAP@.50:.95IoU	0.553335
mAP@.50IoU	0.749452
mAP@.75IoU	0.630845
AP@.50:.95IoU	0.553
AP@.50IoU	0.749
AP@.75IoU	0.631
AR@.50:.95IoU	0.733

As summarised in Table IV, the performance results for Faster R-CNN provide insights into the model's effectiveness in object detection. The metrics evaluated include mAP (mean Average Precision) at various Intersections over Union (IoU) thresholds, AP at specific IoU values, and AR (Average Recall) at a range of IoU thresholds.

The evaluation results produced by Faster R-CNN ResNet50 demonstrate the model's ability to detect objects accurately but with some variations compared to YOLOv8. The Faster R-CNN ResNet50 model detected 287 instances across 241 images.

At an IoU range of 0.50 to 0.95, Faster R-CNN achieves an mAP of 0.553, indicating its ability to accurately detect and classify objects with a wide overlap range with ground truth bounding boxes. This metric is crucial for assessing the model's overall object detection performance across all classes.

When considering a stricter IoU threshold of 0.75, the model maintains a respectable mAP of 0.631. This suggests that Faster R-CNN is capable of precise object localization, particularly when high overlap between predicted and actual bounding boxes is required.

The model's performance is further detailed with AP values at specific IoU thresholds. At IoU of 0.50, Faster R-CNN achieves an AP of 0.749, demonstrating its proficiency in detecting objects with moderate overlap. This is a valuable metric as it reflects the model's performance under common detection scenarios where objects may not be perfectly aligned with ground truth annotations.

In addition, the Average Recall (AR) at an IoU range of 0.50 to 0.95 is reported at 0.733. This indicates the model's ability to balance precision and recall across various IoU thresholds, ensuring it can effectively capture objects even with variations in overlap.

On the other hand, to make a comprehensive assessment, these results should be compared with those of other models, such as YOLOv8, to determine the most suitable option for specific use cases. In this case, it is important to note that the performance of the Faster R-CNN model is relatively lower compared to YOLOv8.

TABLE IV. OVERALL PERFORMANCE EVALUATION

Performance Evaluation	Precision	Recall	F1- score	MAP50
YOLOv8 DarkNet-53	1.00	0.97	0.90	0.95
Faster R-CNN ResNet-50	0.55	0.73	0.63	0.55

The overall performance evaluation of YOLOv8 DarkNet-53 and Faster R-CNN ResNet-50 reveals key distinctions between these object detection models. YOLOv8 DarkNet-53 stands out with a remarkable precision score of 1.00, indicating its exceptional ability to classify and locate objects within the given dataset precisely. This precision score signifies a low rate of false positives, which is a valuable attribute in applications where accuracy is paramount. Additionally, YOLOv8 exhibits a strong recall of 0.97, implying its proficiency in capturing a significant proportion of the actual objects present in the images. This balance between precision and recall is further reflected in its F1-score of 0.90, signifying robust object detection performance. Furthermore, YOLOv8 achieves a MAP50 of 0.95, underscoring its excellence in accurately localizing and classifying objects with a moderate overlap threshold.

Conversely, Faster R-CNN ResNet-50 demonstrates a lower precision score of 0.55, suggesting a higher likelihood of false positives than YOLOv8. While its recall score of 0.73 indicates its capability to capture a substantial portion of actual objects, it does not match the recall performance of YOLOv8. This trade-off between precision and recall is mirrored in its F1-score of 0.63. Additionally, Faster R-CNN achieves a MAP50 of 0.55, indicating that it excels in certain detection scenarios but may struggle with higher accuracy demands.

Therefore, YOLOv8 DarkNet-53 exhibits superior precision, recall, F1-score, and MAP50 values compared to Faster R-CNN ResNet-50, emphasizing its effectiveness in precise and comprehensive object detection tasks. However, the choice between these models should be based on specific application requirements, as Faster R-CNN may still be a suitable option in scenarios where precision-recall trade-offs are acceptable or computational efficiency is a significant concern.

c. Prediction and Comparative Analysis of Model Results on Test Data

a) YOLOv8 DarkNet-53

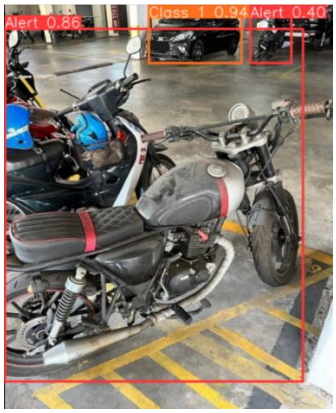

Fig. 37. YOLOv8 result for Image 1.

Fig. 38. YOLOv8 result for Image 2.

b) Faster R-CNN ResNet-50

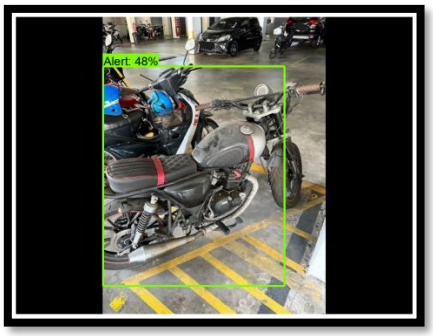

Fig. 39. FRCNN results for Image 1.

Fig. 40. FRCNN results for Image 2.

TABLE V. SUMMARY OF FASTER R-CNN PERFORMANCE RESULTS

Images	Detected Vehicles and Confidence Values	Correctly Classified (Yes/No)
Image 1 YOLOv8	1. Class 1: 0.94 2. Alert: 0.86 3. Alert: 0.40	Yes Yes Yes
Image 2 YOLOv8	1. Class 0: 0.96	Yes
Image 1 FRCNN	1. Alert: 0.48	Yes
Image 2 FRCNN	1. Class 0: 0.86	Yes

In detecting and classifying vehicles using YOLO and Faster R-CNN ResNet50 models, we observed differences in the number of vehicles detected and the confidence levels assigned to the classifications.

Using the YOLO model, Image 1 correctly classified two instances of the motor as Class Alert with a confidence level of 0.86 and 0.40 for the second. Additionally, it accurately classified a car as Class 1 with a high confidence level of 0.94. In Image 2, the YOLO model correctly identified a police car as Class 0 with a confidence level of 0.96.

Comparatively, the Faster R-CNN ResNet50 model exhibited some variations in its detections. In Image 1, it only detected and classified one motor as Class Alert, but with a lower confidence level of 0.48. It did not detect the additional motor or car in the image. Similarly, in Image 2, the model correctly classified the police car as Class 0, but with a confidence level of 0.86.

Both models produced correct classifications for the vehicles present in the images. However, the YOLO model detected and classified more vehicle instances than the Faster R-CNN ResNet50 model. The YOLO model also tended to assign higher confidence levels to its classifications, indicating a higher degree of certainty in its predictions.

These observations suggest that the YOLO model may have a more robust overall detection capability, capturing a greater number of vehicles accurately and with higher confidence. On the other hand, the Faster R-CNN ResNet50 model demonstrated limitations in detecting multiple instances and exhibited lower confidence levels in its classifications.

c) Findings for the Optimal Model: ***YOLOv8 with Darknet-53***

By using the best pre-trained weight model named "best.pt" from YOLOv8 model, we test the capability of the model to detect in two different ways. One way uses test images with multiple images of vehicles, while the other uses images with less than three vehicle classes in each image. The text appears in the detected green, orange, and red boxes, it shows the information on the type of vehicle toll classes versus MAP. For example, in Figure 45, Class 1:0.97 means the detected vehicle is from Class 1, and its confidence score is 0.97, equal to 97%. As depicted below, Figure 41 and Figure 42 illustrate the model cannot capture correctly when handling multiple classes in one frame, that the vehicle classes are not accurately labeled, and that for certain correct classes, low confidence values are recorded. However, vehicles in Figures 43, 44, 45, and 46 record high confidence values and have been captured with correct trained classes accordingly.

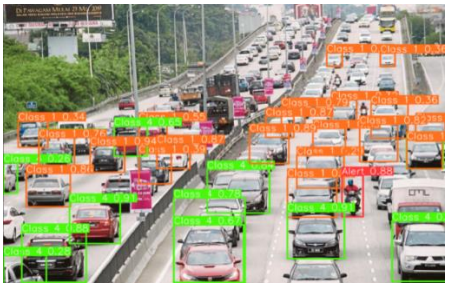

Fig. 41. Result 1 for detecting multiple vehicles in one frame.

Fig. 42. Result 2 for detecting multiple vehicles in one frame.

Fig. 43. Result 1 for detecting single vehicles in one frame.

Fig. 44. Result 2 for detecting single vehicles in one frame.

Fig. 45. Result 3 for detecting single vehicles in one frame.

Fig. 46. Result 4 for detecting single vehicles in one frame.

These findings emphasize the importance of considering both the number of detections and the confidence levels when evaluating the performance of object detection models. Further analysis and comparison of these models across a more extensive and diverse dataset could provide additional insights into their respective strengths and weaknesses in vehicle detection and classification tasks.

The overall results indicate that the YOLOv8 model performs well compared to the F-RCNN model. Therefore, we predict that the setup and training process for the Faster-RCNN model might be wrong. The possibilities might be because of overfitting, the hyperparameter tuning used was unsuitable and needed the adjustment of the parameters defined, the architecture needed more train data, and the differences in training parameters for both models. For the Phase 2 Project, we will -tune the model, try different parameters, try on a new dataset, do data augmentation, and generate synthetic data where we are adding another new feature. The data can be used to adapt to the weather changing in Malaysia.

B. Future Works

The overall results indicate that the YOLOv8 model performs well compared to the F-RCNN model. Therefore, we predict that the setup and training process for F-RCNN model might be wrong. The possibilities might be because of overfitting, the hyperparameter tuning used was unsuitable and needed the adjustment of the parameters defined, the architecture needed more train data, and the differences in training parameters for both models. For the Phase 2 Project, we will fine-tune the model using different parameters, insert a new dataset if needed, and add new features such as data augmentation and synthetic data generation where the data can be used to adapt to the weather changes in Malaysia.

The first feature is data augmentation. Augmented data is derived from original data with minor adjustments. To improve the size and diversity of the training set, we use geometric and color space modifications like flipping, resizing, cropping, brightness, and contrast [18]. Secondly, generates synthetic data that is created without applying the actual dataset. To produce synthetic data, it frequently employs DNNs (Deep Neural Networks) and GANs (Generative Adversarial Networks) [18].

While both models are being optimized, we plan to gather information about Malaysian citizens' experiences with the current toll system and their feedback and suggestions. We will collect the data using Google Forms, which includes several types of questionnaires: structured, unstructured, open, and closed. We would like to explore and summarize the users' insights about our project idea.

Moreover, our current output only shows the vehicle class types and their confidence value for model deployment, respectively. To improve the effectiveness of the toll system implementation for both models, we will also indicate the bounding box with each vehicle's fare. For example (Class 1 RM 5.00). To increase the approach to the current Malaysian toll system, we will try to contact one of the highway concessionaries or build–operate–transfer operator companies in Malaysia, PLUS Expressways Berhad, to confirm how much in one frame vehicles can be detected and other related questions regarding the improvements of the system in toll system in Malaysia.

REFERENCES

[1] Transport Statistics Malaysia 2021. (2021). https://www.mot.gov.my/en/Statistik%20Tahunan%20Pengangkutan/Transport%20Statistics%20Malaysia%202021.pdf.

[2] Syed Izmir. (2022). Research on Improvement of Project Implementation for RFID Toll in Malaysia. Faculty of Built Environment University of Malaya Kuala Lumpur. http://studentsrepo.um.edu.my/14121/3/Syed_Izmir.pdf.

[3] Wenbo Lan, Jianwu Dang, Yangping Wang & Song Wang. (2018). Pedestrian Detection Based on YOLO Network Model. 2018 IEEE International Conference on Mechatronics and Automation : IEEE ICMA 2018 : August 5-8, 2018, Changchun, China.

[4] Yin, Y., Li, H. & Fu, W. (2020). Faster-YOLO: An accurate and faster object detection method. Digital Signal Processing: A Review Journal, 102. https://doi.org/10.1016/j.dsp.2020.102756.R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[5] Kim, J. A., Sung, J. Y. & Park, S. H. (2020, November 1). Comparison of Faster-RCNN, YOLO, and SSD for Real-Time Vehicle Type Recognition. 2020 IEEE International Conference on Consumer Electronics - Asia, ICCE-Asia 2020. https://doi.org/10.1109/ICCE-Asia49877.2020.9277040.

[6] Aleksa Ćorović, Velibor Ilić, Siniša Đurić, Mališa Marijan, and Bogdan Pavković. (2018). The Real-Time Detection of Traffic Participants using YOLO Algorithm. 26th Telecommunications forum TELFOR. https://doi.org/10.1109/TELFOR.2018.8611986.

[7] Diwan, T., Anirudh, G. & Tembhurne, J. V. (2023). Object detection using YOLO: challenges, architectural successors, datasets and applications. Multimedia Tools and Applications, 82(6), 9243–9275. https://doi.org/10.1007/s11042-022-13644-y.

[8] Yuejin Zhang, Guanxiang Yin, Meng Yu, Meng Wang & Yong Hu. (2022, March). Research on highway vehicle detection based on faster R-CNN and domain adaptation. ResearchGate, 52(02):1-16. https://doi.org/10.1007/s10489-021-02552-7.

[9] Wen Li, Haoran Wang, Yuhua Chen, Christos Sakaridis, Dengxin Dai & Luc Van Gool. (2021, May 11). Scale-Aware Domain Adaptive Faster R-CNN. Springer, International Journal of Computer Vision volume 129, pages 2223–2243. https://doi.org/10.1007/s11263-021-01447-x.

[10] Jiani Xi, Zhihui Wang & Daoerji Fan. (2020). A Solution for Vehicle Attributes Recognition and Cross-dataset Annotation. 2020 13th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9263546.

[11] Dilek, E. & Dener, M. (2023). Computer Vision Applications in Intelligent Transportation Systems: A Survey. Sensors, 23(6), 2938. https://doi.org/10.3390/s23062938.

[12] Usha Mittal, Priyanka Chawla & Rajeev Tiwari. (2023). EnsembleNet: a hybrid approach for vehicle detection and estimation of traffic density based on faster R-CNN and YOLO models. Neural Computing and Applications volume 35, pages 4755–4774. https://doi.org/10.1007/s00521-022-07940-9.

[13] Encord. (2022, March). YOLOv8 for Object Detection Explained [Practical Example]. Retrieved from https://medium.com/cord-

tech/yolov8-for-object-detection-explained-practical-example-23920f77f66a.

[14] Keita, Z. (2022, September). YOLO Object Detection Explained. Retrieved from https://www.datacamp.com/blog/yolo-object-detection-explained.

[15] Ren, S., He, K., Girshick, R. & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Computer Vision and Pattern Recognition. http://arxiv.org/abs/1506.01497.

[16] Deval Shah. (2022, March 7). Mean Average Precision (mAP) Explained: Everything You Need to Know. Retrieved from https://www.v7labs.com/blog/mean-average-precision.

[17] Jacob Solawetz, F. (2023, January 11). Roboflow. Retrieved from What is YOLOv8? The Ultimate Guide: https://blog.roboflow.com/whats-new-in-yolov8/.

[18] Datacamp. (2022, November). A Complete Guide to Data Augmentation. Retrieved from https://www.datacamp.com/tutorial/complete-guide-data-augmentation.

[19] [Murtaza's Workshop - Robotics and AI]. (2023, February 11). Object Detection 101 Course - Including 4xProjects | Computer Vision [Video]. Retrieved from https://www.youtube.com/watch?v=WgPbbWmnXJ8.