Palestine Technical University - Kadoorie

Faculty of Engineering and Technology

Department of Computer Systems Engineering

# SMART TRAFFIC MANAGEMENT SYSTEM

By:

Besan Khaled Omar Noor Jehad Khawaja

Maha Emad-Aldin Abu Hamdeh

Supervisor:

Dr. Anas Melhem

A graduation project submitted in partial fulfilment of the requirements for the bachelor's degree in computer systems engineering.

Tulkarm, Palestine

June, 2024

# DEDICATION

الحمدُ لله حُبّاً وشُكراً وامتِناناً، ما كُنّا لنَفعَلَ هذا لولا فضل الله، فالحمدُ لله على البَدءِ وعلى الختامِ.

"وَآخِرُ دَعْوَاهُمْ أَنِ الْحَمْدُ لِلَّهِ رَبِّ الْعَالَمِينَ"

وَسط أحداثٍ أليمةٍ وقعت منذ قُرابة عشرةِ شُهور، ومجازر بشعة وقعت بأبناء شَعبِنا وغَزَّتِنا الحبيبة، ومن قلبِ اقتحاماتٍ امتدت أياماً وأياماً، نَصِلُ اليومَ إلى مُبتغانا وقَلبُنا يَعتصر ألماً وثأراً لأبناء جِلدتنا، ونَقِفُ شامِخين رافعينَ الرأسَ تحيةَ رَحمةٍ وإجلالٍ لشُهدائنا الأبرار، رَحِمهم الله. وما نحن مُكمِّلين اليوم إلا رَغبةً في إحياء بلادِنا وإعادة إعمار فلسطين جديدة، ورَغبةً لقَهر الكِيان الذي يَهدفُ إلى محو وجود فلسطين ونفوذها في العالم ومُتعلِّميها وعُلمائها، وتَعميم الجَهلِ فيها، ولكن هيهاتَ له ذلك.

من قال أنا لها "نالَها"

وأنا لها وإن أبت رغماً عنها أتيت بها.

لم تكن الرِّحلة قصيرة ولا ينبغي لها أن تكون، لم يكن الحُلم قريباً ولا الطريق كان مَحفوفاً بالتسهيلات، لكننا فعلناها ونلناها.

أُهدي هذا النجاح لنفسي أولاً، ثم إلى زميلاتي وكل من سَعى معي لإتمام هذه المسيرة، دمتم لي سنداً لا عُمر له.

وأُهدي ثواب هذا المشروع إلى من لا ينفصل اسمي عن اسمه، إلى مأمني الوحيد وفرحتي الدائمة، إلى مصدر قوتي الذي طالما عاهدته بهذا النجاح وها أنا أتممتُ وعدي وأهديته إليك:

"أبي العزيز".

إلى ملاكي في الحياة، إلى معنى الحُب وإلى معنى الحنان والتفاني، إلى من كان دعاؤها سرَّ نجاحي، إلى داعمتي الأولى ووجهتي التي أستمدُّ منها القوة:

"أمي الحبيبة".

إلى الداعمين الساندين، إلى أرضي الصلبة وجِداري المتين، إلى من مُدت أيديهم في أوقات ضعفي، إلى الواقفين خلفي مثل الظل مهما كثرت تخبطاتي، إلى من بَذلوا جُهداً في مساعدتي وكانوا عَوناً وسنداً:

"إخواني وأخواتي".

إلى رفقاء الروح الذين شاركونا خُطوات هذا الطريق، إلى من شجَّعونا على المثابرة وإكمال المسيرة، إلى الشموع التي تُنير لنا الطريق دوماً، إلى رفقاء السنين، مُمتنين لكم.

وفي النهاية نُقدِّم الشكر إلى من كانوا بفكرهم عُلماء، وبتواضعهم رُفقاء، وبتَرفُّعهم كُبراء، إلى دكاترتِنا الكِرام كُلٌّ بِاسْمِهِ ولَقَبِهِ، كما نَتَقدَّم بالشُّكر الخاصِّ للدكتور أنس مِلحِم على ما قَدَّمَهُ لنا من توجيهٍ وإرشادٍ في إعدادِ هذا المشروع.

# ACKNOWLEDGMENTS

We would like to take this opportunity to thank Allah, the merciful for being our guide, giving us the strength to write this thesis.

Then we extend our sincere appreciation to our project supervisor Dr. Anas Melhem, whose guidance, support, and invaluable insights have been instrumental throughout this journey. He provided us with continuous encouragement, expert advice, and unwavering commitment, steering us towards excellence and fostering our professional growth.

Our sincere appreciation also goes to our fellow classmates and friends for their moral support, encouragement, and camaraderie throughout this challenging yet rewarding endeavor.

We would like to express our gratitude to our families for their unwavering support, understanding, and encouragement throughout our academic journey. Their love and encouragement provided us with the strength and motivation to overcome obstacles and pursue our goals.

Thank you to all those who have contributed to our project's success, directly or indirectly. Your support and encouragement have been invaluable, and we are sincerely grateful for your contributions.

## ABSTRACT

Traffic congestion become an irritating problem that affects our daily life. The weak infrastructure, and the absence of the basic solutions such as tunnels, bridges, and trains motivate us to develop a smart traffic management system to alleviate this problem. The developed system exploits deep learning technologies to determine the density of the traffic, and organize the traffic accordingly. The proposed system capture images periodically, and leverages YOLOv8 to process the captured images and determine the density of the traffic allowing the system to make decisions on how to organize traffic. Our system aims to provide an immediate solution to the traffic problem.

# ملخص

مشروعنا يتناول تطوير نظام ذكي لإدارة حركة المرور يهدف إلى تقليل الازدحام المروري وتحسين انسيابية الحركة المرورية باستخدام تقنيات معالجة الصور والذكاء الاصطناعي. يعتمد النظام على تحليل الصور الملتقطة بشكل دوري لتحديد كثافة المرور واتخاذ القرارات المناسبة لتنظيم الحركة. تم استخدام نموذج YOLOv8 لتحديد كثافة المرور بناءً على البيانات المستخرجة، مما يتيح للنظام تحسين تدفق المركبات وتخفيف الازدحام بشكل فعال. يهدف النظام أيضًا إلى تحسين السلامة المرورية وتقديم الأولوية لمركبات الطوارئ لضمان استجابة سريعة وفعالة في الحالات الحرجة.

# Table of Contents

# LIST OF FIGURES

# LIST OF TABLES

# CHAPTER 1

# INTRODUCTION

In this chapter, we will provide an overview of our project.

Traffic congestion is a condition that occurs on the roads when the traffic volume exceeds the capacity of the road, where the interaction between vehicles leads to slow down the flow of traffic, increasing travel times and leading to the accumulation of vehicles in the streets. The increasing numbers of vehicles on the road and the interactions between them in addition to various human behaviors, are among the reasons that cause traffic congestion [1]. Traffic congestion is a problem that has multiple negative effects, where long travel times caused by traffic will make drivers and pedestrians frustrated, in addition to the delay of employees or students arriving to their place of work or study and may generally lead to wasting hours that could be spent at work or rest. Traffic crisis causes great harm to the environment, as inadequate traffic management leads to increased fuel consumption and high levels of greenhouse gas emissions, which increases the problem of air pollution, these emissions have serious environmental consequences and contribute to global climate change [2]. The damage resulting from the traffic crisis is not limited to this only, but the large number of vehicles represents safety concerns, as the overburdened road network increases the chances of accidents and collisions occurring, which poses a major threat to the safety of all road users [3]. Since the traffic crisis is a problem with many harmful effects, efforts have always been made to address this problem, and one of the solutions is traffic lights, which stand as indispensable tool in addressing the traffic crisis, serving as crucial regulators of vehicular flow at intersections. Their primary function is to organize and optimize the movement of vehicles, pedestrians, ensuring a systematic and safe progression of traffic [4]. However, as time progresses and the number of vehicles multiplies, the traffic crisis requires more comprehensive and effective solutions than traffic lights. Trains, buses, tunnels, and bridges also represent effective solutions to congestion in many parts of cities, as they offer effective alternatives to individual vehicle transportation. Unfortunately, these solutions face major challenges in Palestine due to the problem of weak infrastructure, as the implementation of a public transportation system depends heavily on road networks. reliable and infrastructure, which is often non-existent or insufficiently developed. Tunnels and bridges, which are critical to improving traffic flow, require significant investment and engineering expertise that may be limited by financial and other constraints [5]. This system is designed with the goal of effectively managing traffic.

## 1.1 Motivation

The idea of a smart traffic management system is driven by the following challenges:

1- **Enhancing Traffic Flow and Reducing Congestion:** Traditional traffic light systems are not responsive to real-time conditions, causing unnecessary delays. Our system dynamically adjusts traffic light timings based on current traffic densities, ensuring smoother traffic flow and reducing congestion.

2- **Prioritizing Emergency Vehicles for Safety:** Emergency vehicles often face delays due to traffic congestion, jeopardizing critical response times. Our system detects emergency vehicles in real-time, providing them immediate priority to navigate intersections quickly and safely, thereby improving response times and saving lives.

3- **Environmental and Economic impact:** Traffic congestion leads to increased fuel consumption, emissions, and economic losses. By optimizing traffic flow, our system reduces idling times and stop-and-go driving, leading to lower fuel consumption, reduced emissions, and significant economic savings.

## 1.2 Problems

The proposed system aims to address the following problems:

1- Time wastage in traffic congestion, leading to productivity losses and disruptions in daily routines.

2- Traffic congestion worsens when high-traffic roads are not allocated longer green signal times compared to roads with less traffic density.

3- Delays in traffic prevent emergency vehicles from responding quickly, potentially endangering lives.

4- Environmental impact from increased vehicle emissions due to prolonged traffic congestion.

## 1.3 Objectives

1- **Reducing congestion:** The primary goal of the system is to significantly reduce traffic congestion through the application of adaptive traffic signal control.

2- **Improving traffic flow:** The system seeks to improve traffic flow by adjusting the timing of traffic lights based on real-time traffic conditions. Thus, smoother and more efficient movement of vehicles through the city streets is ensured.

3- **Enhancing safety:** The system aims to improve road safety by reducing sudden stops, reducing driver frustration, and reducing the risk of accidents at intersections. The adaptive traffic light control system should contribute to safer road conditions for all passengers.

4- **Introducing smart city technology:** The system is in line with the broader smart city initiative, which aims to integrate advanced technology into infrastructure. Implementing smart traffic lights is a step towards creating smarter and more efficient cities.

# CHAPTER 2

# LITERATURE REVIEW

In this chapter, we will discuss the literature review (related works)

Object detection frameworks available in the literature are classified into two classes [6]; two-stages and one-stage frameworks. Two-stages frameworks which exploits convolutional neural networks (CNN) for classification the objects in the input images are developed earlier, they show higher accuracy but slower speed than one-stage frameworks. Two-stages frameworks such as R-CNN [7], fast R-CNN [8], and faster R-CNN [9] detect the objects in the input images in two steps:

1- generates the region of interest (ROI).
2- sending the ROI to subsequent network (CNN) for classification.

On the other hand, one-stage frameworks can achieve real time performance by skipping the region proposal step and thus the task can be formulated to regression problem. The one-stage frameworks have less accuracy compared with two-stages frameworks.

## 2.1 Two-stages frameworks

CNN was designed in 2012 to perform tasks that require object recognition, including image classification, detection, and segmentation. In CNN, the image is divided into multiple regions and then each region is classified into various classes. A lot of regions are needed to achieve accurate detection, so it takes high processing time [10]. To address this issue a selective search method that generates regions is used in R-CNN [7] which extracts around 2000 regions from each image and each region is passed to CNN separately. R-CNN process time is better than CNN but still slow (40-50 second). To improve the process time fast R-CNN has been developed. In fast R-CNN each image is passed only once to CNN and feature maps are extracted. Selective search [11] is used on these maps to generate predictions. When there is a large set of data, fast R-CNN takes a long time to complete the process because Selective search is slow. To solve the problem of time caused by selective search the region proposal network (RPN) has been used instead in faster R-CNN [9] which achieves object detection accuracy with real time detection speed. Faster R-CNN generates a lot of anchor boxes of different sizes, all over the image to accommodate all the objects present in an image. First a pretrained CNN like VGG16 [12] can be used to generate a feature map for input image. RPN checks which of the anchors really contain objects and modifies their boundaries to fit the object inside them. Now an object may be present in more than one overlapping anchor. Non-maximum suppression (NMS) [13] is used to keep only the best fitting anchor and reject the rest. These selected anchors (object proposals) are used to extract the fix length feature vectors from feature map using RoI pooling. These feature vectors are further used for multiclass classification to get the object present in the corresponding object proposal and for bounding box regressor to refine the boundaries of bounding box to precisely fit the object.

### 2.1.1 Applications on faster R-CNN

- **Car Detection from Low-Altitude UAV Imagery with the Faster R-CNN**

In [13] the Unmanned Aerial Vehicles (UAVs) are used to take 100 images from low altitude. Captured images are used to compare vehicle detection of the algorithms:
ViBe [14], Frame difference [15], Viola-Jones (V-J) [16], linear support machine (SVM) with histogram of orientated gradient (HOG) features (HOG + SVM) [17] and faster R-CNN [18].
The paper results show that faster R-CNN archives the best quality (94.94%) since it is not sensitive to vehicle in place rotation, and also has a high correctness (98.43 %) and completeness (96.40 %). This research only uses the Faster R-CNN networks for passenger cars detection, no emergency vehicles are detected.

- **Faster RCNN based robust vehicle detection algorithm for identifying and classifying vehicles**

In [19], the researchers proposed an improved vehicle detection algorithm based on the Faster R-CNN framework to address challenges in real-time vehicle detection, such as obscured and truncated cars, as well as scale variations in traffic photos. Their methodology involves the collection of a custom dataset from various road locations using installed cameras, recording videos at 60 frames per second for three days. The dataset comprises five vehicle categories and is categorized into difficulty levels. Annotating the dataset involves manual labeling, bounding box creation, and annotation storage in XML format. The implemented architecture incorporates different base networks, including modified Vgg16[13], with a proposal layer utilizing anchor boxes. The final detection results, including regression and classification boxes, are obtained through the classification and regression layers. Results on their dataset demonstrate enhanced detection efficiency and processing time compared to earlier Faster R-CNN models, showcasing the efficacy of their proposed framework.

## 2.2 One-stage frameworks

Recall one-stage frameworks are real time detectors with less accuracy than two-stage framework. You Only Look Once (YOLO) [20] is a one-stage framework that was first described in 2015. After that, improved versions of the YOLO algorithm were released annually until 2023. YOLO divides the input image into a grid and directly predicts bounding boxes and class probabilities for each cell, eliminating the need for separate proposal generation and classification steps. But YOLOv1 struggled to detect small objects and objects very close to each other. This is because the original YOLO algorithm can only recognize one object per grid cell, and in an attempt to address this problem, YOLO was developed into YOLOv2 with several features including:

Improved Feature Extraction: YOLOv2 integrates Darknet-19[21], a deeper and more powerful convolutional neural network, leading to improved feature extraction and accuracy.

Stabilization boxes: YOLOv2 introduced stabilization boxes to improve translation accuracy. These predefined boxes of different size and aspect ratios helped the model better predict the size and shape of different objects.

Yolo works according to these steps:

1- The image is captured and resized to standard size, 2- deep features are extracted from it using Darknet 19, 3- dividing the image into grid cells, 4- each of which is responsible for detecting objects within its region. 5-the pre-defined boxes guide the model in predicting the object's location and size. 6- the system then works on Each cell predicts bounding box coordinates, confidence score (object presence), separation probabilities, 7- outputs final detections with bounding boxes, class labels, and confidence scores.

### 2.2.1 An Application on YOLOv2

**A vehicle real-time detection algorithm based on YOLOv2 framework**

The algorithm proposed in [22] exploits YOLOv2 algorithm for real-time vehicle detection. The main contributions of the paper are:

1- **Increased grid size:** The authors expand the number of cells in the detection window from the original -7x7 to 14x14. This allows for more detailed feature extraction, especially for small objects.

2- **Optimized bounding boxes:** Instead of using the predefined bounding boxes from YOLOv2, the authors use k-means clustering to determine the optimal sizes and number of bounding boxes for their specific vehicle dataset. This results in better bounding box coverage for vehicles.

3- **Improved training and parameter tuning:** The authors optimize the training parameters and threshold setting to achieve a balance between accuracy and recall rate for vehicle detection. The improved algorithm achieves an accuracy of 91.80% and a recall rate of 63.86% on their vehicle dataset, which is slightly higher than both YOLOv2 and Faster R-CNN. Additionally, the algorithm runs at 27 frames per second, making it suitable for real-time applications.

The paper focuses on improving the accuracy and real-time performance of vehicle detection using YOLOv2. The main improvements involve increasing the grid size, optimizing bounding boxes, and fine-tuning training parameters. The improved algorithm achieves better accuracy and recall rate than YOLOv2 and Faster R-CNN while maintaining an acceptable frame rate.

## 2.3 Comparison between two-stage and one-stage detectors

**Table 1**: The difference between Faster R-CNN and YOLOv2

| Aspect | Two-Stage Detectors (Faster R-CNN) | One-Stage Detector (YOLOv2) |
|---|---|---|
| Framework Operation | RPN for proposals, subsequent nets for classification | Direct prediction without region proposals |
| Region Proposals | Utilizes RPN for generating region proposals | Skips explicit proposals, predicts directly |
| Processing Speed | Slower due to multi-stage process | Faster due to one-stage regression approach |
| Accuracy | High accuracy, especially in complex scenes | Slightly lower accuracy, suitable for real-time |
| Evolution | Evolved from CNN, R-CNN, Fast R-CNN to Faster R-CNN | YOLOv2 evolved from YOLOv1 |
| Complexity | More complex due to multi-stage architecture | Simpler architecture for direct predictions |
| Object Detection | Detailed object detection in varied environments | Real-time detection in dynamic scenarios |
| Computational Load | Higher due to multi-stage processing | Lower due to streamlined one-stage process |
| Key Strengths | Excellent accuracy in detailed object identification | Real-time operation suitable for dynamic apps |
| Application Focus | UAV imagery, complex object recognition | Real-time scenarios like traffic monitoring |
| Performance Metrics | High correctness (98.43%), completeness (96.40%) | Accuracy (91.80%), recall rate (63.86%) |

## 2.4 Conclusion

We can conclude that YOLOv2 is characterized by speed compared to Faster R-CNN, as shown in Table-1 below. YOLOv2 has been developed into several versions [23], the most recent version is YOLOv8 which is so far the fastest and most accurate. YOLOv8 is an anchor-free model which means that it predicts directly the center of an object instead of the offset from a known anchor box that will reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS) thus speeding up the detection process to real time [24]. This is why we will use YOLOv8 in our system.

# CHAPTER 3

# PROPOSED SOLUTION

In this section, we introduce the proposed model for a smart traffic management system aimed at optimizing traffic flow and prioritizing emergency vehicles at intersections. The system operates through multiple interconnected stages, beginning with input acquisition and concluding with real-time adjustments to traffic signals. Key components include real-time image capture and processing. By analyzing these images, the model detects traffic density and identifies emergency vehicles on each road, allowing for dynamic adjustments to traffic signals. This ensures efficient traffic management, prioritizing roads with emergency vehicles first, followed by those with high traffic density.

## 3.1 Proposed model

The proposed smart traffic management system operates as follows:

1- **Input:** The system's input involves two primary components:
   - **Real-Time Image Capture:** Ideally, input images would be captured from the intersection in real-time every 5 seconds. However, in our project, we use a custom dataset instead.
   - **Custom Dataset utilization of a custom dataset:** We collected a dataset, which is used for both training and testing the model.

2- **Image Processing and Object Detection with YOLOv8** shown in Figure 1:
   - **Grid Division:** YOLOv8 divides each input image into a grid of cells, with each cell responsible for detecting objects within its area.
   - **Feature Extraction:** The model extracts features from the image to recognize objects.
   - **Bounding Box Prediction:** For each cell, YOLOv8 predicts multiple bounding boxes, indicating potential object locations and sizes.
   - **Class Probability Prediction:** It also predicts the probability that each object belongs to a specific class (e.g., car, ambulance).
   - **Non-Maximum Suppression:** This technique removes overlapping bounding boxes, retaining only the most confident prediction for each object.

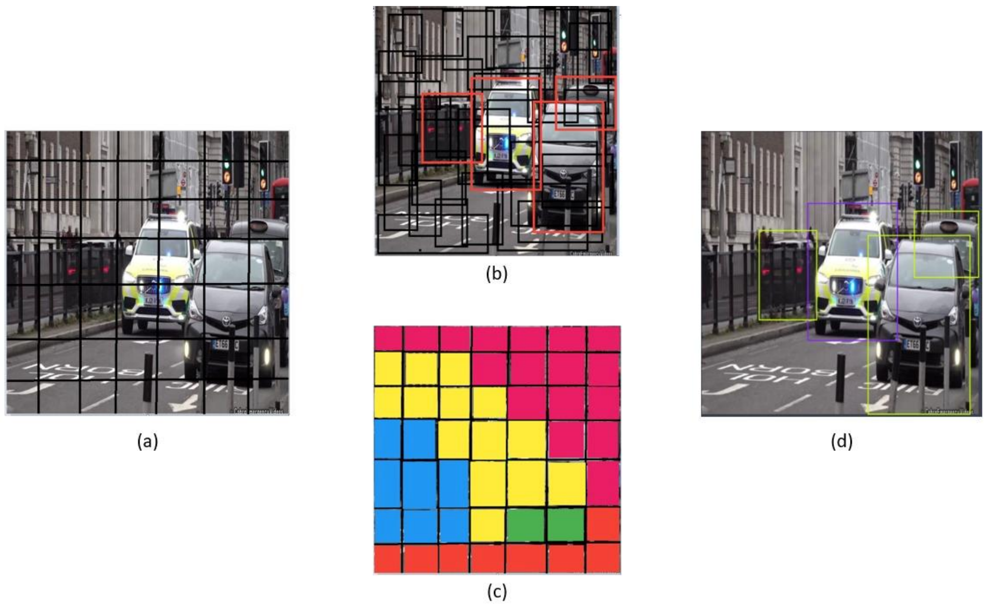


**Figure 1:** Illustrates the object detection process using the YOLO algorithm.(a) The input image is divided into an S×S grid. (b) For each grid cell, bounding boxes and their confidence scores are predicted. (c) A class probability map is generated, indicating the likelihood of each object class within the grid cells. (d) The final detections are produced by combining the bounding boxes and class probabilities, resulting in identified objects with their respective bounding boxes in the image.

**3- Traffic Analysis and Decision Making:**

- **Density Detection:** The model outputs a list of detected objects, showing traffic density on each lane and identifying the presence of emergency vehicles.
- **Traffic signal Adjustment:** Based on detected traffic density and emergency vehicle presence, the system adjusts traffic signals accordingly.
    - **Regular Operation:** Signals operate based on traffic density, with signal durations adjusted to optimize flow.
    - **Emergency Priority:** If an emergency vehicle is detected, traffic lights give it priority, ensuring it can navigate the intersection without delay.
    - **Post-Emergency Return:** After emergency vehicles pass, the system returns to normal traffic control based on current density.

## 3.2 Proposed scenario

In our project, we will consider a crossroad with 4 lines. Each line contains two lanes, that is numbered as 1.1, 1.2, …, 4.2 as shown in Figure 2. The traffic of each lane will be controlled by a traffic light.

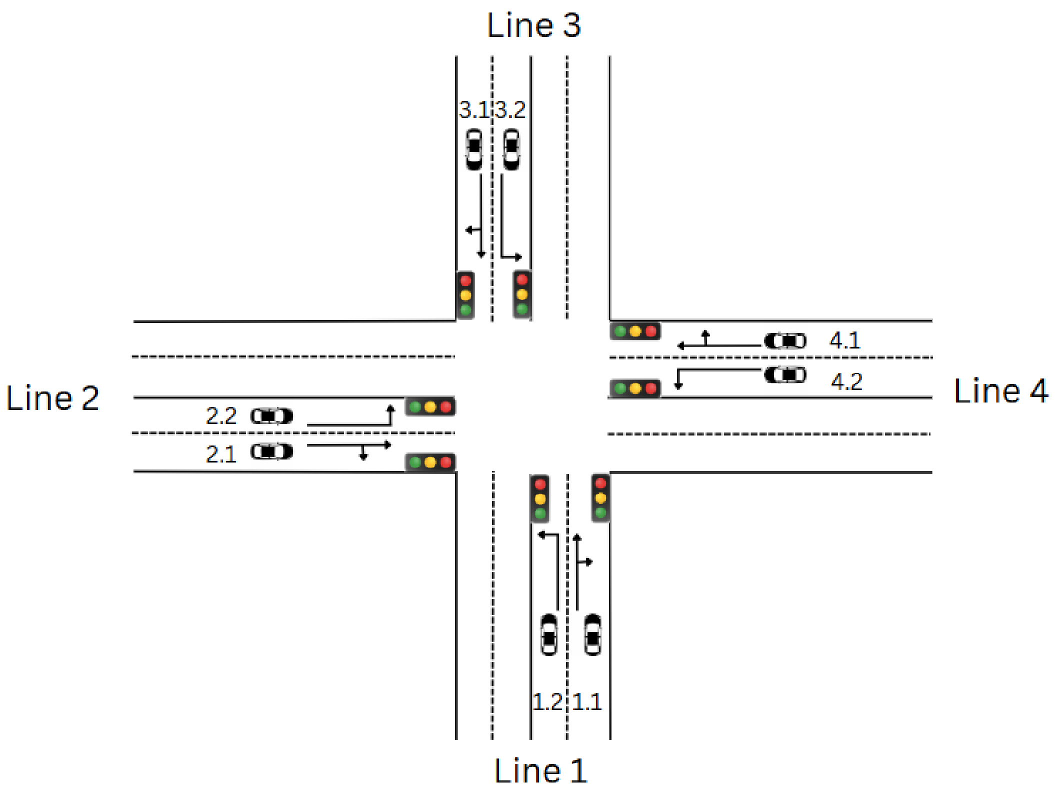


**Figure 2:** Crossroads by scenario

### 3.2.1 Considered cases

The traffic control system manages the crossroads, which consists of four main lines. Where Lines (1.1-3.1), (1.2-3.2), (2.1-4.1) and (2.2-4.2) will open at the same time as shown Figure 2 (with enforcement of traffic laws of course). In the proposed scenario, two aspects are considered: 1) the traffic density, and 2) the existence of an emergency vehicle. Thus, we end up with four different cases.

**Case 1: No emergency vehicles exist, equal density on all lanes:**

The system operates according to the prevailing traffic density cycle, providing the same time duration for each lane if traffic density is identical across all lanes.

If two or more lanes have identical densities, the system prioritizes them in a hierarchical order High, Medium, Low. Each category receives its allocated time duration once before passing the priority to the next in the queue.

**Case 2: No emergency vehicles exist, different densities:**

In this case different time is allocated for each traffic density as follow:

- No Traffic: 0 seconds (No cars present).
- Low Density: 5 seconds (1-20 cars).
- Medium Density: 15 seconds (20-35 cars).
- High Density: 30 seconds (35 cars or more).

**Case 3**: **Single emergency vehicle detection:**

Give immediate priority to the emergency vehicle ensuring its unobstructed passage, adjusting the green signal duration based pre-set time intervals depending on traffic density: 5 seconds (no traffic), 10 seconds (low), 15 seconds (medium), and 30 seconds (high).

**Case 4: Multiple emergency vehicles detected:**

Traffic lights give priority to the first emergency vehicle in the lane with low traffic density, allowing it to navigate the intersection without delay. After the first emergency vehicle passes, the traffic lights turn to give the second emergency vehicle the green light, allowing it to pass unhindered.

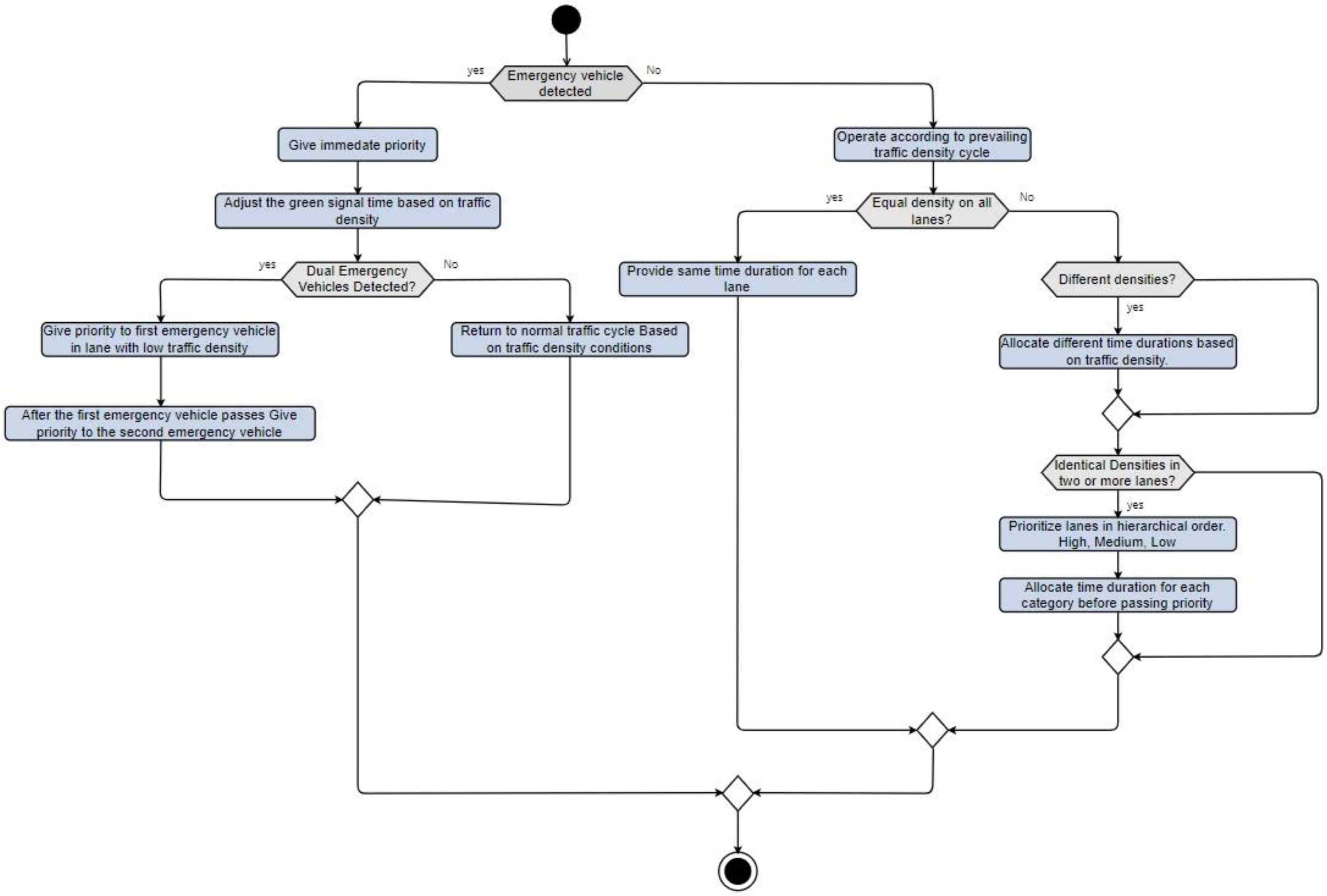
After the two emergency vehicles have passed, the system returns to the normal traffic cycle based on the prevailing traffic density conditions.

## 3.3 Diagrams

In this section we will present two diagrams to describe how the system works.

### 3.3.1 Activity Diagram

Activity diagram in Figure 3 shows how the system makes decisions based on the mentioned cases.



Emergency vehicle detected
yes
No
Give immedate priority
Adjust the green signal time based on traffic density
Dual Emergency Vehicles Detected?
yes
No
Give priority to first emergency vehicle in lane with low traffic density
After the first emergency vehicle passes Give priority to the second emergency vehicle
Return to normal traffic cycle Based on traffic density conditions
Operate according to prevailing traffic density cycle
Equal density on all lanes?
yes
No
Provide same time duration for each lane
Different densities?
yes
Allocate different time durations based on traffic density.
Identical Densities in two or more lanes?
yes
Prioritize lanes in hierarchical order. High, Medium, Low
Allocate time duration for each category before passing priority

**Figure 3:** Activity diagram

This activity diagram represents a traffic signal management process that prioritizes emergency vehicles and adjusts based on traffic density. When an emergency vehicle is detected, it is given immediate priority, and the green signal time is adjusted according to the traffic density. If there are dual emergency vehicles, the one in the lane with lower traffic density is prioritized first, followed by the second. If no emergency vehicle is detected, the system checks if the traffic density is equal across all lanes. If so, it provides the same signal duration for each lane. If the densities are different, the system allocates time durations based on traffic density, prioritizing lanes hierarchically if densities are identical in two or more lanes (High, Medium, Low). This ensures emergency vehicles are prioritized while optimizing regular traffic flow.

### 3.3.2 Sequence Diagram

In Figure 4 sequence diagram that shows the dynamic interactions and message flow between YOLOv8 and the Traffic Management System, emphasizing their collaboration in real-time decision-making for efficient traffic control and emergency handling.

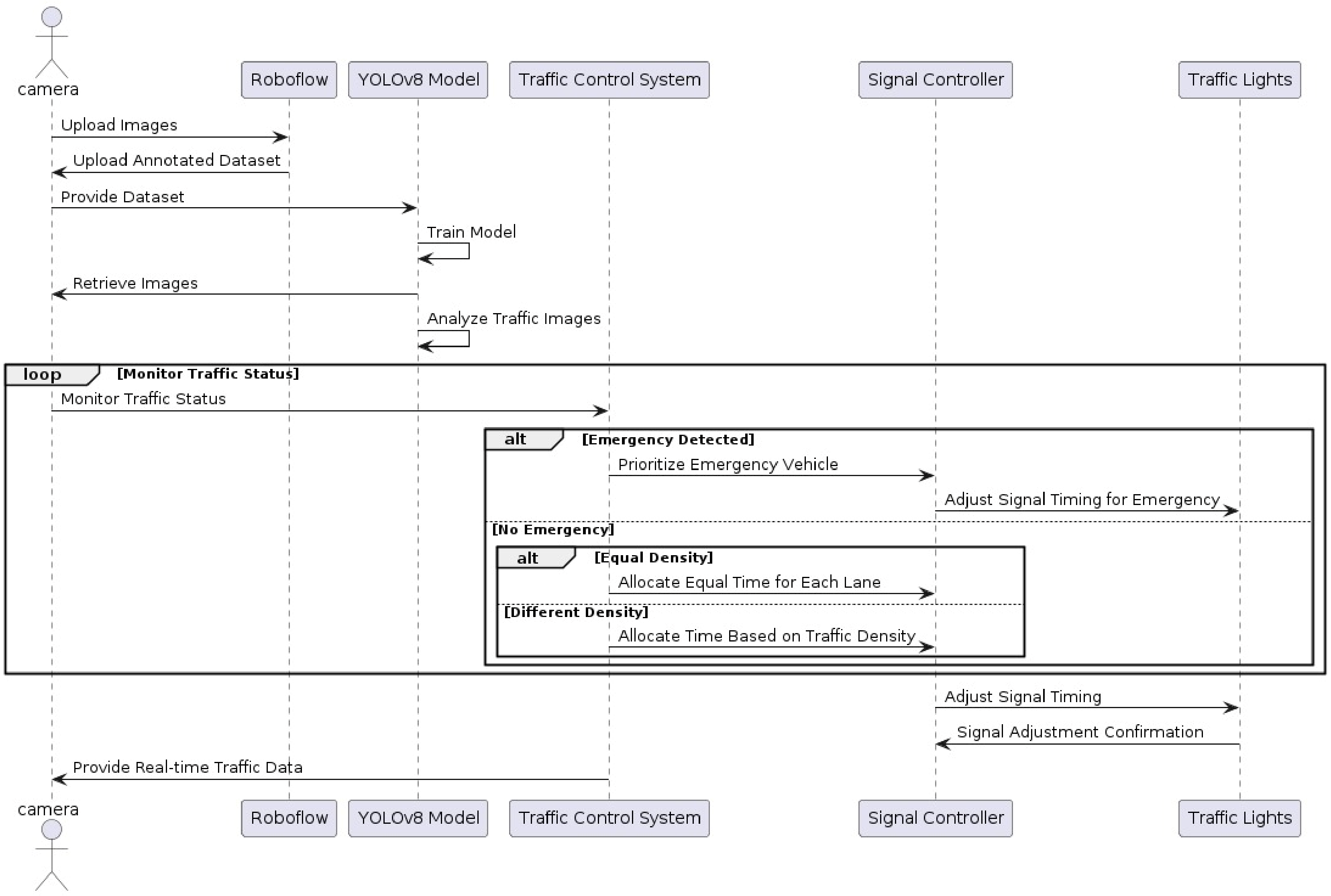


**Figure 4:** System sequence diagram

The provided diagram illustrates a traffic control system leveraging a YOLOv8 model for real-time traffic management and signal control. The process begins with a camera uploading traffic images to Roboflow, which processes and annotates the dataset before providing it to the YOLOv8 model. The model is then trained using this dataset to analyze traffic images. Once trained, the YOLOv8 model continuously monitors the traffic status by retrieving real-time images from the camera.

Within the monitoring loop, the system detects whether there is an emergency. If an emergency is detected, the system prioritizes the emergency vehicle by adjusting the signal timing accordingly. If no emergency is detected, the system assesses traffic density. If the traffic density is equal across all lanes, the system allocates equal signal time to each lane. If the traffic density varies, the system allocates signal time based on the detected traffic density for each lane.

The traffic control system then sends the adjusted signal timings to the signal controller, which confirms the adjustments and implements the new timings at the traffic lights. This entire process ensures efficient traffic flow management, prioritizing emergencies when necessary and optimizing signal timings based on real-time traffic conditions.

# CHAPTER 4

# IMPLEMENTATION

In this chapter, the procedural steps for developing the smart traffic management system are outlined. Initially, the dataset undergoes annotation using the Roboflow platform to ensure accurate image labeling for training. Subsequently, the YOLOv8 model is trained on these annotated images using computational resources from Google Colab. After training, decisions are made to manage the roads, prioritizing emergency vehicles first and then roads with high traffic density. This chapter delves into the intricacies of each step, detailing the methodology behind image annotation and model training. Additionally, the chapter provides the source code here.

## 4.1 Dataset description

The dataset[25], sourced from Kaggle [26] and Roboflow [27], is used to train a model for identifying and classifying various scenarios in traffic images, including detecting emergency vehicle presence and analyzing traffic density. We ensured that the collected data met our system's requirements by selecting images captured from a top perspective, ensuring they are clear, not distorted, captured during daylight, and provide a comprehensive view of traffic from above.

Prior to annotation, the dataset comprised 3,923 images. Among these, 99 images depicted emergency vehicle presence, while the rest depicted non-emergency scenarios. The non-emergency subset was further divided based on road density: 1,188 in no-density conditions, 1,094 in low-density, 822 in medium-density, and 728 in high-density traffic situations.

To refine the data segmentation approach, we established distinct subsets for training, validation, and testing. This adjustment resulted in 80.3% (3,004 images) allocated for training, 13.6% (507 images) for validation, and 6% (227 images) for testing. This distribution prioritized dense training data to enhance learning potential while also dedicating a substantial portion for validation to assess the model's efficacy.

Here are four levels of traffic density:

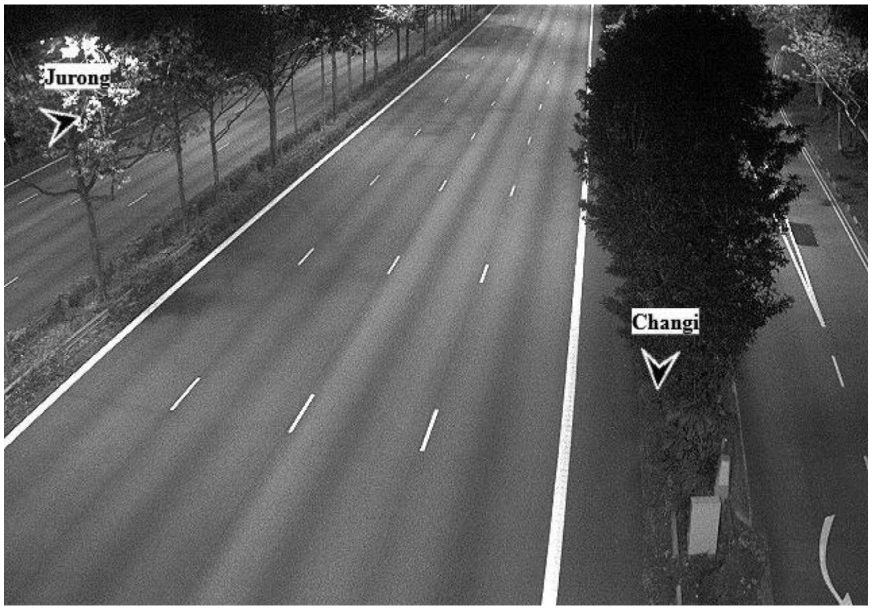


**Figure 5:** No traffic

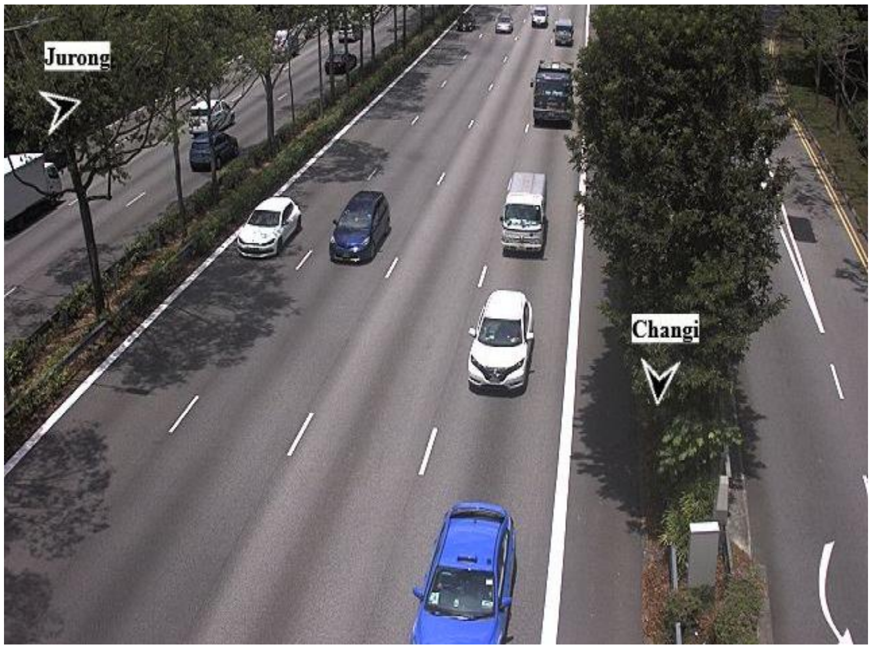


**Figure 6:** Low density

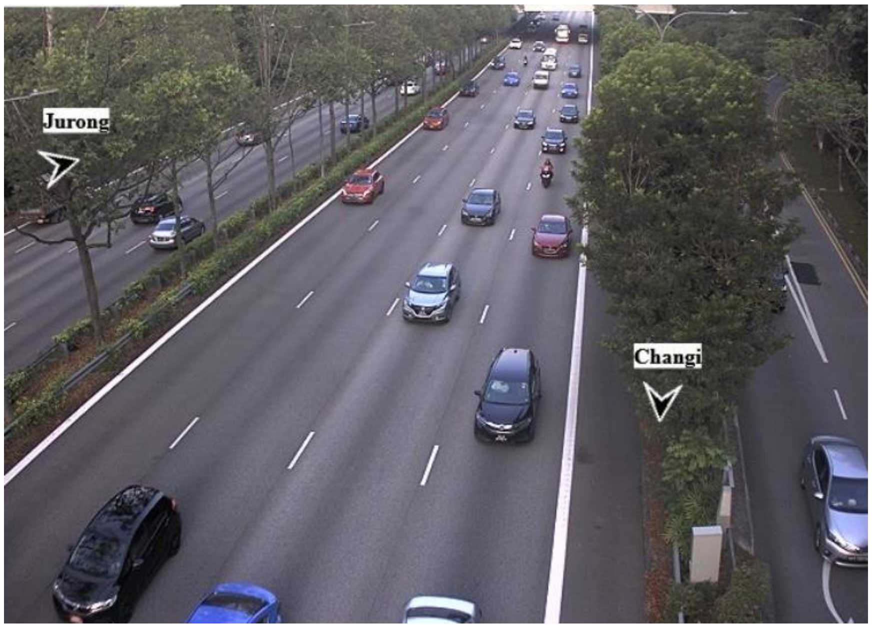


**Figure 7:** Medium density

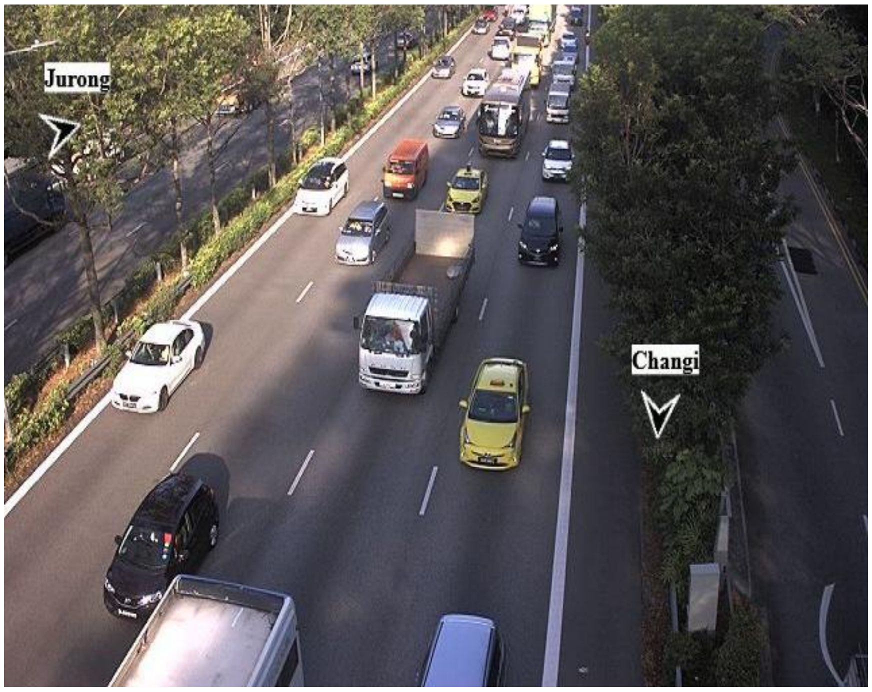


**Figure 8:** High density

**Figure 9:** Emergency vehicle

## 4.2 Annotation process

After exploring various options, our focus shifted to platforms offering automatic annotation capabilities, leading us to select Roboflow as the most suitable platform for our model. Roboflow[28] is an end-to-end computer vision platform that provides tools for developers to build and deploy computer vision models. It's designed to streamline the process of creating datasets, training models, and deploying them to production. Our interaction with Roboflow unfolded in two phases, each encompassing several steps, which will be detailed later: initially utilizing the platform's automated annotation features, followed by subjecting the annotations to a manual review process to ensure accuracy and comprehensiveness. Additionally, our process involved watching several informative YouTube videos that thoroughly explained the annotation process.

Roboflow offers a comprehensive suite of features, including Dataset Management, Labeling, Model Training, and Deployment. With Dataset Management, users can search, curate, and manage visual data from various sources. The Labeling tool provides options for annotating images and videos, allowing for precise and detailed labeling of objects. Model Training capabilities enable users to train models with just one click and receive results for deployment. Finally, Roboflow supports deploying models via hosted API or at the edge, facilitating seamless integration into production environments.

The annotation process on Roboflow[29] proceeded through the following steps within our model:

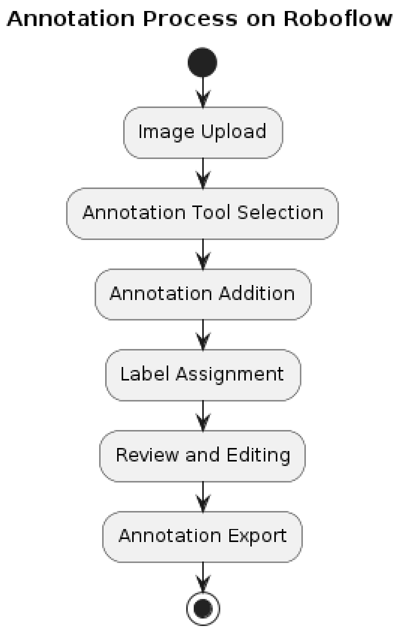


**Figure 10:** Annotation Process flowchart

1- **Image Upload:** We uploaded images in JPG format to the Roboflow platform for annotation.

2- **Annotation Tool Selection:** Within our model, we chose bounding boxes as the annotation tool, selecting from the options provided by Roboflow.

3- **Annotation Addition:** Utilizing the selected bounding box tool, we commenced adding annotations to vehicles on the road within the images.

4- **Label Assignment:** In addition to annotating the vehicles, we assigned labels to categorize them as cars or ambulances.

5- **Review and Editing:** Upon completing the annotations, we meticulously reviewed and edited them as necessary, carefully scrutinizing each image and either accepting or rejecting the annotations.

6- **Annotation Export:** Finally, we exported the completed annotations to our devices for further utilization within our model.

After completing the data annotation process, we transferred it to our devices for training with YOLOv8 in an optimal workspace. Visual Studio emerged as the preferred environment, as it housed all necessary Python scripts for training, detection, and counting.

Meanwhile, as we searched for the most suitable environment to execute our codes, we embarked on learning the Python language, which later became indispensable for all YOLOv8 codes. However, deploying YOLOv8 on Visual Studio proved challenging, as code execution and parameter adjustments, such as epoch and batch size, consumed significant time due to its CPU-centric processing. Consequently, we transitioned to the Google Colab platform to capitalize on its online data visualization tools, thereby streamlining our workflow.

## 4.3 Training process

The YOLOv8 model was trained on a dataset of traffic images from Roboflow using Google Colab. Google Colab[30], or Colaboratory, is an online platform where you can write and run Python code right in your browser. It provides access to GPUs, which is incredibly helpful for machine learning projects. Colab also makes it easy to share notebooks and collaborate with others, allowing for real-time comments and edits.

The dataset we used includes a variety of traffic scenarios, such as vehicles and ambulances. After training, the YOLOv8 model is able to effectively detect and categorize objects in real-time traffic scenes captured by cameras.

To train model using Google Colab[31] followed these steps:

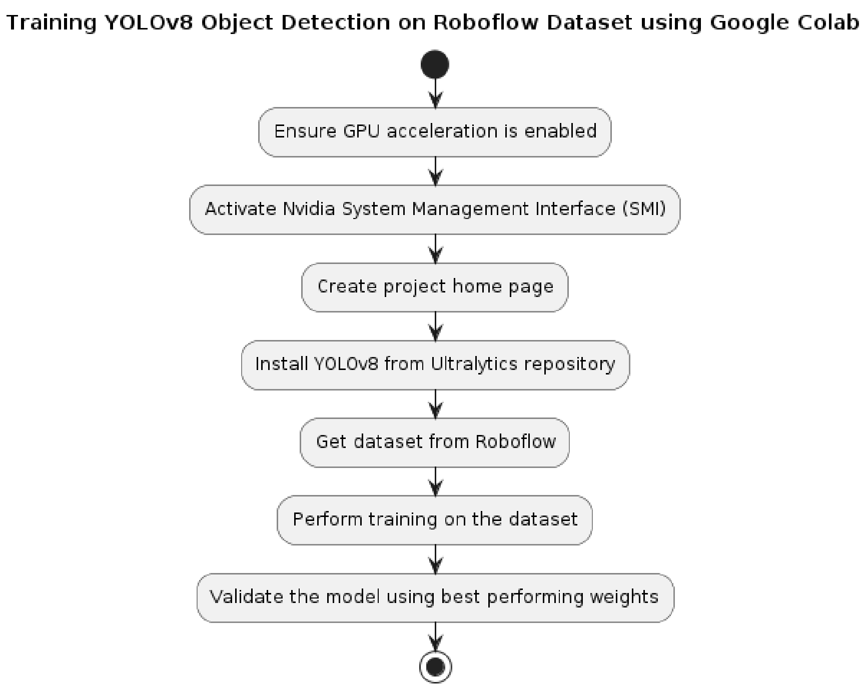
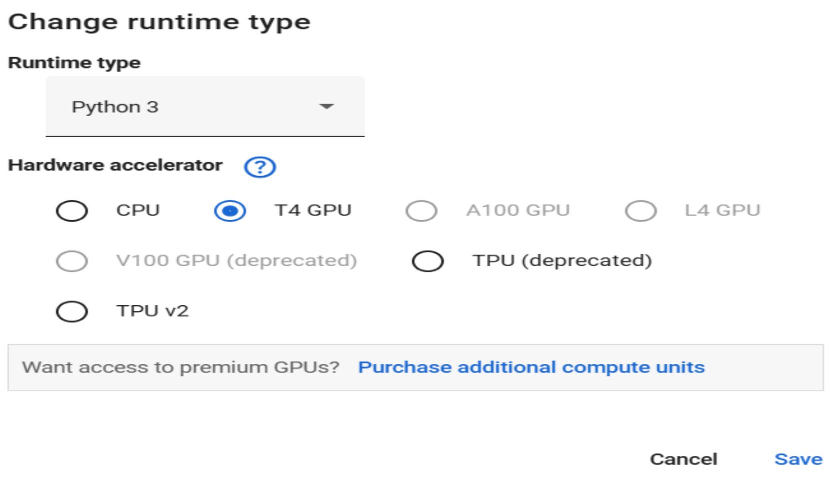

Training YOLOv8 Object Detection on Roboflow Dataset using Google Colab
Ensure GPU acceleration is enabled
Activate Nvidia System Management Interface (SMI)
Create project home page
Install YOLOv8 from Ultralytics repository
Get dataset from Roboflow
Perform training on the dataset
Validate the model using best performing weights

**Figure 11:** Training Process flowchart

1- **Enabling GPU Acceleration:** To leverage the computational power of the GPU for faster training, we enabled GPU acceleration. Deep learning models like YOLOv8 benefit significantly from GPU acceleration due to their high computational demands. As shown in the figure 12, we switched the runtime type from CPU to T4 GPU.


Change runtime type

Runtime type: Python 3

Hardware accelerator: CPU [✓ T4 GPU] A100 GPU L4 GPU V100 GPU (deprecated) TPU (deprecated) TPU v2

Want access to premium GPUs? Purchase additional compute units
Cancel
Save

**Figure 12:** Enabling GPU Acceleration

2- **Activating Nvidia System Management Interface (SMI):** We used the !nvidia-smi command to activate Nvidia SMI, which allows us to monitor GPU usage. This is crucial for ensuring that the GPU resources are being effectively utilized during training, and also for troubleshooting any potential issues related to GPU availability.

```
!nvidia-smi
```

3- **Creating a Project Home Page for Organization:** We created a project home page to maintain organization and structure within our Colab environment. This helps us keep track of files, directories, and other resources related to our project.

```
import os
HOME = os.getcwd()
print(HOME)
```

4- **Installing YOLOv8 from the Ultralytics Repository:** We installed YOLOv8 from the Ultralytics repository[27] using the command !pip install ultralytics==8.0.196. This ensures that we have the latest version of YOLOv8 installed and ready for use in our training pipeline.

```
!pip install ultralytics==8.0.196
from IPython import display
display.clear_output()
import ultralytics
ultralytics.checks()
```

5- **Obtaining the Dataset from Roboflow:** We obtained the dataset named 'Smart Traffic Management System' from Roboflow using the roboflow Python package. This package provides convenient access to datasets stored on the Roboflow platform, allowing us to seamlessly integrate them into our training workflow.

```
!mkdir {HOME}/datasets
%cd {HOME}/datasets
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="sdxIR5efXXEFYsED0Fde")
project = rf.workspace("noor-7vdrq").project("smart-traffic-management-system-xne3g")
version = project.version(5)
dataset = version.download("yolov8")
```

6- **Performing Training with 51 Epochs:** We trained the YOLOv8 model on the dataset for 51 epochs using the yolo command-line interface. This command specifies various parameters such as the model to use (yolov8s.pt), the dataset location, image size (imgsz), and the number of epochs to train for. Training for multiple epochs allows the model to learn from the data and improve its performance over time.

```
%cd {HOME}
!yolo task=detect mode=train model=yolov8s.pt data={dataset.location}/data.yaml epochs=51
imgsz=800 plots=True
```

7- **Validating the Model:** Finally, we validated the trained model using the best performing weights saved in the 'best.pt' file. This step is essential for assessing the model's performance on unseen data and ensuring that it generalizes well to new samples. The validation process helps us evaluate metrics such as accuracy, precision, and recall, providing insights into the model's overall effectiveness.

```
%cd {HOME}
!yolo task=detect mode=val model={HOME}/runs/detect/train3/weights/best.pt
data={dataset.location}/data.yaml
```

### 4.3.1 Selecting optimal number of epochs

Selecting the ideal number of training epochs for YOLO is essential for balancing model performance, generalization, and training efficiency. After experimenting with 20, 50, and 100 epochs, it was determined that 51 epochs yielded the best results. An epoch is a single pass through the entire dataset during training. Key performance metrics include Train Box Loss and Train Class Loss, which indicate how well the predicted bounding boxes match the ground truth and the accuracy of object classification within those boxes, respectively, with lower values representing better performance. Precision (B) and Recall (B) measure the ratio of true positive predictions to all positive predictions and all actual positives, respectively, where higher values indicate fewer false positives and false negatives. The mAP50 (Mean Average Precision at IoU threshold 0.5) evaluates detection accuracy at a specific overlap threshold, while mAP50-95 averages mAP across multiple IoU thresholds from 0.5 to 0.95, providing a comprehensive assessment of model performance. Validation metrics such as Val Box Loss and Val Class Loss assess how well the predicted bounding boxes and object classification perform on unseen data, with lower values indicating better alignment and accuracy.

**Why Not 20 Epochs?**

1- **Underfitting:** Training for only 20 epochs may not provide sufficient time for the model to adequately learn the patterns in the data, leading to underfitting. In this case, the model fails to capture the underlying structure properly, resulting in poor performance on both the training and validation sets.

2- **Suboptimal Performance:** Training for too few epochs might result in lower performance metrics, as the model may not have fully converged to a suitable state.

**Table 2:** The result for epochs from 15 to 20

| epoch | train/box_loss | train/cls_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) | val/box_loss | val/cls_loss |
|---|---|---|---|---|---|---|---|---|
| 15 | 0.76549 | 0.66932 | 0.7351 | 0.59435 | 0.71774 | 0.54925 | 0.74595 | 0.61981 |
| 16 | 0.76044 | 0.66262 | 0.64441 | 0.7178 | 0.70079 | 0.54942 | 0.72244 | 0.6012 |
| 17 | 0.73646 | 0.6458 | 0.69858 | 0.59111 | 0.73836 | 0.57062 | 0.68935 | 0.58946 |
| 18 | 0.73064 | 0.63104 | 0.61403 | 0.68975 | 0.65017 | 0.51096 | 0.68113 | 0.58507 |
| 19 | 0.71752 | 0.63314 | 0.63857 | 0.6825 | 0.69145 | 0.55365 | 0.67527 | 0.61093 |
| 20 | 0.70328 | 0.61531 | 0.63062 | 0.68655 | 0.71281 | 0.57301 | 0.66945 | 0.56765 |

**Why Not 100 Epochs?**

1- **Overfitting**: Training for 100 epochs often leads to overfitting, where the model performs exceptionally well on the training data but poorly on unseen validation data. This happens when the model memorizes specific details of the training set instead of learning general patterns.

2- **Decreased Performance**: In the previous analysis, metrics like mAP50 and mAP50-95 declined when the model was trained for 100 epochs, indicating overfitting and degraded performance.

**Table 3:** The result of epochs from 95 to 100

| epoch | train/box_loss | train/cls_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) | val/box_loss | val/cls_loss |
|---|---|---|---|---|---|---|---|---|
| 95 | 0.6093 | 0.4668 | 0.64129 | 0.7192 | 0.7329 | 0.5977 | 0.63652 | 0.5778 |
| 96 | 0.60633 | 0.4644 | 0.67884 | 0.613 | 0.7013 | 0.5776 | 0.63768 | 0.5888 |
| 97 | 0.60142 | 0.4557 | 0.68488 | 0.6323 | 0.7368 | 0.6007 | 0.63856 | 0.5875 |
| 98 | 0.60091 | 0.4522 | 0.76659 | 0.5575 | 0.7416 | 0.6147 | 0.63738 | 0.5934 |
| 99 | 0.60068 | 0.4516 | 0.62539 | 0.6839 | 0.721 | 0.5957 | 0.6345 | 0.5855 |
| 100 | 0.60116 | 0.4504 | 0.6769 | 0.6187 | 0.7235 | 0.5931 | 0.63883 | 0.5941 |

**Why 51 Epochs?**

1- **Balanced Performance:** The performance metrics (precision, recall, mAP) after 50 epochs were stable and strong. By extending training to 51 epochs, we are allowing the model just a bit more time to converge and potentially reach a slightly more optimal state.

2- **Exploration:** Since the metrics indicated good performance around the 50-epoch mark, training for 51 epochs allows as to see if a small increase in training time improves or maintains performance.

3- **Generalization:** By training for 51 epochs, we are giving the model sufficient training time while avoiding overfitting, which often occurs when training for too many epochs.

4- **Optimal Choice:** Training the model for 51 epochs offered a good balance between precision, recall, and mAP metrics while also achieving the lowest validation losses, generally indicating better performance compared to nearby values.

**Table 4:** The result for epochs from 48 to 54

| epoch | train/box_loss | train/cls_loss | metrics/precision(B) | metrics/recall(B) | metrics/mAP50(B) | metrics/mAP50-95(B) | val/box_loss | val/cls_loss |
|---|---|---|---|---|---|---|---|---|
| 48 | 0.71299 | 0.6183 | 0.7927 | 0.652 | 0.7807 | 0.6122 | 0.66443 | 0.5691 |
| 49 | 0.70507 | 0.6117 | 0.7961 | 0.551 | 0.7364 | 0.5781 | 0.66304 | 0.5685 |
| 50 | 0.69437 | 0.6043 | 0.71877 | 0.6251 | 0.7804 | 0.625 | 0.66296 | 0.5682 |
| 51 | 0.71773 | 0.6128 | 0.82843 | 0.6321 | 0.7764 | 0.6236 | 0.6568 | 0.5514 |
| 52 | 0.70865 | 0.6106 | 0.78324 | 0.6182 | 0.7593 | 0.5934 | 0.66163 | 0.5594 |
| 53 | 0.69483 | 0.6066 | 0.73851 | 0.6098 | 0.7092 | 0.5542 | 0.65521 | 0.558 |
| 54 | 0.69928 | 0.6071 | 0.68323 | 0.6195 | 0.7481 | 0.5715 | 0.65384 | 0.5708 |

### 4.3.2 Result of training

From the training process, we have a folder that includes **`args.yaml`** and many figures (matrices and curves). The **`args.yaml`**[32] file contains all the parameters, and here, we highlight the most important ones along with the significant figures.

- **task:** Specifies the task to be performed. In this case, it's set to "detect", indicating that the model will be trained for object detection.
- **mode:** Specifies the mode of operation. It's set to "train", indicating that the model will be trained.
- **model:** Specifies the base YOLOv8 model file ("yolov8s.pt") to be used for training.
- **data:** Specifies the path to the data configuration file ("data.yaml") containing information about the dataset.
- **epochs:** Specifies the number of epochs for training. It's set to 1, indicating a single epoch will be performed.
- **batch:** Specifies the batch size for training. It's set to 16, indicating that each batch will contain 16 images.
- **imgsz:** Specifies the input image size for the model. It's set to 640x640 pixels.
- **save:** Specifies whether to save the trained model weights. It's set to true, indicating that the weights will be saved.
- **device:** Specifies the device to be used for training. It's set to "cpu", indicating that the CPU will be used for training.
- **workers:** Specifies the number of worker processes for data loading. It's set to 8, indicating that 8 worker processes will be used.
- **optimizer:** Specifies the optimizer to be used for training. It's set to SGD (Stochastic Gradient Descent).
- **lr0:** Specifies the initial learning rate for the optimizer. It's set to 0.01.
- **momentum:** Specifies the momentum for SGD optimizer. It's set to 0.937.
- **weight_decay:** Specifies the weight decay (L2 penalty) for regularization. It's set to 0.0005.

- **warmup_epochs:** Specifies the number of warm-up epochs for learning rate warm-up. It's set to 3.0.
- **box, cls, dfl, pose, kobj:** Parameters for loss function weighting.
- **augment:** Specifies whether data augmentation will be applied during training. It's set to false, indicating no augmentation.
- **plots:** Specifies whether to generate plots during training. It's set to true.
- **save_dir:** Specifies the directory where the trained model and results will be saved

Here are the training process matrix and curves.

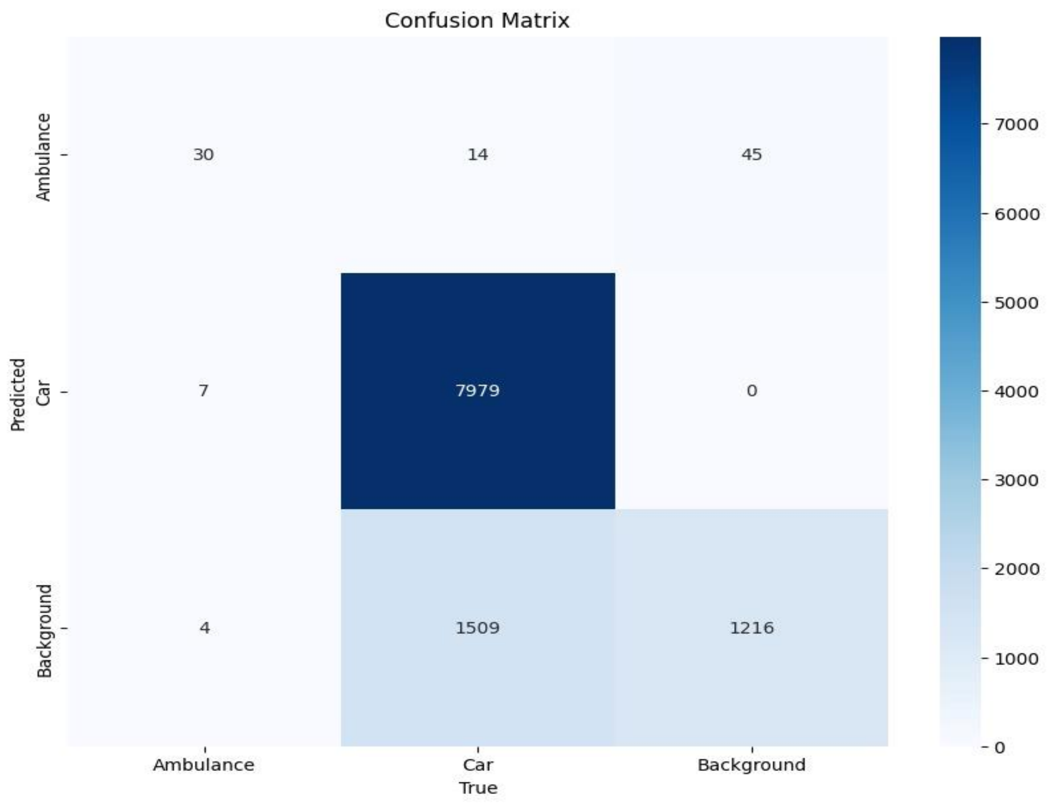


**Figure 13:** Confusion matrix, This matrix shows highlights the model's strong accuracy in predicting cars but reveals challenges distinguishing ambulances due to misclassifications.

This figure shows a confusion matrix, a tool used to assess the performance of a classification model. It summarizes the counts of correct and incorrect predictions made by the model in comparison to the actual outcomes.

**Breakdown of the Matrix**

1- **Ambulance (True) Predictions**:
   - Correctly predicted as Ambulance: 30 instances
   - Incorrectly predicted as Car: 7 instances
   - Incorrectly predicted as Background: 4 instances

2- **Car (True) Predictions**:
   - Correctly predicted as Car: 7979 instances
   - Incorrectly predicted as Ambulance: 14 instances
   - Incorrectly predicted as Background: 1509 instances

3- **Background (True) Predictions**:
   - Correctly predicted as Background: 1216 instances
   - Incorrectly predicted as Ambulance: 45 instances
   - Incorrectly predicted as Car: 0 instances

**Sum each column:**

- Sum of Ambulance column: 30 + 7 + 4 = 41
- Sum of Car column: 14 + 7979 + 1509 = 9502
- Sum of Background column: 45 + 0 + 1216 = 1261

**Interpretation**

- **High Accuracy:**
  - The model is highly accurate in predicting "Car" instances, with 7979 correct predictions.
  - The model is also relatively accurate in predicting "Background" instances, with 1216 correct predictions.
- **Misclassification Issues:**
  - Car Instances: A notable number of car instances (1509) are incorrectly predicted as "Background."

**Conclusion**

- **Strengths:**
  - The model is very effective in correctly identifying "Car" instances.
  - It performs well with "Background" instances, though there's room for improvement.

- **Weaknesses:**
  - The model struggles to accurately classify "Ambulance" instances, often confusing them with "Car" or "Background."
  - There is a significant number of "Background" instances being misclassified as "Car."

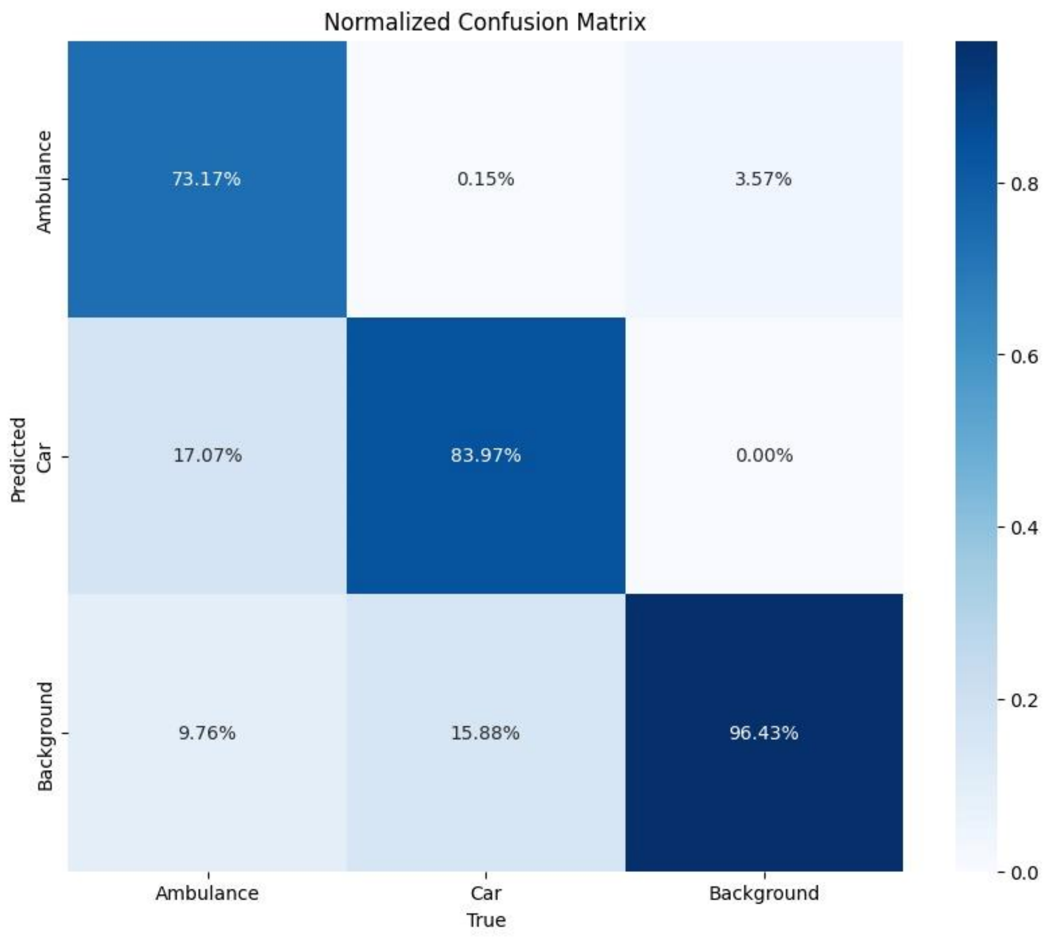


**Figure 14:** Normalized confusion matrix, This matrix shows high accuracy in classifying background (96.43%) and cars (83.97%), but struggles with ambulances, misclassifying 26.83% as cars or background.

This figure shows a normalized confusion matrix that assesses the performance of a classification model by displaying the percentage of correct and incorrect predictions, normalized by the total number of instances in each class. The values are taken from the previous confusion matrix. The classes represented are "Ambulance," "Car," and "Background," with the percentage calculated using the equation “normalized value = cell value / column sum.”

**Breakdown of the Matrix**

1- **Ambulance (True) Predictions:**
   - Correctly predicted: 73.17% (30 / 41 = 0.7317)
   - Misclassified as Car: 0.17% (7 / 41 = 0.1707)
   - Misclassified as Background: 9.76% (4 / 41 = 0.0976)

2- **Car (True) Predictions:**
   - Correctly predicted: 83.97% (7979 / 9502 = 0.8397)
   - Misclassified as Ambulance: 0.15% (14 / 9502 = 0.0015)
   - Misclassified as Background: 15.88% (1509 / 9502 = 0.1588)

3- **Background (True) Predictions:**
   - Correctly predicted: 96.43% (1216 / 1261 = 0.9643)
   - Misclassified as Ambulance: 3.57% (45 / 1261 = 0.0357)
   - Misclassified as Car: 0.00% (0 / 1261 = 0.00)

**Interpretation**

- **High Accuracy:**
  - The model shows high accuracy in predicting "Car" (83.97%) and "Background" (96.43%).

**Conclusion**

- The model performs well in distinguishing "Car" and "Background" but has difficulty distinguishing "Ambulance" from the other classes.

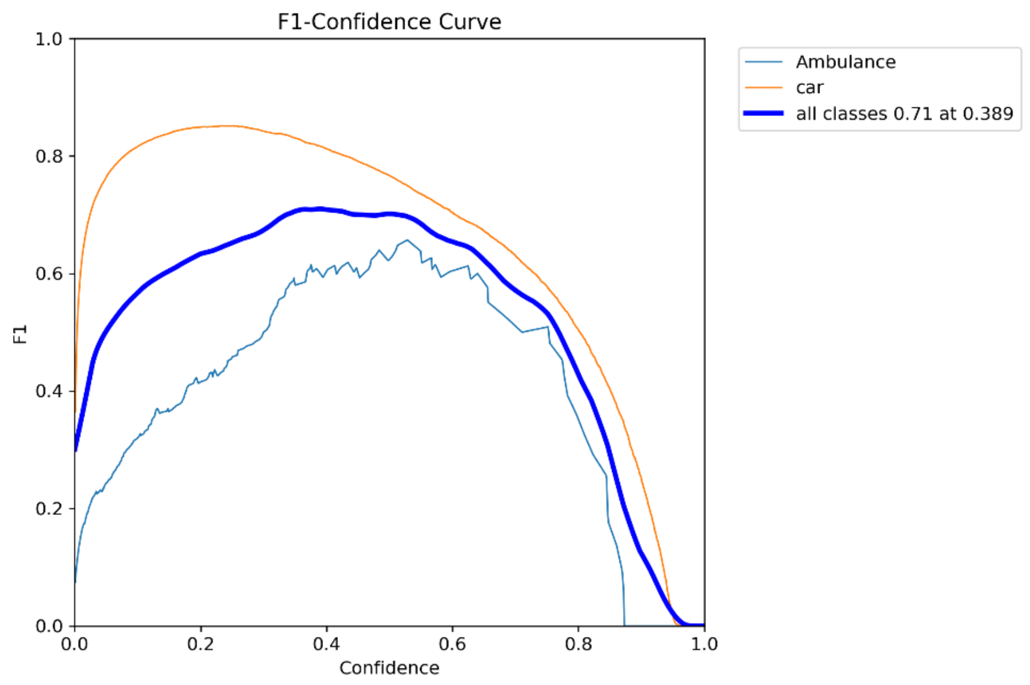


**Figure 15:** F1-Confidence Curve, The highest F1 score for all classes is 0.71, achieved at a confidence threshold of 0.389

This figure shows F1-Confidence Curve, which evaluates the performance of a classification model by plotting the F1 score against confidence levels for different classes. The F1 score is a measure of a test's accuracy, considering both precision and recall.

**Breakdown of the Curves:**

**1- Ambulance (Light Blue Line):**

- The F1 score for the "Ambulance" class starts relatively low, increases to a peak around a mid-confidence level, and then declines as confidence increases.
- This indicates that the model has variable performance in predicting "Ambulance" instances, with a peak performance at a specific confidence threshold.

2- **Car (Orange Line):**

- The F1 score for the "Car" class is consistently higher than for the "Ambulance" class across all confidence levels.
- The curve reaches its peak at a lower confidence threshold and maintains a high F1 score over a broader range of confidence levels.
- This shows that the model is very effective in predicting "Car" instances, maintaining high accuracy across different confidence thresholds.

3- **All Classes (Thick Blue Line):**

- The overall F1 score for all classes combined peaks at 0.71 when the confidence level is 0.389.
- This combined score curve is a weighted representation of the individual class performances, indicating the model's general effectiveness.

**Interpretation**

- **High Performance for "Car" Class:**
  - The "Car" class exhibits the highest F1 score, indicating strong model performance in predicting this class.
  - The broad peak suggests that the model's predictions for cars are reliable across a range of confidence levels.

- **Variable Performance for "Ambulance" Class:**
  - The "Ambulance" class shows a significant peak but at a specific confidence threshold, indicating the model's performance is more variable and less reliable for this class.

- **Overall Model Performance:**
  - The combined F1 score curve shows that the model's optimal performance, considering all classes, is at a confidence level of 0.389 with an F1 score of 0.71.
  - This suggests a general threshold where the model balances precision and recall most effectively across all classes.

## Conclusion

- **Strengths:**
  - The model performs exceptionally well for the "Car" class, maintaining high accuracy across a wide confidence range.
  - The overall model performance is robust at a confidence level of 0.389.

- **Weaknesses:**
  - The model's performance for the "Ambulance" class is less reliable, with significant variability and a lower peak F1 score.
  - Adjustments may be needed to improve predictions for the "Ambulance" class to achieve more consistent performance across all classes.

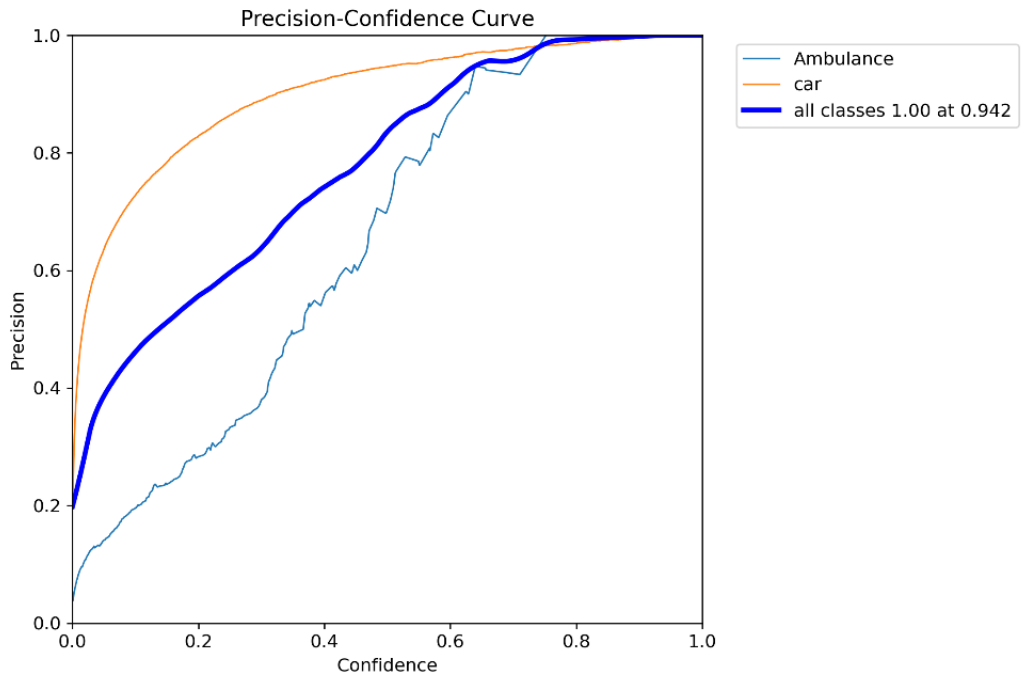


**Figure 16:** Precision-Confidence curve ,The highest precision for all classes is 1.00, achieved at a confidence threshold of 0.942.

This figure shows Precision-Confidence Curve. The Precision-Confidence Curve evaluates the performance of a classification model by plotting precision against confidence levels for different classes. Precision is the ratio of true positive predictions to the total number of positive predictions, reflecting the model's accuracy in identifying true positives.

**Breakdown of the Curves:**

1- **Ambulance (Light Blue Line):**
   - Initial Precision: The precision for the "Ambulance" class starts relatively low.
   - Peak Performance: Precision increases with confidence, peaking around a higher confidence level.
   - Decline: As confidence continues to increase, the precision decreases slightly.
   - Interpretation: This indicates that the model's performance in predicting "Ambulance" instances improves up to a certain confidence threshold but declines with very high confidence levels.

2- **Car (Orange Line):**
   - Consistently High Precision: The precision for the "Car" class is consistently higher than for the "Ambulance" class across all confidence levels.
   - Broad Peak: The curve reaches its peak at a lower confidence threshold and maintains a high precision over a broad range of confidence levels.
   - Interpretation: This shows that the model is very effective in predicting "Car" instances, maintaining high accuracy across different confidence thresholds.

3- **All Classes (Thick Blue Line):**
   - Overall Precision: The overall precision for all classes combined peaks near a confidence level of 0.942.
   - Interpretation: This combined precision curve represents the weighted performance of the model across all classes, indicating its general effectiveness.

**Interpretation:**

- **High Performance for "Car" Class**:
    - The "Car" class exhibits the highest precision, indicating strong model performance in predicting this class.
    - The broad peak suggests that the model's predictions for cars are reliable across a range of confidence levels.

- **Variable Performance for "Ambulance" Class:**
    - The "Ambulance" class shows a significant peak but at a specific confidence threshold, indicating the model's performance is more variable and less reliable for this class.

- **Overall Model Performance:**
    - The combined precision curve shows that the model's optimal performance, considering all classes, peaks at a confidence level of 0.942.
    - This suggests a general threshold where the model balances precision most effectively across all classes.

**Conclusion:**

- **Strengths**:
    - The model performs exceptionally well for the "Car" class, maintaining high accuracy across a wide confidence range.
    - The overall model performance is robust at a confidence level of 0.942.

- **Weaknesses**
    - The model's performance for the "Ambulance" class is less reliable, with significant variability and a lower peak precision.

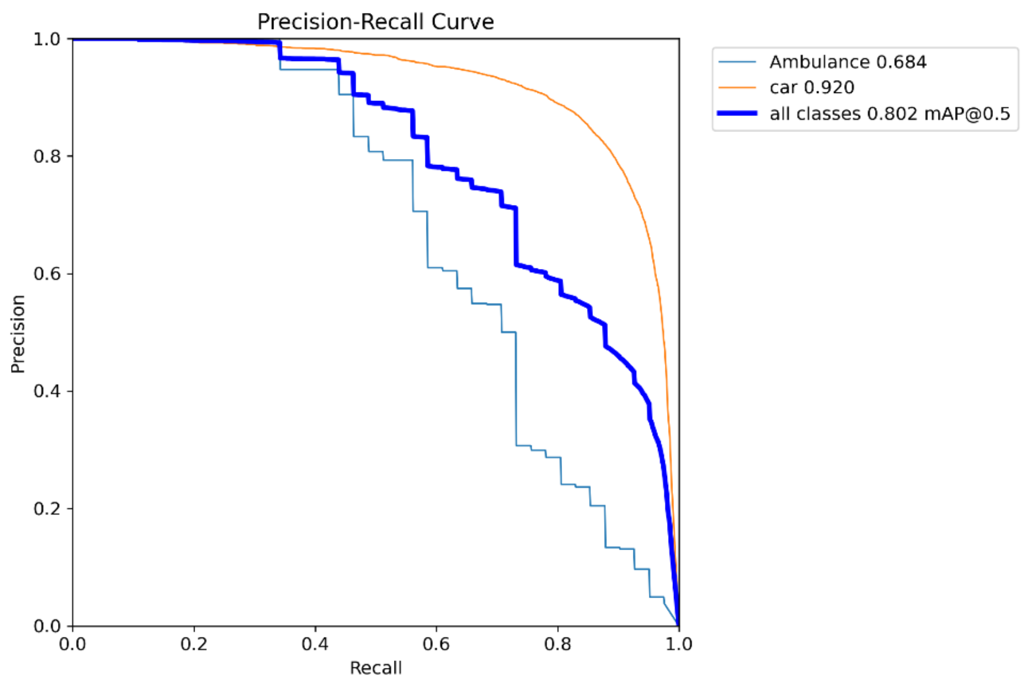


**Figure 17:** Precision-Recall Curve, The mean Average Precision (mAP) for all classes at an IoU threshold of 0.5 is 0.802.

This figure shows Precision-Recall Curve, which evaluates the performance of a classification model by plotting precision against recall for different classes. Precision is the ratio of true positive predictions to the total number of positive predictions, while recall is the ratio of true positive predictions to the total number of actual positives. The Precision-Recall Curve is particularly useful when dealing with imbalanced datasets.

**Breakdown of the Curves:**

1- **Ambulance (Light Blue Line):**

- Precision-Recall Relationship: The curve for the "Ambulance" class starts with high precision and moderate recall.
- Decline: As recall increases, precision decreases, indicating a trade-off between precision and recall for this class.
- Area Under Curve (AUC): The area under the curve (AUC) for the "Ambulance" class is 0.684, indicating moderate performance.

**2- Car (Orange Line):**

- Precision-Recall Relationship: The curve for the "Car" class shows high precision even as recall increases.
- Sustained Performance: The curve maintains a high level of precision over a wide range of recall values, suggesting the model's strong performance in predicting "Car" instances.
- AUC: The area under the curve (AUC) for the "Car" class is 0.920, indicating excellent performance.

**3- All Classes (Thick Blue Line):**

- Combined Performance: The combined precision-recall curve for all classes is represented by the thick blue line.
- Mean Average Precision (mAP): The mean average precision (mAP) for all classes at a threshold of 0.5 is 0.802.
- Interpretation: This combined score provides a weighted average of the precision-recall performance across all classes, reflecting the model's general effectiveness.

**Interpretation:**

- **High Performance for "Car" Class:**
  - The "Car" class exhibits the highest precision and recall values, indicating strong model performance in predicting this class.
  - The curve shows a broad range where both precision and recall are high, suggesting reliable predictions for the "Car" class.
- **Moderate Performance for "Ambulance" Class:**
  - The "Ambulance" class shows lower precision and recall values, indicating more variability and less reliability in predictions.
  - The significant drop in precision as recall increases suggests that while the model can identify more true positives, it also produces more false positives.

- **Overall Model Performance:**
  - The combined precision-recall curve shows that the model's mean average precision (mAP) is 0.802 at a threshold of 0.5.
  - This suggests that the model balances precision and recall effectively across all classes, with generally good performance.

**Conclusion:**

- **Strengths:**
  - The model performs exceptionally well for the "Car" class, maintaining high precision and recall across a wide range.
  - The overall model performance is strong, with a mean average precision (mAP) of 0.802.
- **Weaknesses:**
  - The model's performance for the "Ambulance" class is less reliable, with a noticeable trade-off between precision and recall.
  - Improvements may be needed to enhance the model's predictions for the "Ambulance" class, aiming for more consistent performance across all classes.

**Final Thoughts:**

Understanding the Precision-Recall Curve helps in assessing the trade-offs between precision and recall for different classes, particularly in scenarios with imbalanced data. This evaluation can guide model adjustments to improve performance where needed.

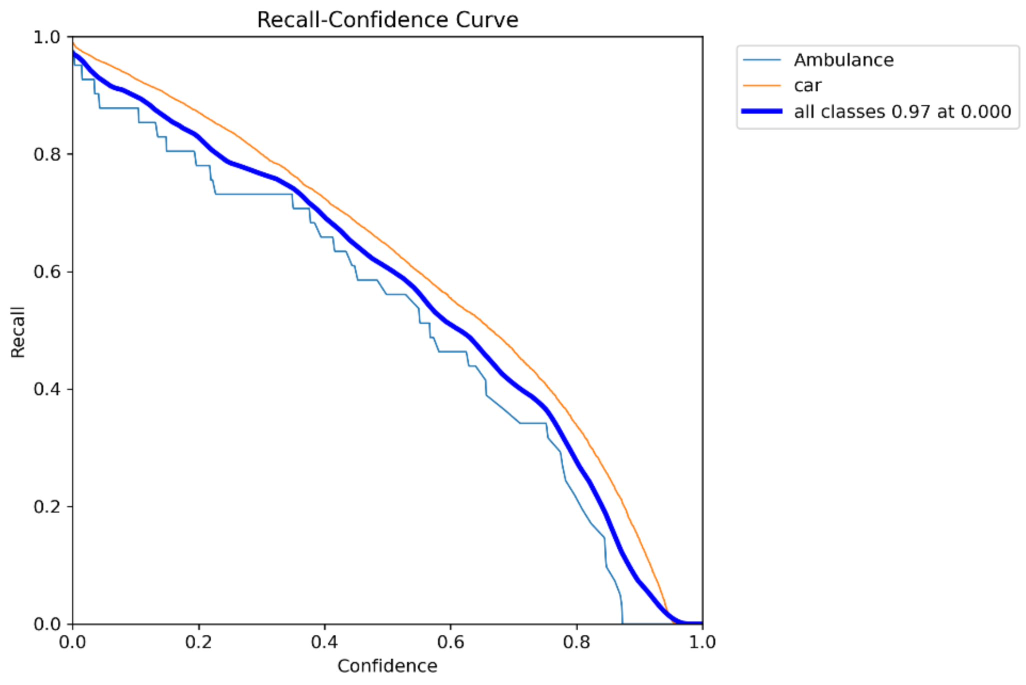


**Figure 18:** Recall-Confidence Curve, The 'car' class achieving higher recall at lower confidence than the 'Ambulance' class, with an overall AUC of 0.97.

This figure shows Recall-Confidence Curve, which evaluates the performance of a classification model by plotting recall against confidence levels for different classes. Recall is the ratio of true positive predictions to the total number of actual positives, reflecting the model's ability to identify all relevant instances.

**Breakdown of the Curves:**

1- **Ambulance (Light Blue Line):**

- High Initial Recall: The recall for the "Ambulance" class starts high when the confidence level is low.
- Decline with Confidence: As the confidence level increases, the recall decreases, indicating that fewer true positives are identified at higher confidence thresholds.
- Interpretation: This suggests that the model is more inclusive at lower confidence levels but becomes more conservative (and thus less inclusive) as confidence increases.

2- **Car (Orange Line):**

- Consistently High Recall: The recall for the "Car" class starts high and declines more gradually compared to the "Ambulance" class.
- Sustained Performance: The curve maintains higher recall values over a broader range of confidence levels.
- Interpretation: This indicates that the model is effective in identifying "Car" instances even at higher confidence levels, demonstrating more robust performance for this class.

3- **All Classes (Thick Blue Line):**

- Overall Recall: The combined recall for all classes starts high and decreases steadily as confidence increases.
- Max Recall: The overall recall is 0.97 at a confidence level of 0.000.
- Interpretation: This combined recall curve provides a weighted average of recall across all classes, reflecting the model's general ability to identify true positives across all classes.

**Interpretation:**

- **High Performance for "Car" Class:**
  - The "Car" class exhibits the highest recall values across a wide range of confidence levels, indicating strong model performance in identifying this class.
  - The gradual decline in recall suggests that the model maintains high sensitivity for the "Car" class, even at higher confidence levels.
- **Variable Performance for "Ambulance" Class:**
  - The "Ambulance" class shows a sharper decline in recall as confidence increases, indicating that the model's ability to identify "Ambulance" instances diminishes more quickly with higher confidence thresholds.
  - This suggests that the model is less reliable in identifying all "Ambulance" instances, particularly at higher confidence levels.

- **Overall Model Performance:**
    - The combined recall curve shows that the model's highest recall is achieved at the lowest confidence level.
    - This suggests that, while the model is highly sensitive overall, its ability to maintain high recall diminishes with increasing confidence thresholds.

**Conclusion:**

- **Strengths:**
    - The model performs exceptionally well for the "Car" class, maintaining high recall across a wide range of confidence levels.
    - The overall model performance shows high initial recall, indicating good sensitivity in identifying true positives when confidence thresholds are low.

- **Weaknesses:**
    - The model's performance for the "Ambulance" class is less reliable, with recall declining sharply at higher confidence levels.
    - Improvements may be needed to enhance the model's ability to maintain high recall for the "Ambulance" class at higher confidence thresholds, aiming for more consistent performance across all classes.

**Final Thoughts:**

The Recall-Confidence Curve provides insights into how the model's ability to identify true positives changes with varying confidence levels. Understanding this relationship helps in assessing the model's sensitivity and can guide adjustments to improve performance, especially for classes where recall declines significantly at higher confidence levels.

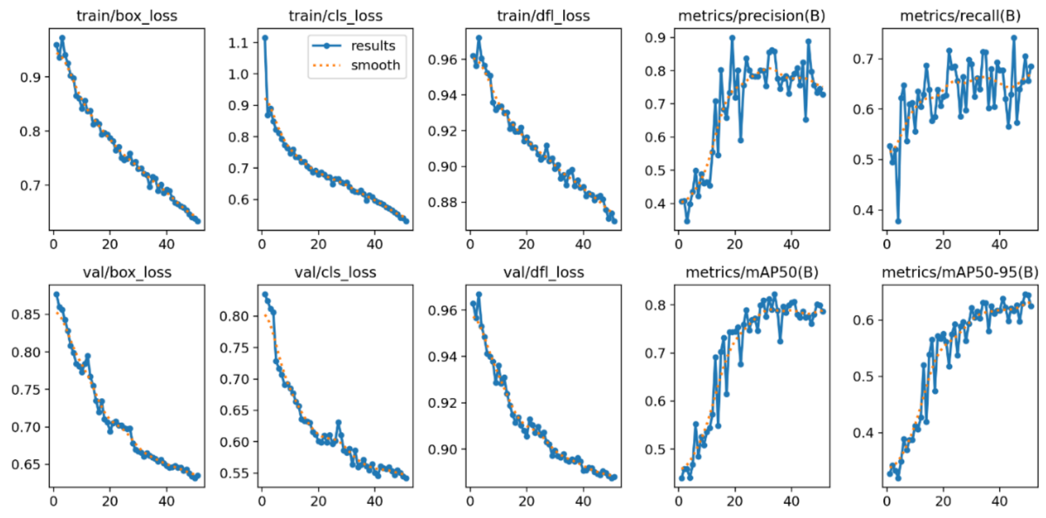


**Figure 19:** Results of training, The training and validation loss curves consistently decrease over epochs, while the precision, recall, and mAP metrics show significant improvement, indicating effective model training.

This figure shows displays various performance metrics and loss values over training epochs for a model, likely used in an object detection context. Here's an explanation of each plot in the image:

1- **Train/Box Loss:**
   - This plot shows the box loss during training over epochs.
   - The loss decreases steadily, indicating that the model is improving its localization accuracy for bounding boxes as training progresses.

2- **Train/Cls Loss:**
   - This plot shows the classification loss during training over epochs.
   - The loss decreases consistently, showing that the model is getting better at classifying objects correctly over time.

3- **Train/DFL Loss:**
   - This plot shows the distribution focal loss (DFL) during training.
   - A steady decline suggests that the model's confidence in its predictions is improving.

4- **Metrics/Precision(B):**
   - This plot shows the precision of the model on the validation set, likely for a specific class or a set of classes, over epochs.

- Precision fluctuates but generally trends upwards, indicating that the model becomes more precise in its predictions with more training.

5- **Metrics/Recall(B):**

- This plot shows the recall of the model on the validation set over epochs.
- Recall improves over time, meaning the model is getting better at detecting relevant objects.

6- **Val/Box Loss:**

- This plot shows the box loss on the validation set over epochs.
- The loss decreases, showing that the model's performance on unseen data is improving for bounding box predictions.

7- **Val/Cls Loss:**

- This plot shows the classification loss on the validation set over epochs.
- Similar to training loss, it decreases, suggesting better classification accuracy on validation data.

8- **Val/DFL Loss:**

- This plot shows the distribution focal loss on the validation set over epochs.
- The loss decreases steadily, indicating that the model's confidence and calibration on validation data are improving.

9- **Metrics/mAP50(B):**

- This plot shows the mean Average Precision (mAP) at 50% Intersection over Union (IoU) on the validation set.
- The mAP50 increases over time, indicating overall improvement in the model’s detection performance.

10- **Metrics/mAP50-95(B):**

- This plot shows the mean Average Precision over a range of IoU thresholds (from 50% to 95%) on the validation set.
- An increasing trend suggests that the model is getting better at detecting objects across varying levels of overlap between predicted and true bounding boxes.

**Interpretation:**

- **High Performance:**
  - The model shows a steady decrease in both training and validation losses, along with an increase in precision, recall, and mAP metrics. This suggests effective learning and generalization to new data.

- **Improvements:**
  - Continued training and potentially fine-tuning can further improve the model’s performance, particularly in areas where metrics still fluctuate.

**Conclusion:**

- **Strengths:**
  - The model is showing significant improvements in both localization and classification tasks, with increasing precision, recall, and mAP scores.
- **Weaknesses:**
  - Areas with more fluctuation in metrics might need additional focus, potentially indicating the need for further data augmentation or hyperparameter tuning.

Overall, the model’s performance is progressing well, showing effective learning from the training data and improving its predictions on the validation set.

### 4.3.3 Conclusion from training

In conclusion, the training and validation losses for both box, class, and object detection (train/box_loss, train/cls_loss, train/dfl_loss, val/box_loss, val/cls_loss, val/dfl_loss) have shown consistent decreases throughout the training process. This trend signifies that the model is learning and improving its predictive accuracy over time without overfitting to the training data. Moreover, precision metrics (metrics/precision(B)) have demonstrated a steady increase, indicating the model's proficiency in correctly identifying true positives while minimizing false positive errors. Concurrently, recall metrics (metrics/recall(B)) have also shown improvement, illustrating the model's ability to identify a greater proportion of actual positives while reducing false negative errors. Additionally, both mean Average Precision (mAP50) and mean Average Precision across IoU thresholds of 50% to 95% (mAP50-95) have exhibited upward trends, suggesting enhanced overall performance in accurately detecting objects across varying sizes and shapes. These collective findings affirm that the model is effectively learning, generalizing well to validation data, and achieving high accuracy in object detection tasks.

## 4.4 Counting and making decision

After completing the model training process, we are just one step away from observing the results of our project, we proceeded to the stage of calculating the number of cars and ambulance in the input images and classifying them according to their density. This classification allows for comparison to determine which road should be opened. These steps are carried out using Python code, with explanations provided for each part. We exclusively utilize the Python programming language for all code within our project. Below, we elaborate on this decision and outline several reasons for choosing Python for our model:

Python stands out for its simplicity and readability, making it a popular choice across a wide array of domains, from machine learning to web development and automation. **The selection of Python for our model is grounded in several compelling reasons:**

- **Ease of Use:** Python's intuitive syntax and coherent structure streamline the coding process, empowering developers to write and comprehend code with ease.
- **Abundance of Libraries:** Python boasts an extensive library ecosystem, offering ready-made solutions for various tasks and accelerating development by providing access to a wealth of pre-built functionalities.
- **Strong Community Support:** Python enjoys robust community backing, with a large and active developer base contributing to its continual enhancement. This vibrant community ensures ample online resources, including comprehensive documentation, tutorials, and forums, fostering seamless troubleshooting and knowledge exchange throughout the development lifecycle.

In summary, Python's user-friendly nature, extensive library support, and thriving community make it the ideal programming language for implementing a smart traffic management system.

The following steps enable our code to function effectively:

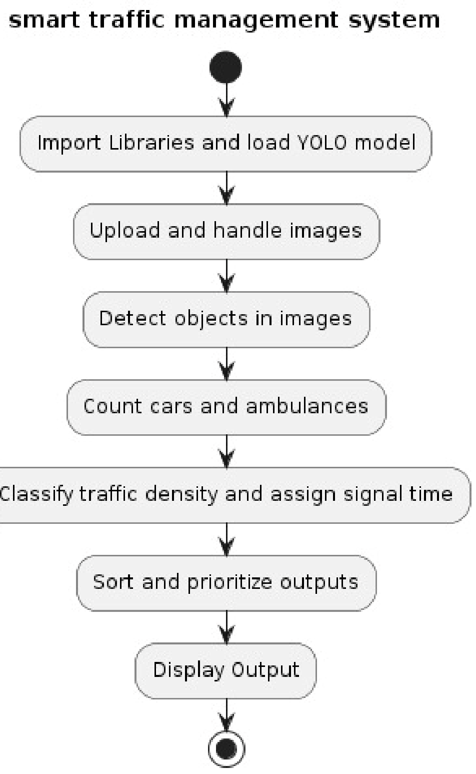

smart traffic management system
Import Libraries and load YOLO model
Upload and handle images
Detect objects in images
Count cars and ambulances
Classify traffic density and assign signal time
Sort and prioritize outputs
Display Output

**Figure 20:** Code flowchart

As shown as figure 20 the functionality of our code are:

1- **Import Libraries and load YOLO model:** This step involves importing necessary libraries and loading a YOLO (You Only Look Once) model, which is commonly used for object detection tasks due to its speed and accuracy.

2- **Upload and handle images:** Images are uploaded from traffic cameras or other sources and prepared for processing. This may involve resizing, normalization, or other preprocessing steps.

3- **Detect objects in images:** Utilizing the YOLO model, objects such as cars and ambulances are detected within the images. This step identifies the presence and location of these objects.

4- **Count cars and ambulances:** Once objects are detected, the system counts the number of cars and ambulances present in the scene. This count is crucial for traffic management decisions.

5- **Classify traffic density and assign signal time:** Based on the counts and possibly other factors (like time of day, weather conditions), the traffic density is classified (e.g., low, medium, high). Signal times at intersections may be dynamically adjusted to optimize traffic flow.

6- **Sort and prioritize outputs:** Outputs from the system are sorted and prioritized, potentially based on urgency (e.g., emergency vehicle prioritization) or traffic flow optimization strategies.

7- **Display Output:** Finally, the processed information and decisions are displayed as output. This could be in the form of graphical displays for traffic controllers or public information systems.

### 4.4.1 Source code

Here, we will display the complete code for our project along with detailed explanations: "The code has been uploaded to GitHub[33]."

```
import cv2
from ultralytics import YOLO
import base64
from IPython.display import display, HTML
import ipywidgets as widgets

# Load a pretrained YOLO model
model = YOLO('/content/runs/detect/train/weights/best.pt')

# Define the pairs of lines that will open simultaneously
line_pairs = {
   '1.1': '3.1',
   '1.2': '3.2',
   '2.1': '4.1',
   '2.2': '4.2'
}

# Create a file upload widget
upload_widgets = []
for i in range(8):
    upload_widgets.append(widgets.FileUpload(accept='.jpg,.jpeg,.png',
multiple=False))

upload_button = widgets.Button(description="Process Images")

# Display the file upload widgets
display(widgets.VBox(upload_widgets))
display(upload_button)

# Function to handle file uploads
def handle_uploads(change):
    image_paths = {}
    counter = 0

    for i in range(1, 5):
        for j in range(1, 3):
            key = f"{i}.{j}"
            uploaded_file = list(upload_widgets[counter].value.values())[0]
            image_path = f'/tmp/{uploaded_file["metadata"]["name"]}'

            # Save the uploaded image to a temporary path
            with open(image_path, 'wb') as f:
                f.write(uploaded_file['content'])

            image_paths[key] = image_path
            counter += 1

    process_images(image_paths)
# Attach the function to the button click event
upload_button.on_click(handle_uploads)
```

**1- Import Libraries:**

- cv2: This is OpenCV, a library for computer vision tasks, including reading and manipulating images.
- ultralytics.YOLO: Imports the YOLO object detection model from the Ultralytics library.
- base64: Module for encoding and decoding binary data to ASCII strings.
- display and HTML from IPython.display: Utilities for displaying HTML content.
- ipywidgets: Library for creating interactive widgets in Jupyter notebooks.

2- **Load a Pre-trained YOLO Model:**

This model is trained to detect objects (cars, ambulances) in images, which is crucial for analyzing traffic at various intersections.

3- **Define Pairs of Traffic Lines:**

Maps pairs of traffic lines that should have synchronized signals.

4- **File Upload Widget:**

sets up the user interface for uploading images. It utilizes the **ipywidgets. FileUpload** class to create file upload widgets, each capable of accepting image files with the extensions .jpg, .jpeg, and .png. These widgets are then stored in a list named upload_widgets. Additionally, a button widget labeled "Process Images" (named upload_button) is created to initiate the image processing once the user has uploaded the desired images.

5- **Displaying Widgets and Handling File Uploads:**

Widgets for file uploads and an upload button are displayed using the **display** function. Upon clicking the upload button, the **handle_uploads** function is triggered. This function iterates over the file upload widgets, saving uploaded image files to temporary paths. It then constructs a dictionary called **image_paths**, where keys represent line numbers and values represent the paths to the uploaded images. Afterward, the **process_images** function is invoked with the **image_paths** dictionary as input for further processing.

```
def analyze_traffic(image_path):
    img = cv2.imread(image_path)
    car_count, ambulance_count, image_with_counts = analyze_traffic_image(img, 
model)

    # Save the modified image
    modified_image_path = image_path.replace('.jpg', '_bbox.jpg')
    cv2.imwrite(modified_image_path, image_with_counts)

    # Determine the density (car count)
    if car_count == 0:
        density = 'no_traffic'
    elif 1 <= car_count < 20:
        density = 'low'
    elif 20 <= car_count <= 35:
        density = 'medium'
    elif 35 < car_count <= 100:
        density = 'high'

    return density, car_count, ambulance_count, modified_image_path
```

1- **Function Input and Image Loading:**

   The function accepts a single parameter **image_path**, representing the path to the image file to be analyzed.

2- **Image Processing:**

   The function begins by utilizing OpenCV's **cv2.imread** function to read the image located at the specified **image_path**. This image is then passed as input to another function named **analyze_traffic_image**, along with the YOLO model, for further analysis. Within the **analyze_traffic_image** function, various aspects of the image are examined, including the detection of cars and ambulance vehicles. Upon completion of the analysis, the **analyze_traffic** function retrieves pertinent information from **analyze_traffic_image**, such as the number of cars (**car_count**), the count of ambulance vehicles (**ambulance_count**), and an annotated version of the image with counts (**image_with_counts**). These details are essential for understanding the traffic situation and are subsequently used for determining the traffic density and further processing.

3- **Saving Modified Image:**

   After analysis, the function forms a modified image path by replacing '.jpg' with '_bbox.jpg'. Then, it writes the annotated image (**image_with_counts**) to this path using **cv2.imwrite**.

4- **Determine Traffic Density:**

   Based on the number of cars counted, the function categorizes traffic density into one of several levels (no traffic, low, medium, high). This categorization is crucial for deciding how long traffic signals should remain green.

5- **Return Traffic Information:**

   The function returns the determined traffic density and the count of cars and ambulances and the path to the modified image. This output informs subsequent traffic management decisions, particularly adjusting signal timings to enhance flow and prioritize emergency responses.

```python
def analyze_traffic_image(image, model):
    results = model.predict(image, agnostic_nms=True)[0]
    car_count = 0
    ambulance_count = 0
    merged_boxes = []

    for pred in results:
        cls = int(pred.boxes.cls)
        x1, y1, x2, y2 = map(int, pred.boxes.xyxy[0])

        center_x = (x1 + x2) // 2
        center_y = (y1 + y2) // 2

        merged = False
        for box in merged_boxes:
            x1m, y1m, x2m, y2m = box
            center_xm = (x1m + x2m) // 2
            center_ym = (y1m + y2m) // 2

            if abs(center_x - center_xm) < 30 and abs(center_y - center_ym) < 30:
                x1 = min(x1, x1m)
                y1 = min(y1, y1m)
                x2 = max(x2, x2m)
                y2 = max(y2, y2m)
                merged_boxes.remove(box)
                merged = True
                break

        if not merged:
            merged_boxes.append((x1, y1, x2, y2))

        if cls == 1:
            color = (0, 255, 0)  # Green for cars
            car_count += 1
        elif cls == 0:
            color = (0, 0, 255)  # Red for emergency vehicles
            ambulance_count += 1
        cv2.rectangle(image, (x1, y1), (x2, y2), color, 2)
        label = 'Car' if cls == 1 else 'Emergency'
        cv2.putText(image, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9,
color, 2)

    return car_count, ambulance_count, image
```

1- **Function Definition:**

The function accepts two parameters: **image**, representing the input image for analysis, and **model**, which is a YOLO object detection model.

2- **Object Detection:**

The function employs the YOLO model (**model**) to predict objects in the input image (**image**) using the **predict** method. Subsequently, it retrieves the detection results (**results**), which encompass crucial details like bounding box coordinates and class predictions for the detected objects. This step forms the foundation for further analysis and annotation of the image based on the identified objects.

3- **Annotation and Counting:**

The function initializes counters for car count (**car_count**) and ambulance count (**ambulance_count**) to zero. It then iterates through each prediction (pred) in the results, extracting the class prediction (**cls**) and bounding box coordinates (**x1, y1, x2, y2**). Using these coordinates, it determines the center coordinates of the bounding box (**center_x, center_y**). To avoid duplicate counts, it checks for overlapping bounding boxes and merges them if necessary. Based on the class prediction (1 for cars, 0 for emergency vehicles), the counters are updated accordingly. Finally, the function draws rectangles around detected objects and adds labels to the image for visualization purposes.

4- **Returning Results:**

Returns the counts of cars (**car_count**) and ambulance vehicles (**ambulance_count**), along with the annotated image (**image**) with bounding boxes and labels added.

```
# Function to process images
def process_images(image_paths):
    # Define time allocation for different car counts
    time_allocation = {'no_traffic': 0, 'low': 5, 'medium': 15, 'high': 30}

    # Emergency time allocations adjusted to consider the highest applicable car
count
    emergency_time_allocation = {'no_traffic': 5, 'low': 10, 'medium': 15,
'high': 30}

    # Update traffic data and apply correct durations
    traffic_data = {}
    for line, path in image_paths.items():
        density, car_count, ambulance_count, modified_image_path = 
analyze_traffic(path)
        traffic_data[line] = {
            'density': density,
            'car_count': car_count,
            'emergency_vehicles': ambulance_count,
            'time': time_allocation[density],
            'image_path': modified_image_path
        }
```

1- **Define Time Allocation for Traffic Conditions and Emergencies:**
   sets the duration for which traffic lights should remain green based on assessed traffic density with categories like 'no_traffic', 'low', 'medium', and 'high'. It also establishes emergency time allocations, ensuring that even in minimal traffic situations, emergency vehicles are given priority with sufficient green time to navigate intersections quickly.
2- **Updating Traffic Data:**
   The function starts by initializing an empty dictionary named **traffic_data**, intended to store traffic-related details for each line. It then iterates through each key-value pair in the **image_paths** dictionary. For each line, the function invokes the **analyze_traffic** function to analyze the associated image, extracting information like density, car count, ambulance count, and the path to the modified image. Subsequently, this data is incorporated into the **traffic_data** dictionary, encompassing density, car count, emergency vehicle count, allocated time, and the path to the modified image. This organized collection of information facilitates efficient management and decision-making in traffic control systems.

```
emergency_output = []
    regular_output = []
    for line, paired_line in line_pairs.items():
        line_info = traffic_data[line]
        paired_line_info = traffic_data[paired_line]

        if line_info['emergency_vehicles'] > 0 or 
paired_line_info['emergency_vehicles'] > 0:
            priority_line = line if line_info['emergency_vehicles'] > 
paired_line_info['emergency_vehicles'] else paired_line
```

```
            priority_density = line_info['density'] if
line_info['emergency_vehicles'] > paired_line_info['emergency_vehicles'] else
paired_line_info['density']
            emergency_duration = emergency_time_allocation[priority_density]
            emergency_output.append({'text': f"Emergency priority: Lines {line}
and {paired_line} open for {emergency_duration} seconds due to emergency vehicle
in {priority_line} with {priority_density} density.", 'priority': 'Emergency',
'density_key': priority_density})
        else:
            # Determine which line has the higher density and use it to dictate
opening times
            if line_info['density'] == 'high' or paired_line_info['density'] ==
'high':
                maximum_density = 'high'
            elif line_info['density'] == 'medium' or paired_line_info['density']
== 'medium':
                maximum_density = 'medium'
            elif line_info['density'] == 'low' or paired_line_info['density'] ==
'low':
                maximum_density = 'low'
            else:
                maximum_density = 'no_traffic'

            maximum_density_line = line if line_info['density'] ==
maximum_density else paired_line
            regular_duration = time_allocation[maximum_density]
            regular_output.append({
                'text': f"Regular traffic: Lines {line} and {paired_line} open
for {regular_duration} seconds based on {maximum_density} density in line
{maximum_density_line}.",
                'priority': 'Regular',
                'density_key': maximum_density
            })

    emergency_density_sort_order = {
        'no_traffic': 0,  # highest priority in emergencies
        'low': 1,
        'medium': 2,
        'high': 3
    }

    # Sort emergency output by priority first, and within the emergency priority,
use the emergency_density_sort_order for further sorting
    emergency_output.sort(key=lambda x: (x['priority'],
emergency_density_sort_order.get(x['density_key'], float('inf'))))
```

```
    # Print emergency output
    for result in emergency_output:
        print(result['text'])

    # Sort regular output by priority first, and within the regular priority, use
the reverse of density order for further sorting
    regular_output.sort(key=lambda x: (-
emergency_density_sort_order.get(x['density_key'], float('inf'))))

    # Print regular output
    for result in regular_output:
        print(result['text'])
```

1- **Initialization:**

Two empty lists, **emergency_output** and **regular_output**, are initialized to hold traffic management decisions for emergency and regular scenarios, respectively.

2- **Decision Making:**

- For each line and its paired line in **line_pairs**, the code evaluates if any emergency vehicles are present on either line. If so, it prioritizes the line with more emergency vehicles and calculates the emergency duration based on the highest density among the two lines. Information about these emergency situations is appended to **emergency_output**.
- If no emergency vehicles are detected on either line, the code determines the line with the highest density and allocates regular duration accordingly. This information is appended to **regular_output**.

3- **Sorting:**

- **emergency_output** is sorted first by priority and then by emergency density order.
- **regular_output** is sorted by priority and density order, but in reverse to prioritize higher density.

```
# Function to convert image to base64 encoding
    def image_to_base64(image_path):
        with open(image_path, "rb") as img_file:
            return base64.b64encode(img_file.read()).decode('utf-8')
```

```
# Define the output HTML code
output_html = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Traffic Management System</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 0;
            padding: 20px;
            background-color: #f2f2f2;
        }
        .container {
            max-width:700px;
            margin: 0 auto;
            background-color: gray;
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        }
        h1 {
            text-align: center;
        }
        .output {
            margin-top: 20px;
        }
        .output-item {
            padding: 20px;
            margin-bottom: 20px;
            border-radius: 5px;
            background-color: #f9f9f9;
            box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
            display: flex;
            align-items: center;
            justify-content: center; /* Center align items horizontally */
            flex-wrap: wrap; /* Wrap items to next line if necessary */
        }
        .output-item img {
            max-width: 200px; /* Adjust the width of the images */
            border-radius: 5px;
            margin: 10px; /* Add margin around images */
            }
            .emergency {
                color: red; /* Set emergency text color to red */
            }
            .image-path {
                font-size: 14px; /* Adjust font size */
                color: gray; /* Set color for path numbers */
                text-align: center; /* Center align path numbers */
            }
            .regular-text {
                color: black; /* Set regular text color to black */
            }
            .details {
                text-align: center;
                margin-top: 5px;
            }
        </style>
    </head>
    <body>
        <div class="container">
            <h1>Smart Traffic Management System</h1>
            <div class="output">
    """

    # Append emergency output with images and text results
    for result in emergency_output:
        split_text = result["text"].split()
        line_numbers = split_text[3]  # Extracting line numbers from the text
        paired_line_numbers = split_text[5]  # Adjusted index to extract paired
line numbers
        line_info = traffic_data[line_numbers]
        paired_line_info = traffic_data[paired_line_numbers]
        output_html += f'<div class="output-item emergency">'
        output_html += f'<div>{result["text"]}</div>'
        output_html += f'<img
src="data:image/jpeg;base64,{image_to_base64(line_info["image_path"])}"
alt="Image for Line {line_numbers}">'
        output_html += f'<div class="details">Line {line_numbers}:
{line_info["car_count"]} cars, {line_info["emergency_vehicles"]} emergency
vehicles</div>'
        output_html += f'<img
src="data:image/jpeg;base64,{image_to_base64(paired_line_info["image_path"])}"
alt="Image for Line {paired_line_numbers}">'
```

```
        output_html += f'<div class="details">Line {paired_line_numbers}:
{paired_line_info["car_count"]} cars, {paired_line_info["emergency_vehicles"]}
emergency vehicles</div>'
        output_html += f'</div>'

    # Append regular output with images and text results
    for result in regular_output:
        split_text = result["text"].split()
        line_numbers = split_text[3]  # Extracting line numbers from the text
        paired_line_numbers = split_text[5]  # Adjusted index to extract paired
line numbers
        line_info = traffic_data[line_numbers]
        paired_line_info = traffic_data[paired_line_numbers]
        output_html += f'<div class="output-item regular-text">'
        output_html += f'<div>{result["text"]}</div>'
        output_html += f'<img
src="data:image/jpeg;base64,{image_to_base64(line_info["image_path"])}"
alt="Image for Line {line_numbers}">'
        output_html += f'<div class="details">Line {line_numbers}:
{line_info["car_count"]} cars, {line_info["emergency_vehicles"]} emergency
vehicles</div>'
        output_html += f'<img
src="data:image/jpeg;base64,{image_to_base64(paired_line_info["image_path"])}"
alt="Image for Line {paired_line_numbers}">'
        output_html += f'<div class="details">Line {paired_line_numbers}:
{paired_line_info["car_count"]} cars, {paired_line_info["emergency_vehicles"]}
emergency vehicles</div>'
        output_html += f'</div>'

    output_html += """
            </div>
        </div>
    </body>
    </html>
    """

    # Display the generated HTML
    display(HTML(output_html))
```

1- **Image to Base64 Encoding Function:**

Defines a function image_to_base64 to convert images to base64 encoding. This function is used to embed images directly into the HTML output.

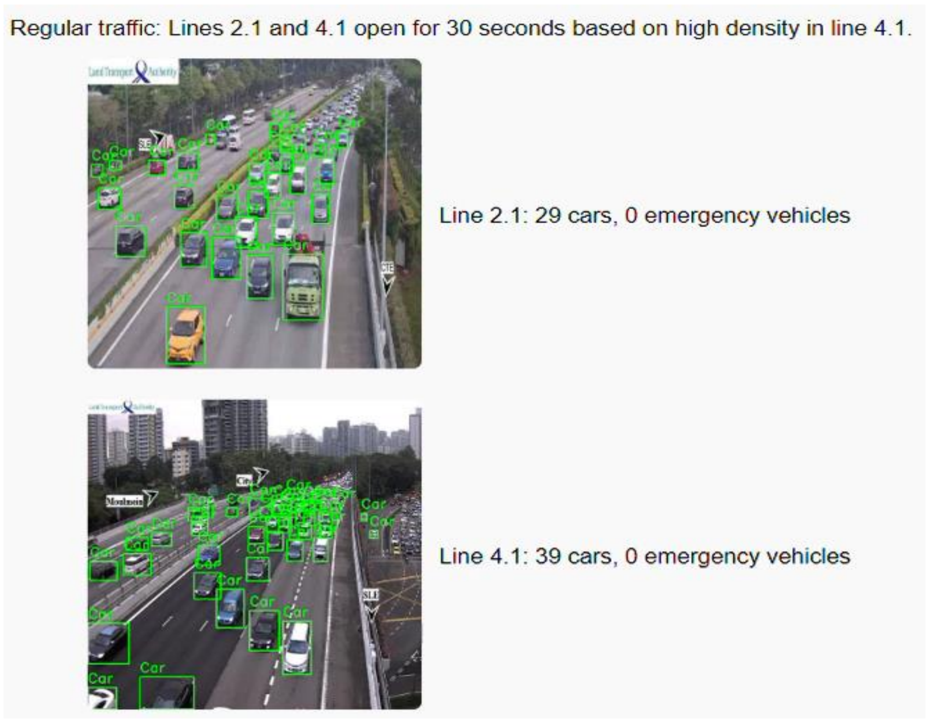
2- **HTML Output Generation:**

This code segment initializes an HTML string (**output_html**) to organize the output content, incorporating metadata, styling, and a title ("Smart Traffic Management System"). It establishes a container with a header for the title and another container to display traffic management outputs. The code iterates through emergency and regular outputs, embedding pertinent images using base64 encoding. Each output item comprises decision text, images for the line and its paired line, and details regarding car count and emergency vehicles. The HTML content is dynamically structured according to traffic management decisions.

3- **Displaying HTML:**

Uses IPython's **display** function to render the generated HTML output within the notebook environment.

## 4.4.2 Final output

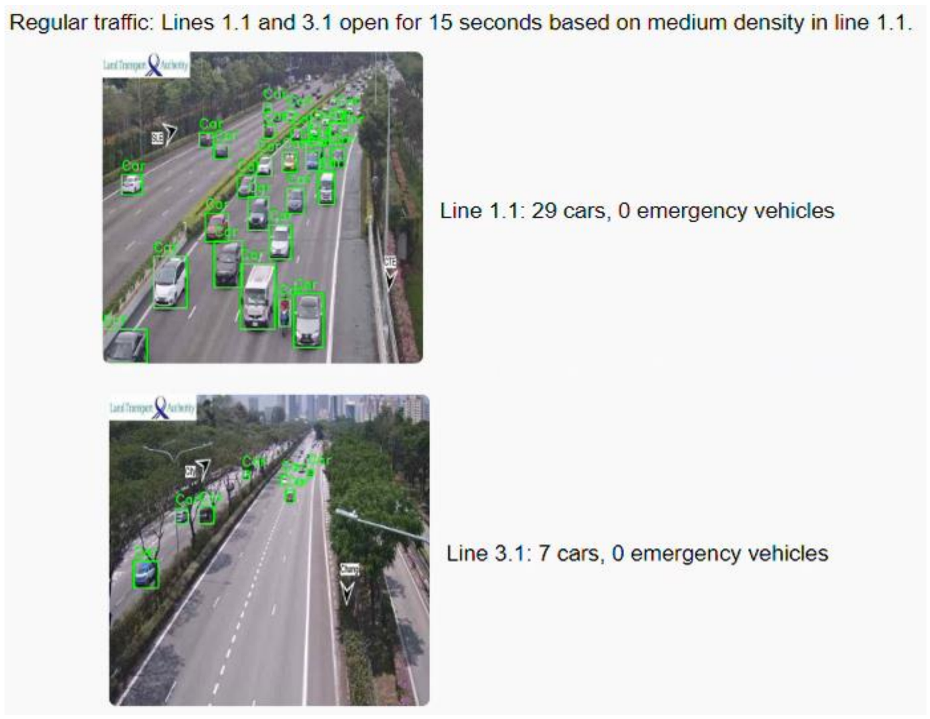


Regular traffic: Lines 2.1 and 4.1 open for 30 seconds based on high density in line 4.1.

Line 2.1: 29 cars, 0 emergency vehicles

Line 4.1: 39 cars, 0 emergency vehicles

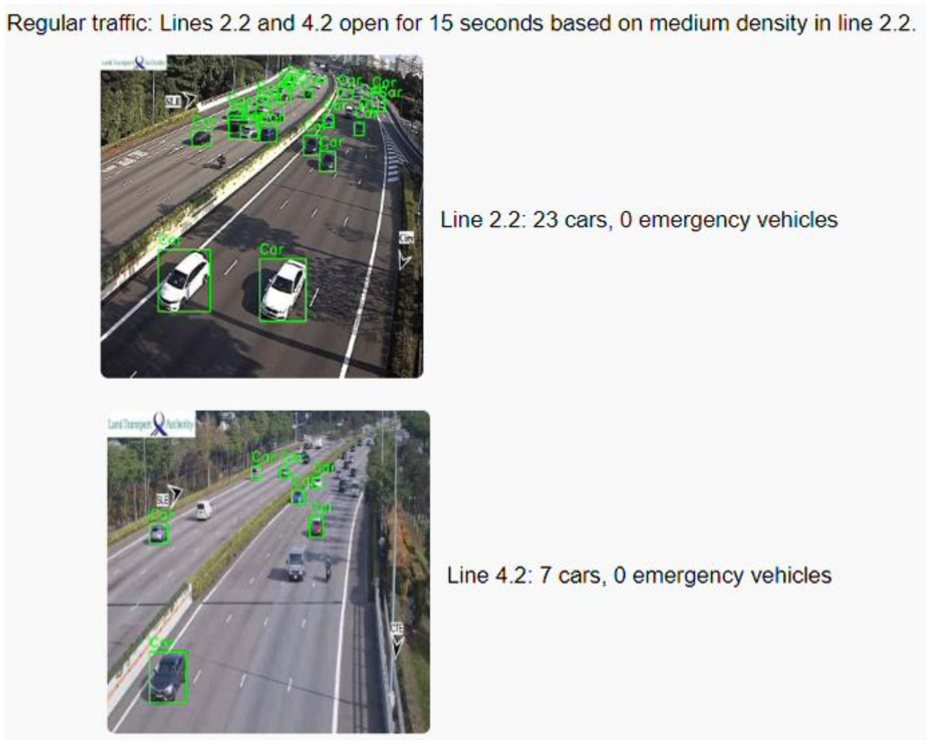
**Figure 21**


Regular traffic: Lines 1.1 and 3.1 open for 15 seconds based on medium density in line 1.1.

Line 1.1: 29 cars, 0 emergency vehicles

Line 3.1: 7 cars, 0 emergency vehicles

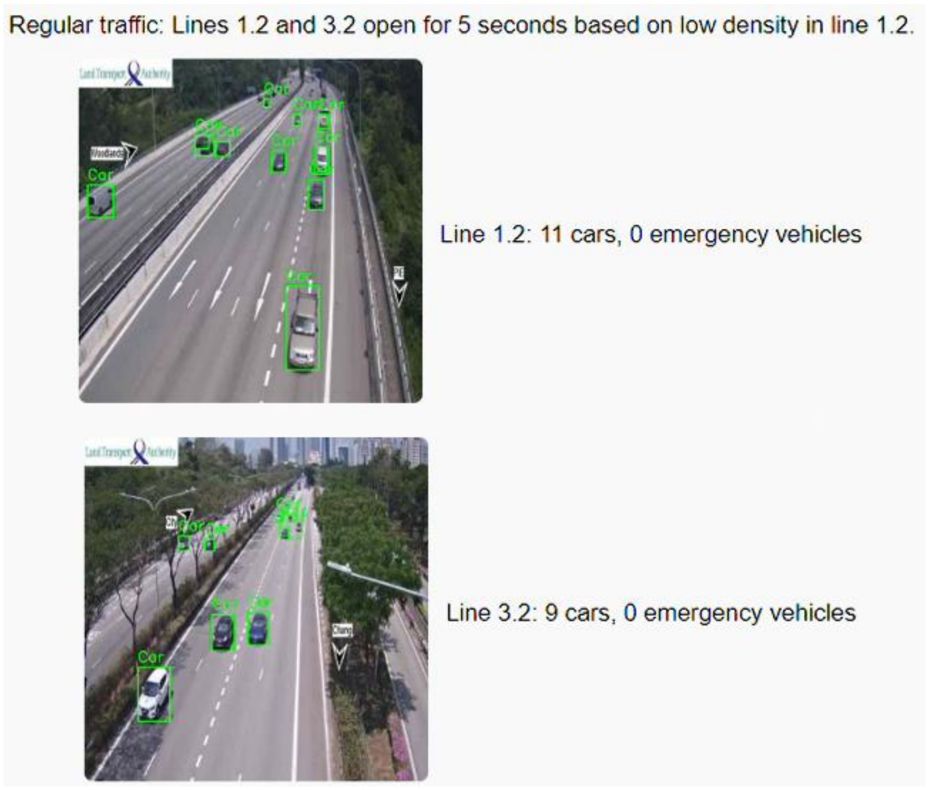
Figure 22


Regular traffic: Lines 2.2 and 4.2 open for 15 seconds based on medium density in line 2.2.

Line 2.2: 23 cars, 0 emergency vehicles

Line 4.2: 7 cars, 0 emergency vehicles

Figure 23

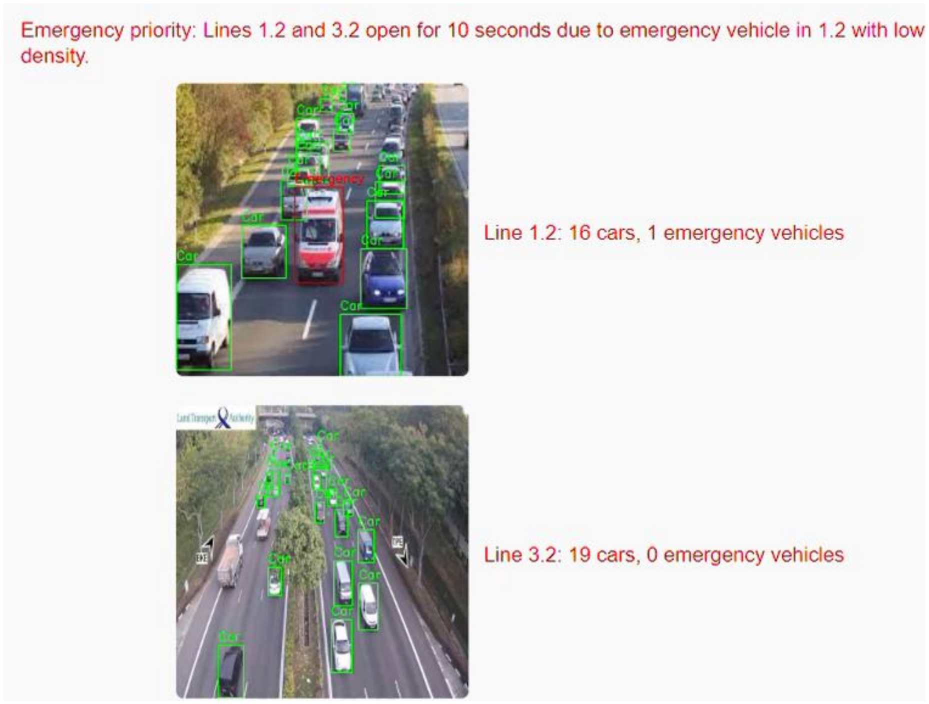


Regular traffic: Lines 1.2 and 3.2 open for 5 seconds based on low density in line 1.2.

Line 1.2: 11 cars, 0 emergency vehicles

Line 3.2: 9 cars, 0 emergency vehicles

Figure 24



Emergency priority: Lines 1.2 and 3.2 open for 10 seconds due to emergency vehicle in 1.2 with low density.

Line 1.2: 16 cars, 1 emergency vehicles

Line 3.2: 19 cars, 0 emergency vehicles

Figure 25

Figure 21 – Figure 25: HTML Output from Our Model

# CHAPTER 6

# CHALLENGES AND FUTURE WORK

## 6.1 Challenges

Challenges Encountered Dauring System Development:

1- Data Collection: Obtaining suitable datasets and images posed challenges due to their specificity. We required street and top-view images depicting emergency scenarios with realistic traffic conditions around high-traffic areas.

2- Literature Review and Model Selection: Conducting an extensive literature review was essential to select the optimal vehicle detection model. We compared various models such as YOLO and its variants, as well as Faster-RCNN, focusing on their accuracy and speed of detection. Understanding the underlying principles of each model was crucial in making an informed choice.

3- Finding an Effective Auto Annotation Platform: Locating a reliable platform capable of performing auto annotation accurately was another hurdle. We needed a solution that could automate annotation tasks to streamline the process, while ensuring precision through manual verification when necessary.

4- Transitioning from Visual Studio to Google Colab: Initially, we started our work in Visual Studio, which provided a familiar environment for development. However, as our project progressed, we found that training our models on Google Colab offered significant advantages. We had to adapt our workflow to seamlessly transition between the two platforms. This involved optimizing our code and processes for Colab's cloud-based environment, ensuring efficient training and smooth collaboration among team members.

## 6.2 Future work

1- Meet Real-Time Traffic Monitoring: Integrate the system with live traffic cameras to analyze traffic conditions in real-time. This would involve continuously processing video feeds to detect vehicles and emergency vehicles, and dynamically adjusting traffic light timings accordingly.

2- Integration with Smart City Infrastructure: Explore integration opportunities with other smart city infrastructure components, such as public transportation systems, to create a more cohesive and efficient urban environment.

3- Multi-Intersection Coordination: Extend the system to manage traffic lights at multiple intersections simultaneously. Develop algorithms for coordinated signal timing to improve traffic efficiency across a network of intersections.

4- Pedestrian Detection and Safety Features: Enhance the system to detect and prioritize pedestrian crossings. Implement safety features such as alert systems for drivers when pedestrians are detected near intersections.

5- Robustness and Reliability: Continuously work on improving the robustness and reliability of the system, ensuring smooth operation under various conditions such as in night, adverse weather, hardware failures, and unexpected events.

6- Implement Multi-Class Classification: Extend the classification capabilities of the system to accurately classify detected vehicles into their respective categories, such as trucks, buses, motorcycles, etc.

7- Improving Emergency Vehicle Detection: Emphasizing the identification of emergency vehicles, including police cars, fire trucks, and ambulances, as a primary focus within the detection process.

# CONCLUSION

After tireless efforts and careful planning, our project has come to fruition, offering a solution that's finely tuned to the intricacies of urban life. It's not just about managing traffic; it's about making our streets safer and our cities more efficient.

By blending advanced technologies with creative problem-solving, we've built a system that can swiftly adapt to changing traffic patterns and respond to emergencies with precision. It's like having a traffic manager who's always one step ahead, ensuring smooth sailing even in the busiest of streets.

Our project isn't just about optimizing traffic lights; it's about improving the way we move through our cities. It's about using data and innovation to create a transportation system that works for everyone, making our urban spaces more livable and enjoyable for all.

# CHAPTER 7

# REFFERENCES

[1] Afrin, S., & Yodo, N. (2020). Traffic congestion: causes and effects. MDPI.

[2] Smith, J., & Jones, A. (Year). "The Environmental Impact of Traffic Crisis: Effects on Air Pollution and Climate Change." Journal of Environmental Studies, Volume (Issue), Page numbers.

[3] Springer. (2020). Addressing safety concerns in traffic congestion management. MDPI.

[4] JSTOR. (2021). The role of traffic lights in managing traffic flow. JSTOR.

[5] JSTOR. (2021). Challenges and solutions in addressing traffic congestion in Palestine. JSTOR.

[6] Azzedine Boukerche and Zhijun Hou. 2021. Object Detection Using Deep Learning Methods in Traffic Scenarios. ACM Comput. Surv. 54, 2, Article 30 (March 2021).

[7] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. 2016. Region-based convolutional networks for accurate object detection and segmentation. IEEE Trans. Pattern Anal. Mach. Intell. 38, 1 (2016), 142–158.

[8] Ross Girshick. 2015. Fast R-CNN. In Proceedings of the IEEE International Conference on Computer Vision. 1440–1448.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In Advances in Neural Information Processing Systems. MIT Press, 91–99.

[10] A. Voulodimos, N. Doulamis, A. Doulamis, E. Protopapadakis, Deep learning for computer vision: a brief review. Comput. Intell. Neurosci. 1–13 (2018).

[11] Jasper R. R. Uijlings, Koen E. A. Van De Sande, Theo Gevers, and Arnold W. M. Smeulders. 2013. Selective search for object recognition. Int. J. Comput. Vision 104, 2 (2013), 154–171.

[12] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” Computer Science, 2014.

[13] Yongzheng Xu, Guizhen Yu, Yunpeng Wang, Xinkai Wu, and Yalong Ma. 2017. Car detection from low-altitude UAV imagery with the faster R-CNN. J. Adv. Transport. 2017 (2017).

[14] O. Barnich and M. van Droogenbroeck, “ViBe: a universal background subtraction algorithm for video sequences,” IEEE Transactions on Image Processing, vol. 20, no. 6, pp. 1709–1724, 2011.

[15] A. C. Shastry and R. A. Schowengerdt, “Airborne video registration and traffic-flow parameter estimation,” IEEE Transactions on Intelligent Transportation Systems, vol. 6, no. 4, pp. 391–405, 2005.

[16] P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, pp. 511–518, December 2001.

[17] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR ’05), vol. 1, pp. 886–893, June 2005.

[18] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: towards real-time object detection with region proposal networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, no. 6, pp. 1137–1149, 2017.

[19] Md Khorshed Alam, Asif Ahmed, Rania Salih, Abdullah Faiz Saeed Al Asmari, Mohammad Arsalan Khan, Noman Mustafa, Mohammad Mursaleen, Saiful Islam. Faster RCNN based robust vehicle detection algorithm for identifying and classifying vehicles, published online: 30 July 2023.

[20] Redmon, J., & Farhadi, A. (2017). YOLO9000: Better, faster, stronger. arXiv preprint arXiv:1612.08242.

[21] Ren S, Kai H, Ross G. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks[J]. IEEE Transactions on Pattern Analysis & Machine Intelligence, 2017, 39(6):1137-1149.

[22] Wei Yang, Ji Zhang, Hongyuan Wang, Zhongbao Zhang. A vehicle real-time detection algorithm based on YOLOv2 framework. 2018.

[23] Muhammad Hussain. YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection.

[24] Naif Al Mudawi, Asifa Mehmood Qureshi, Maha Abdelhaq, Abdullah Alshahrani, Abdulwahab Alazeb, Mohammed Alonazi, Asaad Algarni. Vehicle Detection and Classification via YOLOv8 and Deep Belief Network over Aerial Image Sequences.

[25] Our DataSet

smart traffic management system Object Detection Dataset by noor (roboflow.com)

[26] Website for the dataset - first source

https://www.kaggle.com/datasets/rahat52/traffic-density-singapore/data

[27] Website for the dataset - second source

https://universe.roboflow.com/grad-project-tjt2u/emergency-cars-s0yq

[28] Roboflow homepage

Roboflow: Computer vision tools for developers and enterprises

[29] Roboflow website (Annotation process)

smart traffic management system Dataset > Overview (roboflow.com)

[30] Google Colab Homepage

Welcome To Colab - Colab (google.com)

[31] Google Colab website (Train process)

Google Colab

[32] Yaml file

args.yaml - Google Drive

[33] GitHub link to the source code for our project

GitHub - NoorJehadKhawaja/Smart-traffic-management-system: using custom dataset for object detection using YOLO model optimizes traffic flow, prioritizes emergency vehicles, and enhances safety at intersections.