T.C.
ISTANBUL OKAN UNIVERSITY
INSTITUTE OF GRADUATE SCIENCES

THESIS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN AUTOMOTIVE MECHATRONICS AND
INTELLIGENT VEHICLES PROGRAM

Abeer Sarhan Mazyoon AL-RUFAYE

DRIVING ASSISTANT USING YOLOV8 DEEP LEARNING MODEL FOR CAR, PERSON AND TRAFFIC DETECTION WITH A DECISION-SUPPORT ALGORITHM

ADVISOR
Dr. Öğr. Üyesi Sina ALP

ISTANBUL, September 2023

T.C.
ISTANBUL OKAN UNIVERSITY
INSTITUTE OF GRADUATE SCIENCES

THESIS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN AUTOMOTIVE MECHATRONICS AND
INTELLIGENT VEHICLES PROGRAM

Abeer Sarhan Mazyoon AL-RUFAYE
(203005005)

DRIVING ASSISTANT USING YOLOV8 DEEP LEARNING MODEL FOR CAR, PERSON AND TRAFFIC DETECTION WITH A DECISION-SUPPORT ALGORITHM

ADVISOR
Dr. Öğr. Üyesi Sina ALP

ISTANBUL, September 2023

T.C.

ISTANBUL OKAN UNIVERSITY

INSTITUTE OF GRADUATE SCIENCES

THESIS
FOR THE DEGREE OF
MASTER OF SCIENCE
IN AUTOMOTIVE MECHATRONICS AND
INTELLIGENT VEHICLES PROGRAM

Abeer Sarhan Mazyoon AL-RUFAYE

(203005005)

DRIVING ASSISTANT USING YOLOV8 DEEP LEARNING MODEL FOR CAR, PERSON AND TRAFFIC DETECTION WITH A DECISION-SUPPORT ALGORITHM

Presentation Date of Thesis : September 10, 2023

Submission Date of Thesis : September 10, 2023

Advisor: Dr. Öğr. Üy. Sina ALP _____

Jury Members: Doç. Dr. Ömer Cihan KIVANÇ _____

Dr. Öğr. Üy. Can GÖKÇE _____

Dr. Öğr. Üy. Didem KIVANÇ TÜRELİ _____

Prof. Dr. Mehmet Serdar Ufuk TÜRELİ _____

İSTANBUL, September 2023

To My Beloved Family

ACKNOWLEDGMENT

I would like to begin by expressing my deepest gratitude to my supervisor, Assoc. Prof. Dr. Sina Alp, for his invaluable support and guidance throughout this journey. His mentorship has been instrumental in helping me reach this moment. I also want to acknowledge every challenging experience and obstacle, as they have strengthened my character and resilience. After this, my heartfelt thanks go to my friends and classmates who supported me during my time in the master's program, and to my family for their unwavering encouragement. I am grateful to Turkey, the country where I reside, for enriching my life with valuable experiences and cultural insights that have shaped my outlook on the future. I would also like to extend my sincere appreciation to Okan University for providing me with this opportunity, and to the members of my thesis jury, including Assoc. Prof. Dr. Omer Cihan Kivanc and Assist. Prof. Dr. Can Gokce for their time, insights, and constructive feedback.

TABLE OF CONTENTS

ABBREVIATIONS

AI : Artificial Intelligence
BBOX : Bounding Boxes
BDD : Diverse Driving Dataset for Heterogeneous Multitask Learning
CLAHE : Contrast Limited Adaptive Histogram Equalization
CLS : Class
CLS_LOSS : Class Loss
CM : Confusion Matrix
CNN : Convolutional Neural Networks
COLAB **:** Collaboration
CONF : Confidence Value
DF : Distribution Focal
DFL_LOSS : Distribution Focal Loss
DL : Deep Learning
FLOPs : Giga Floating-Point Operations
FN : False Negatives
FP : False Positives
GPU : Graphical Processing Unit
IoT : Internet of Things
LDB : Learning Domain Biased
mAP : mean Average Precision
ML : Machine Learning
OS : Pixel-Level Adaptation
PLA : Pixel-Level Adaptation

PROPS **:** probability of the predicted class

SPPF : Spatial Pyramid Pooling Feature

TP : True Positives

TN : True Negatives

VAL : Validation

YAML : yet another markup language

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

DRIVING ASSISTANT USING YOLOV8 DEEP LEARNING MODEL
FOR CAR, PERSON AND TRAFFIC DETECTION
WITH A DECISION-SUPPORT ALGORITHM

Decision-support systems that is based on artificial intelligence become very common in computer science applications nowadays. However, they are still facing some challenges and needs more improvements. In this research, a driving assistant comprehensive framework based on two main stages (multi-stage). The first step is the deep learning object detection model in which, the YOLOV8 model is utilized. While in the second part, a novel decision-support algorithm is proposed to make the driving decision based on the YOLOV8 predictions. First, a comprehensive dataset of traffic-based objects (car, bike, truck, bus, train, motor, person, traffic light, traffic sign, and rider) which is the BDD100K dataset is utilized. This dataset is already split into training and test sets. The training set is used to train the YOLOV8 model after applying many data albumenations operation. The resulting YOLOV8 based model is then evaluated. After that the predictions results, including the bounding boxes, the predicted classes and the confidence values of each test sample is passed to the decision-support algorithm that is responsible of counting number of instances per component, define the traffic status (light, moderate heavy), define the driving decision, and define the routing direction of the car. The proposed methodology is evaluated using many test cases which proved the high detection accuracy and the right driving decision. Current car systems can use the designed decision-support model supported by the YOLOV8 model for a better safe driving.

Keywords: Decision-support, Deep Learning, YOLO, YOLO V8, Object Detection, Driver Assistant, Car Detection.

ÖZET

YOLOV8 DERİN ÖĞRENME MODELİ KULLANAN
ARABA, KİŞİ VE TRAFİK TESPİTİ İÇİN
KARAR DESTEK ALGORİTMASI İLE SÜRÜŞ ASİSTANI

Yapay zekaya dayalı karar destek sistemleri günümüzde bilgisayar bilimi uygulamalarında çok yaygın hale gelmiştir. Ancak, hala bazı zorluklarla karşı karşıyadırlar ve daha fazla iyileştirmeye ihtiyaç duymaktadırlar. Bu araştırmada, iki ana aşamaya (çok aşamalı) dayalı kapsamlı bir sürüş asistanı çerçevesi geliştirilmiştir. İlk adım, YOLOV8 modelinin kullanıldığı derin öğrenme nesne algılama modelidir. İkinci bölümde ise, YOLOV8 tahminlerine dayalı sürüş kararını vermek için yeni bir karar destek algoritması önerilmektedir. İlk olarak, BDD100K veri kümesi olan trafik tabanlı nesnelerin (araba, bisiklet, kamyon, otobüs, tren, motor, kişi, trafik ışığı, trafik işareti ve sürücü) kapsamlı bir veri kümesi kullanılmıştır. Bu veri kümesi halihazırda eğitim ve test kümelerine ayrılmıştır. Eğitim seti, birçok veri albüminasyon işlemi uygulandıktan sonra YOLOV8 modelini eğitmek için kullanılır. Daha sonra eğitilen YOLOV8 modeli, test seti ve performans ölçümleri (hassasiyet, geri çağırma ve mAP) kullanılarak değerlendirilir. Bundan sonra, sınırlayıcı kutular, tahmin edilen sınıflar ve her test örneğinin güven değerleri dahil olmak üzere tahmin sonuçları, bileşen başına örnek sayısını saymaktan, trafik durumunu (hafif, orta ağır) tanımlamaktan, sürüş kararını tanımlamaktan ve aracın rota yönünü tanımlamaktan sorumlu olan karar destek algoritmasına geçirilir. Önerilen metodoloji, yüksek algılama doğruluğunu ve doğru sürüş kararını kanıtlayan birçok test vakası kullanılarak değerlendirilir. Mevcut

araç sistemleri, daha güvenli bir sürüş için YOLOV8 modeli tarafından desteklenen tasarlanmış karar destek modelini kullanabilir.

Anahtar Kelimeler: Karar desteği, Derin Öğrenme, YOLO, YOLO V8, Nesne Algılama, Sürücü Asistanı, Araç Algılama.

CHAPTER 1. INTRODUCTION

1.1 Background

Decision support systems are type of artificial intelligence frameworks that use intelligent algorithms to let the machine making its decision about a specific topic [1, 2].

In the current study's domain, the decision support idea is based on capturing the surrounding scenes, analyzing them, detecting possible objects (car, person, traffic), and then make the final decision of routing the car (for example, slow down the speed, increase the speed, turn left or right, immediately stop, etc.) [3].

Recently, artificial intelligence AI and its main branches (machine learning ML and deep learning DL) improved the decision making process in all life fields, including: medical domain [4, 5], security [6], military [7], space applications [8], geology [9], auto-driving [10], remote sensing [11], etc.

Auto-driving and driving assistant applications have received a good attention in the computer vision-related studies. Such applications contain many sub-tasks, including, car detection, person detection, traffic sign detection, traffic detection, traffic light detection, etc. [12].

Driver assistant systems increase the car and road safety in many points of view. Such systems can detect nearby objects (like cars, trucks, persons, etc.), give information about the traffic status, produce warning before any possible accident, etc. [12, 13].

Car accidents are responsible of hundreds of deaths annually. Crash accidents likelihood increases in crowded roads, especially if the driver is drowsy or there are persons passing the street.

Providing the car and the driver by an AI technology that makes warning, facilitates his driving, monitors the traffic, and gives directions advices can reduce such accidents and improve the car and driver safety [14, 15].

Figure 1.1 shows how to fix a capturing device in front of the driver capturing the direct scene in front of the camera. This device is designed in study [16].

Figure 1.1. Fixing camera equipment (capturing device) in the car.

1.2 Scope of drive assistant systems

Different activities are involved within the drive assistant system and can be concluded in the following [17] as illustrated in Figure 1.2.

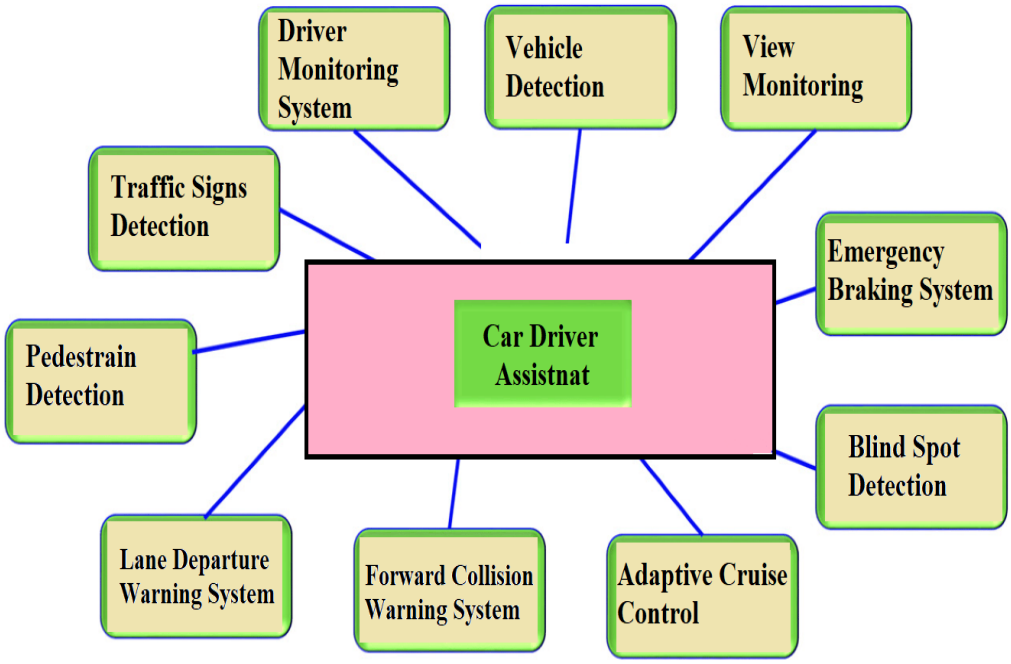

Figure 1.2. Car driver assistant main operations.

1. Vehicle detection
2. Traffic sign detection
3. Driver monitoring
4. Driver warning system
5. Forward collision warning system
6. Emergency braking system

1.3 Challenges of drive assistant systems:

There are many challenges facing drive assistant systems including:

Occlusion: some cars, persons, traffic lights, and other objects can be occluded by other objects partially so this can affect the detection process.

Changing environment: from day to night, high illumination to low illumination, changes in objects shapes, their colors, sizes, etc. can affect the performance of the detection systems.

Multi-objects: any tracking or detection system should consider the problem of detecting all possible objects that can affect the driving decision.

Need to work in real-time: detection algorithms should take into consideration the computational time since they will be executed in real environments.

All above mentioned challenges can be solved by using a good comprehensive dataset which may include all possible detectable objects. Besides that, a lightweight AI model is needed in order to make a real-time response (less than one second).

In our study, we try to solve these problems by using a comprehensive dataset includes 10 different classes that can be detected on the road. The second solution is the usage of the lightweight efficient model (YOLOV8) which is one of the fastest models in DL object detection field.

1.4 Research problem

Current state-of-art object detection AI models have the ability to only detect objects within a specific scene. The AI or DL model can detect cars, trucks, trains, bikes, persons, riders, traffic lights, traffic signs, etc. However, they can't make driving decisions. Any AI model gives only predictions which represent the bounding boxes around the detected objects, their detected class, and the confidence value of detecting the object. However, if we want to make driving decisions, we need to derive a decision-support algorithm which can analyze the output of the AI model and define if there any possible warning, detect any possible heavy traffic, define the distance between the car and all cars in the scene, and finally make decision (stop the car, continue with the same speed, slow down, speed up, move right, or move left).

In this research, we will make a multi-stage traffic monitoring system which includes two main tasks. The first one is the detection task and the second one is the decision-support algorithm.

In this research, a comprehensive dataset of 10 classes will be utilized in order to detect all possible important objects within the scene. Then, a novel decision-support algorithm is created and applied to the output of the detection model in order to make a driving decision and help the driver to safely drive the car.

1.5 Research objectives

The study aims to:

1. Train a YOLOV8 model using a comprehensive traffic dataset consisting of 10 different detectable classes in roads.
2. Create a novel decision-support algorithm that is capable of analyzing the detection output of the YOLO v8 trained model and make driving decision.
3. Help driver to make right driving decisions in different environments (dark or night).
4. Help driver to stay safe by providing him/her with all possible traffic information, including the number of detected objects of each class, the traffic status (light, moderate, heavy), the routing (left/right), and the number of nearby cars (or trucks, bikes, etc.).
5. Improve the safety levels for cars and persons and limit the accidents likelihood.
6. Validate the proposed AI model by testing the trained model using test images.
7. Validate the proposed decision-support algorithm by performing real-time tests and see the response of the algorithm in many cases.

1.6 Research Methodology

In this research, the following methodology is proposed:

A multi-stage decision-support deep learning based system will be designed, implemented and tested. To do this, the following steps is proposed:

First, a dataset of the most common objects on the road in front of the car driver will be acquired. The dataset must contain annotations or bounding box ground truth. Second, a deep learning model capable of making object detection tasks will be utilized. In this step, we choose the YOLOV8 model due to its efficiency, high performance, and low computational time.

The AI model will be trained using the training set of the dataset. However, in this step not only images but also their corresponding ground truth will be utilized to make the supervised learning of YOLOV8 model.

After the model is trained, the AI-backbone of our system will be examined to evaluate it ability to detect images as shown in Figure 1.3. After the evaluation process is done, YOLO V8 model will be ready for the decision-support step.

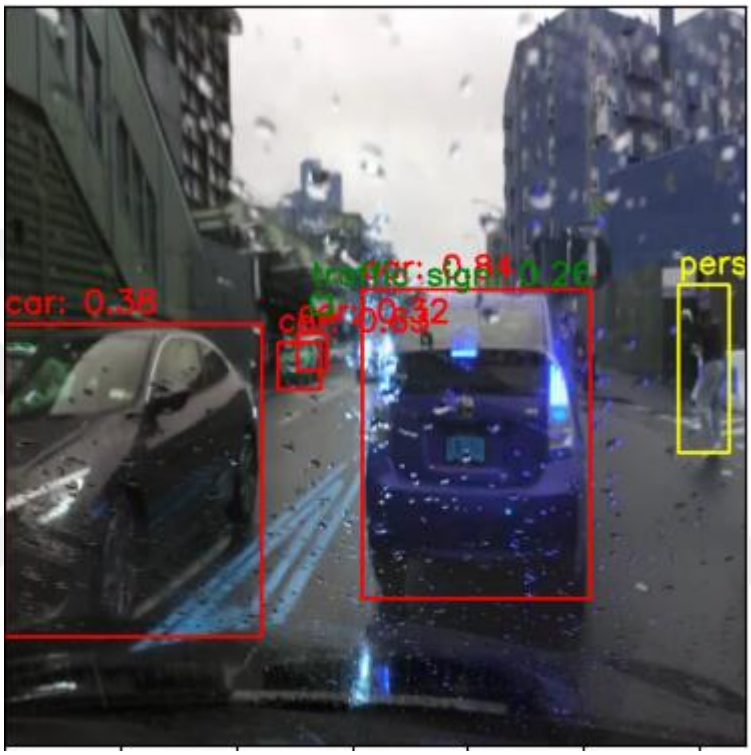

Figure 1.3. Detection example of YOLO V8 model.

The detection result will contain bounding boxes around the detected objects, the type or class of each detected object, and the confidence values of each detected component.

The decision-support step will take the detection results of the trained and evaluated YOLO V8 model and analyze them to interpret and derive the driving decision.

The decision-support algorithm includes counting detected objects, computing their areas, computing their percentage in the scene, defining their distance from the capturing device inside the car, define the traffic status, the location of the detected objects inside the scene, route the driver to the appropriate direction, and make the

driving decision that can be either slowing down, or speeding up, or remaining the same speed, or stopping.

1.7 Thesis overview

The current thesis consists of five chapters:

Chapter 1:

Chapter 1 stands for the introduction in which the main theoretical concepts about the studies problem will be introduced and discussed.

The importance and aim of the research is also clarified. The proposed methodology is also described.

In ***chapter 2***:

The literature review, including a comprehensive study of the most recent studies in the fields of: car detection, traffic light detection, traffic sign detection, road monitoring based on AI, ML or DL technologies.

Chapter 3:

Chapter 3 will include the utilized materials (software and hardware materials), description about the utilized dataset, number of classes in the dataset, image characteristics, and main benefit of the dataset.

The implementation, experiments, and the important results will be presented in ***Chapter 4***:

Not only results, but also a detailed discussion will be introduced inside Chapter 4.

Besides that, the evaluation of the trained YOLOV8 model with many test images will be introduced. Moreover, the decision-support algorithm will also be tested using many real-time test images, including multiple classes with different scales, different distance from camera, different environment conditions, different traffic status, and various cars distribution.

Chapter 5:

The final chapter (chapter 5) will include the conclusion, and possible future works. In this chapter, the main finding and the most notable results will be clarified.

A summary with all methodology steps, the utilized dataset, the main results, and the possible improvement and future perspectives will be presented.

CHAPTER 2. LITERATURE REVIEW

2.1 Related work in the field of object detection

Many studies have been introduced in the field of traffic sign detection [18], car detection, person detection, traffic light detection, crowd detection, and traffic peak detection. However, all these studies didn't take into consideration the decision-support capability.

2.1.1 Traffic sign detection

Traffic sign detection using spatial pyramid pooling and scaling techniques was proposed by Tai et al. [19] The idea was to improve the YOLO model to better feature extraction and detection. To apply their methodology, they collected a dataset of four main signs (no stop sign: 250 images, no entry: 235 images, speed limit: 185 images, and no parking: 230 images). A dataset of 4 classes and 900 images was collected and split into 70% for training and 30% for test. They got 99% as precision and 90.09% as mAP. However, their methodology utilized too small dataset and only 4 classes.

YOLO V4-tiny was utilized in a study by Wang et al. [20] for traffic sign recognition. To enhance the detection operation, the K-means clustering was applied to the generate the anchor appropriate size. They also applied the large-scale feature map optimization for improving the detection operation. They applied the experiments to the TT100k dataset and got mAP value of 46.36% without clustering, but by applying clustering, they got 47.16% as mAP. The utilized dataset size was 9176

images with 221 types of annotations. Since the image resolution was huge, they resized images into 608*608.

Ahmed et al. [21] introduced a deep learning framework for the aim of traffic sign detection in challenging weather conditions based on two convolutional neural networks CNN architectures for sign detection and classification. They utilized the CURE-TSD dataset and got precision and recall values of 91.1% and 70.71%, respectively. The dataset consists of 49 videos, each of which contains 300 frames of traffic signs.

Another study by Min et al. [22] utilized the TT100k dataset and a lightweight deep learning model called “Light-weight RefineNet” for the purpose of traffic light semantic understanding. Then, the multi-scale densely connected detector was utilized along with the k-means++ clustering algorithm for the aim of traffic light detection. They achieved accuracy of 92.8%.

In a study by Wang et al. [23], an efficient real-time traffic detection based deep learning model was proposed. They Utilized the YOLOV4 model and a bidirectional pyramid module for the aim of traffic detection. They used the TT100k dataset and got 66% as mAP.

2.1.2 Car detection

Car detection is another essential part of driving support systems.

Vehicle detection in infrared surveillance images was proposed in a study of Fan et al. [24] YOLO V5 model was also trained using the “Space Cup competition” dataset consisting of 16000 images. They also used a DenseBlock module to improve the detection capability of the YOLO V5 model. They achieved mAP value of 73.1%.

Kang et al. [25] introduced a type-1 fuzzy attention technique supporting the YOLO model for a better car detection operation. They utilized two datasets during the training process and one for the validation task, and achieved 70% and 50.3% as AP50 and AP, respectively. The utilized dataset consists of 82085 training and 56127 test images. However, they only used it for validation part, while in the training process, they utilized the DAWN dataset consisting of 1000 images and Cityscapes

dataset containing 2975 training images and 500 validation images and 1525 test images.

Mishra et al. [26] introduced a vehicle detection framework based on YOLO V5 model and KITTY dataset. The dataset consisted of 11682 images of three main classes: cars, trams and vans. They got a mAP value of 0.67. They also evaluated their methodology on the Indian Traffic dataset and got mAP of 0.65.

Safaldin et al. [27] utilized the recent version of YOLO model (YOLO V8 model) for the aim of vehicle detection. They utilized the KITTY dataset and applied many preprocessing operations, including preparing video frames, decoding and isolation. Then, they trained the YOLO V8 model using the training set. As a result, they got 90% as mAP.

2.1.3 Traffic monitoring and assistance

Zho et al. [28] utilized both internet of things (IoT) and machine learning (ML) in a hybrid driving monitoring system. Their model was capable of producing instructions to drivers for a safe driving. Another monitoring system proposed by Darapaneni et al. [29] proposed a driver monitoring system for safety driving. They depended on AI technology. However, such systems lack of validation and real tests.

Jegham et al. [30] introduced a deep learning-based model for driving assistant and monitoring. They utilized the hard spatial attention module. They utilized the MDAD dataset consisting of 50 driver videos performing 16 in-vehicle different actions. The experiments showed that the proposed methodology achieved 80.73% detection accuracy of driver actions. However, the study considered the actions of the drivers and not monitoring the traffic status.

Both machine learning and YOLO model were utilized in a study by Sakhare et al. [30] for the purpose of traffic monitoring. They first took images from video camera. Then they processed them and entered them to the YOLO V3 model to make the detection part. After that, the waiting time of the driver is calculated and the decision to pass or wait was made. Their system was evaluated only in real-time and no evaluation metrics were conducted.

YOLO V7 was utilized in a study by Kunekar et al. [31] for traffic management and increase the driving safety. In their work, they tracked and counted all vehicles on

the road. Besides that, they calculated the traffic signal. No dataset mentioned and no evaluation process were applied. They experimented their model on two images only.

2.1.4 Studies that worked on the BDD100K dataset

In order to show the most related work to our study, the recent studies that worked on the same dataset of the current thesis are studied.

Song et al. [32] proposed a non-exemplar learning domain biased method (LDB). They freeze the weights of the base model and learn the domain bias weights for each different domain. In real tests, they implemented the nearest mean classifier to select the appropriate domain of the test image. They utilized both BDD100K and Pascal VOC datasets. A total dataset of 80000 images of BDD100K were utilized (70000 for training and 10000 for validation and test). For Pascal VOC dataset, they selected 6 categories. As a result, they got 51.1% as mAP on the BDD100K dataset.

Patch-based selection method was proposed in a study by Zhang et al. [33]. They also utilized the reinforcement for the aim of car and traffic detection. They utilized the BDD100K dataset and achieved mAP of 4.33 and a precision of 0.457.

Yu and Lu [34] presented an anchor-free object detection model for traffic monitoring. They adopted the Pixel-Level Adaptation (PLA) in order to get the local features of the scene to improve the detection task. Experiments were applied to many datasets; however, they got a mAP of 30.69 on the BDD100K dataset.

Another study by Cong et al. [35] utilized the same BDD100K dataset for visual detection in autonomous driving. They created a two-stage detection method based on efficient mining idea and lightweight architecture including, YOLO, fast RCNN, and MILLIEYE models. They also applied the label allocation optimization idea that can take into consideration the similarities and dissimilarities between target labels (classes) leading to a better detection operation. The achieved mAP values between 0.578.

Another study by Lu et al. [36] used the BDD100K dataset and developed a lightweight vehicle detection systems based on the concept of feature pyramid and attention mechanism.

They first utilized the MobileNetV3_large model in the front-end part. Then the weighted bi-directional fusion network was utilized to acquire multi-dimensional features of the objects within the scene.

They compared their work by many previous methodologies that were worked on the BDD100K dataset and prove that their methodology outperformed all previous ones with mAP value of 17.6 and 88.7 on BDD100K and KITTI datasets, respectively.

2.2 Related work conclusion

The following notes can be concluded about the previous studies:

1. Some previous studies utilized small datasets.
2. Some studies achieved low or bad performance.
3. Most studies focused on specific object to be detected (either vehicle, or car, or traffic lights).
4. None of the studies performed an entire driving monitoring systems that is capable of detecting the most relevant objects in front of the driver. Most of studies either monitor the driver and made warning in case of drowsiness, or monitor specific objects like car and trucks and define if the road is crowded or not.

In this study we will:

1. Utilize a huge size dataset (BDD100K dataset) that contains 10 different classes (car, truck, bike, person, rider, traffic light, traffic sign, train, bus, motor).
2. Use a lightweight model (YOLO V8) model which is not only fast but also accurate.
3. Add a novel decision-support algorithm to the output of the YOLO model to make a multi-stage driving assistant model that is capable not only to detect 10 different classes in front of the driver, but also take the right driving decision and deliver it to the driver for a safe driving.
4. The proposed decision-support algorithm will be able to count samples of each class within the scene in front of the driver, compute their distance from the camera of the driver's car, compute their area, and find their location in the scene.

All this information will be next used to make two decisions:

1. The driving decision: stop the car, continue with the same speed, increase the speed a little bit, reduce the speed, etc.
2. The routing choice: to guide the driver to the direction that has the least traffic.
3. The traffic status in general: which can be light, moderate or heavy.

CHAPTER 3. MATERIALS AND METHODOLOGY

In this chapter, the utilized materials and the proposed methodologies, including the learning model, and the decision support system will be described, explained and further clarified either by algorithms or diagrams.

3.1 Materials

The used materials are divided into two main categories:

The software tools, including:

1. Python programming language.
2. Colab environment to implement Python codes.
3. The utilized dataset.

The hardware tools:

Virtual Colab environment consists of a 12.7 RAM, 15 GB of GPU and disk.

3.1.1 Dataset

In this study, the BDD100K for self-driving car dataset [37] is utilized due to many reasons:

1. The dataset size is huge and suitable for training YOLOV8 model since the deep learning networks requires too much data to make a reasonable knowledge.

2. The dataset consists of all possible trackable objects in the road that can affect the driving process, including: car, person, traffic sign, traffic light, truck, train, bus, motor, bike, and rider).
3. The nature of the captured images of the BDD100K dataset s very suitable for our mission in which the aim is to detect road's objects in front of a camera fixed on the driver car.
4. Each image in the dataset is available with bounding boxes defining regions of each object in the image (so our mission here is a supervised learning process in which the model needs to train on couple of input/output samples. In our mission, the couple represents an image and their corresponding bounding boxes around all possible detectable objects in the captured scene.

Figure 3.1 shows some examples of the dataset with their corresponding bounding boxes.

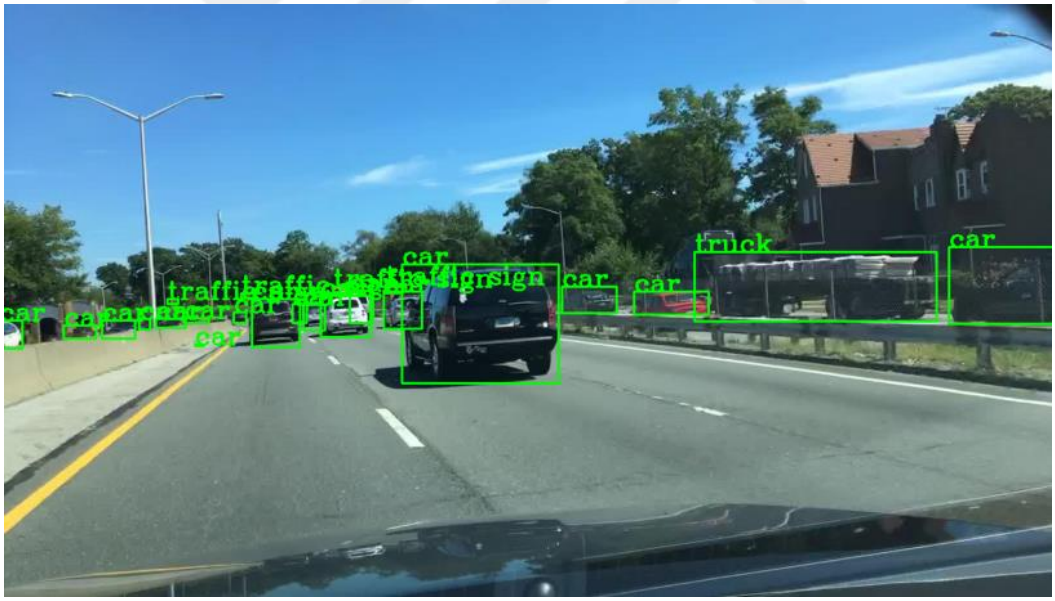

Figure 3.1. Examples of the BDD100K dataset: Example 1 of a test sample with bounding boxes.

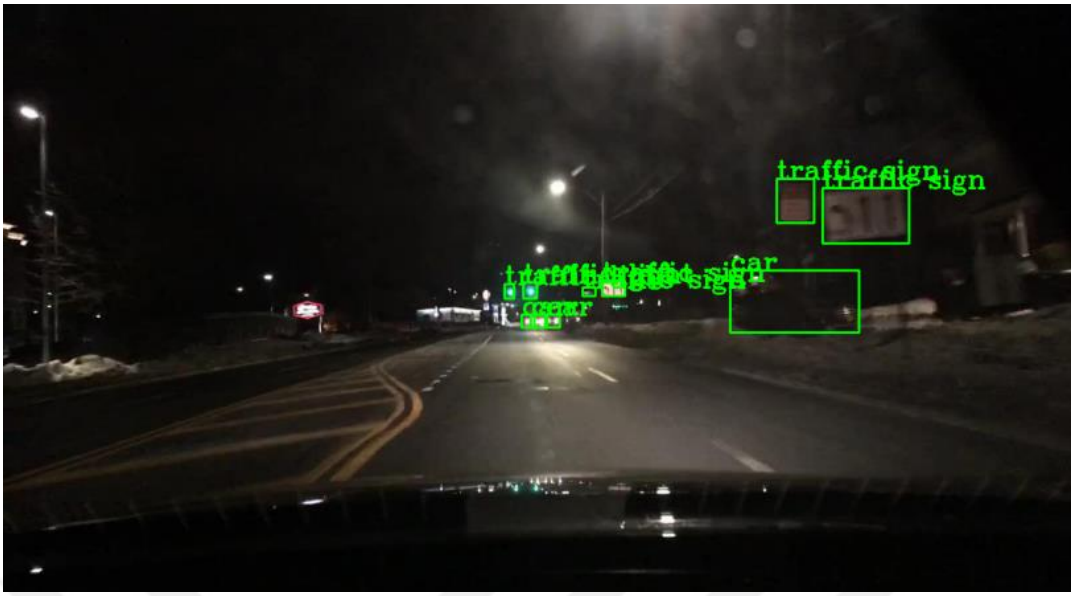

Figure 3.2. Examples of the BDD100K dataset: Example 2 of a test sample with bounding boxes.

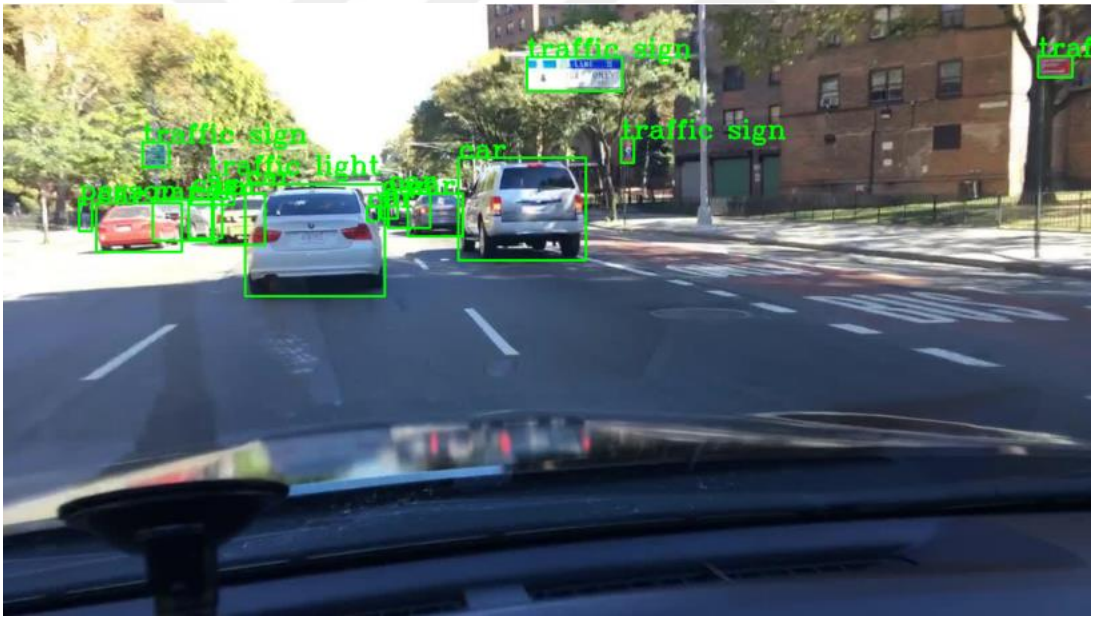

Figure 3.3. Examples of the BDD100K dataset: Example 3 of a test sample with bounding boxes.

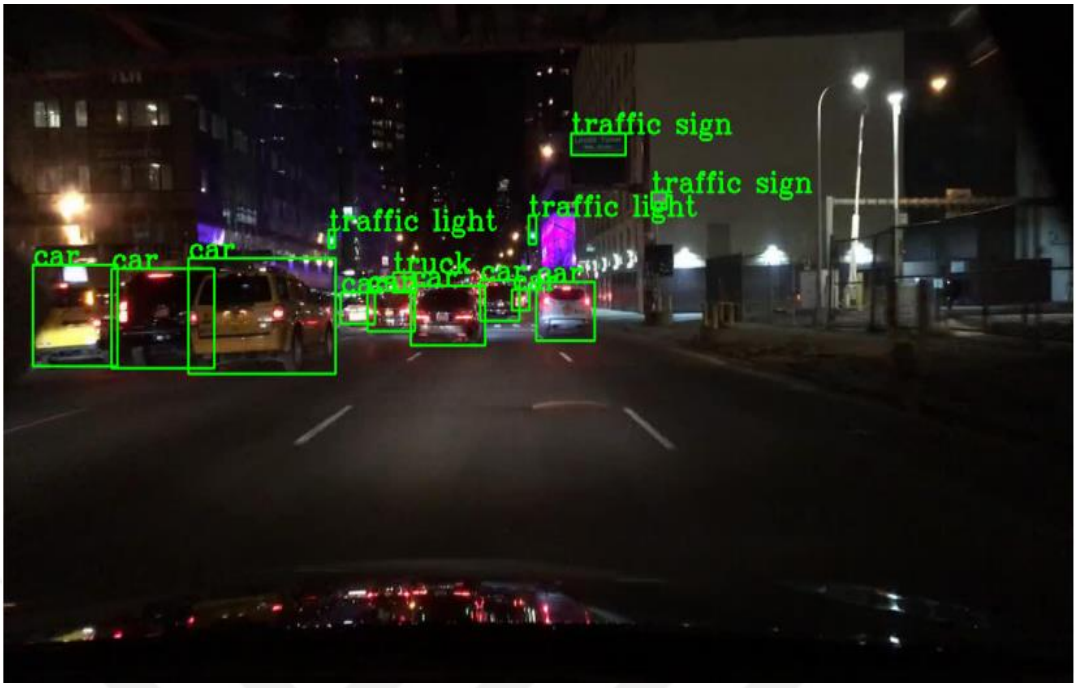

Figure 3.4. Examples of the BDD100K dataset: Example 4 of a test sample with bounding boxes.

Table 3.1 contains the main information about the BDD100K dataset utilized in this study.

Table 3.1 BDD100K dataset characteristics.

Property	Description
Number of images in training set	20000
Number of images in validation set	2000
Number of annotations (bounding boxes (BBOX)) in training set	20000
Number of annotations (bounding boxes) in validation set	2000
Number of classes	10
Image Dimension	720*1280
Availability	Open[1]

[1] Available at: https://www.kaggle.com/datasets/aayusmaanjain/bdd100k-for-self-driving-cars

3.1.2 Utilized Python Libraries

There are many dependencies (libraries) which are used in this study, including:

- Pytorch: A machine learning framework to build and train deep neural networks including a support by the GPU capabilities.
- OS: operating system library that is used to perform file-related operations like joining paths.
- Numpy: the matrix creation and manipulation library.
- Cv2: the open-cv library is an image processing library that is used to apply image-related operations starting from reading, showing, saving to preprocessing and manipulating images.
- Pandas: a framework to create, read, write, manipulate data frames.
- Matplotlib: a library for plotting or showing figures.
- Glob: This library is used to search directories and retrieve lists of filenames that match specific criteria.
- Yaml: Human readable serialization file allowing to read and write dataset yaml files required for some training operations like the training of YOLOV8 model.
- Ultralytics: a framework for YOLO model (creation, training, validation and tracking).

3.2 Proposed Methodology:

Figure 3.5 illustrates the detailed methodology proposed in this study.

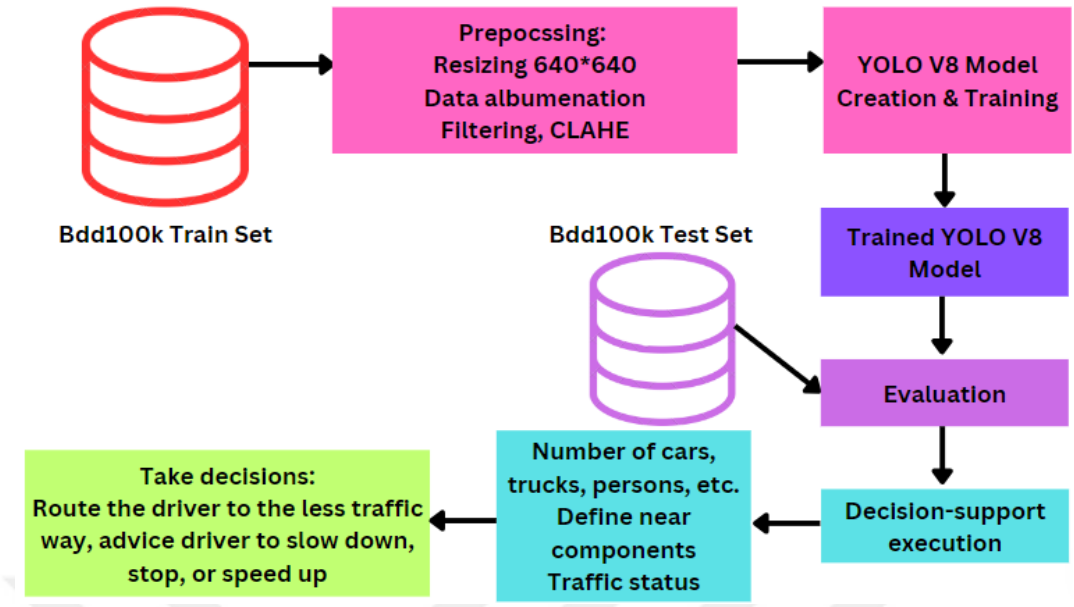

Figure 3.5. The proposed methodology.

The first step of our proposed methodology is the dataset preparation and preprocessing in which many ***data augmentation*** and ***albumentation*** operations are performed, including:

1. Blur: In this operation, the Gaussian blur [38] is applied to the dataset images. The blur_limit parameter specifies the range of blur kernel sizes which will be randomly chosen between 3 and 7 with a probability of 1%.
2. MedianBlur [39]: This stands for the median filtering on the image. Similar to the Blur transformation, it also has a blur_limit parameter. The kernel size for median filtering will be randomly selected between 3 and 7 pixels with a probability of 1%. The median smoothing targets the isolated points in images.
3. ToGray: Converting the image to grayscale with a probability of 1%, meaning that only 1% of the images will be converted to grayscale.
4. CLAHE (Contrast Limited Adaptive Histogram Equalization) [40, 41]: This transform enhances the contrast of an image by equalizing the histogram in specific regions of the images. It has two parameters: the clip_limit and the tile_grid_size. The clip_limit parameter controls the amount of contrast enhancement, while the tile_grid_size defines the grid size for dividing the image. In our experiments, CLAHE will be

performed with a probability of 1%, and the clip limit ranges from 1 to 4.0.

The second step is the creation of YOLOV8 model and train it using the training set of the BDD100K dataset. Figure 3.6 illustrates the architecture of the YOLOV8 model.

```
                 from  n    params  module                                       arguments
  0                -1  1       928  ultralytics.nn.modules.conv.Conv             [3, 32, 3, 2]
  1                -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  2                -1  1     29056  ultralytics.nn.modules.block.C2f             [64, 64, 1, True]
  3                -1  1     73984  ultralytics.nn.modules.conv.Conv             [64, 128, 3, 2]
  4                -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  5                -1  1    295424  ultralytics.nn.modules.conv.Conv             [128, 256, 3, 2]
  6                -1  2    788480  ultralytics.nn.modules.block.C2f             [256, 256, 2, True]
  7                -1  1   1180672  ultralytics.nn.modules.conv.Conv             [256, 512, 3, 2]
  8                -1  1   1838080  ultralytics.nn.modules.block.C2f             [512, 512, 1, True]
  9                -1  1    656896  ultralytics.nn.modules.block.SPPF            [512, 512, 5]
 10                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11           [-1, 6]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 12                -1  1    591360  ultralytics.nn.modules.block.C2f             [768, 256, 1]
 13                -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14           [-1, 4]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 15                -1  1    148224  ultralytics.nn.modules.block.C2f             [384, 128, 1]
 16                -1  1    147712  ultralytics.nn.modules.conv.Conv             [128, 128, 3, 2]
 17          [-1, 12]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 18                -1  1    493056  ultralytics.nn.modules.block.C2f             [384, 256, 1]
 19                -1  1    590336  ultralytics.nn.modules.conv.Conv             [256, 256, 3, 2]
 20           [-1, 9]  1         0  ultralytics.nn.modules.conv.Concat           [1]
 21                -1  1   1969152  ultralytics.nn.modules.block.C2f             [768, 512, 1]
 22      [15, 18, 21]  1   2119918  ultralytics.nn.modules.head.Detect           [10, [128, 256, 512]]
Model summary: 225 layers, 11139470 parameters, 11139454 gradients, 28.7 GFLOPs
```

Figure 3.6. YOLO V8 model architecture.

According to our data, Figure 3.7 clarifies the model main characteristics and the utilized training parameters.

Table 3.2 YOLOV8 model characteristics and training parameters according to our dataset.

Property	Description
Input size	736*736
Output size	10 classes
Number of layers	225
Giga floating-point operations (FLOPs)	28.7
Number of trainable parameters	11139470
Optimizer	Adam
Initial learning rate	0.01
Weight decay factor	0.0005
Loss function	Categorical cross entropy (since we have 10 classes)

The third step is the evaluation process of the trained YOLOV8 model in which, the test set is utilized to evaluate the model (BDD100K is already divided into train set with a ratio of 90% and test (val) set with a ratio of 10%).

In the validation process, the precision, recall, mAP50 and mAP50-95 are utilized.

In the last step, the decision-support algorithm is applied to the output prediction of the trained and evaluated YOLOV8 model in order to make the decision and help the driver for a safe driving.

3.3 YOLOV8 model architecture

In 2015, Redmon et al. [33] introduced the YOLO (you only look once) model by means of Ultralytics company. YOLO is a fast anchor-free model which doesn't depend on a pre-defined bounding boxes to detect objects inside image. Otherwise, YOLO model directly predicts the bounding boxes coordinates. YOLOV8 model also disables the mosaic augmentation operations at the final 10 training epochs reducing the training time. YOLOV8 model is the last version of YOLO model which has a history of versions (YOLOV1 in 2015, YOLOV2 in 2016, YOLOV3 in 2018, YOLOV4 and YOLOV6 in 2020, YOLOV7 in 2022, and YOLOV8 in 2023).

The original YOLO models accepts input image of size 224*224 and consists of 24 convolutional layers with 1*1 kernel-size filters to extract image features, and two fully-connected layers.

The first twenty layers of the YOLO V1 model are originally trained using the ImageNet dataset. The last four Convolutional layers are trained using PASCAL VOC 2007 and 2012 datasets using a randomly initialized weights and under imzge size of 448*448. This higher size of image helps YOLO to capture better features and details, resulting in a more precise object detection operation [42, 43].

The input image of YOLO model is split into S*S grid predicting bounding boxes denoted by B. Each resulted bounding box is corresponding to confidence values for all classes in the dataset (C).

The prediction process involves five parts (Figure 3.7-3.12):

- The confidence score PC.

- The coordinates of the center of the bounding box: denoted by (b_x, b_y).
- The width and height of the bounding box and are denoted by (b_h, b_w).
- The output of the YOLO model: a tensor of size $S \times S \times (5B + C)$.
- Non-maximum suppression step: this step is aimed to eliminate the duplicated predictions (in case that the same bounding box at the same coordinates and with the same width and height is re-produced, this step removes it).

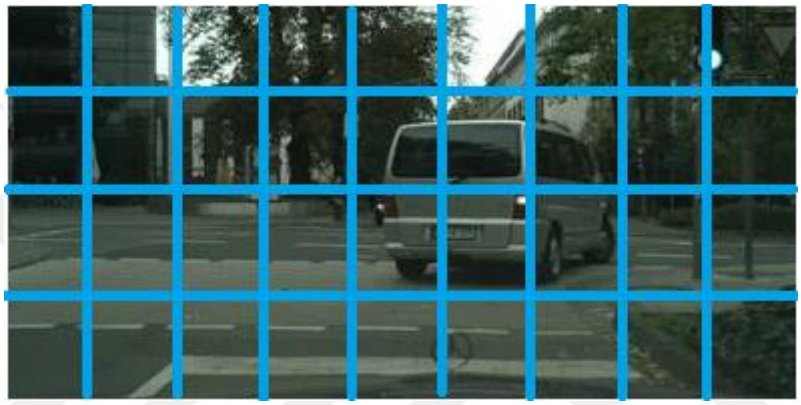

Figure 3.7. Illustration of how YOLO works. A. Image boxes.

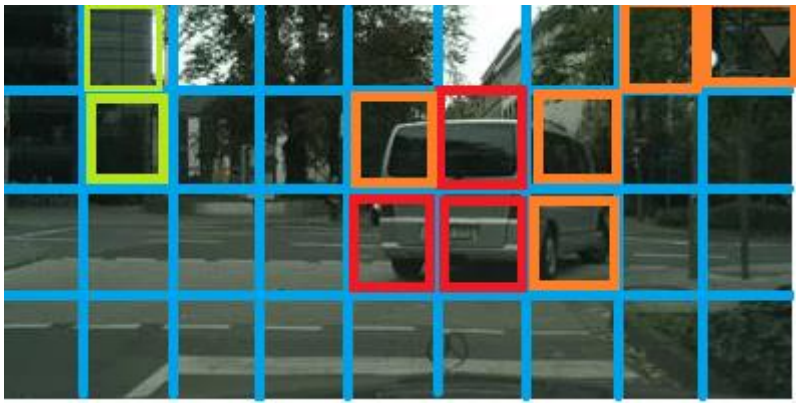

Figure 3.8. B. Define the most and least significant bounding boxes based on their confidence level.

Figure 3.9. C. Cell probabilities (is there a possible detectable object inside this cell): red color for car object, orange color for traffic light object and the blue color for background.

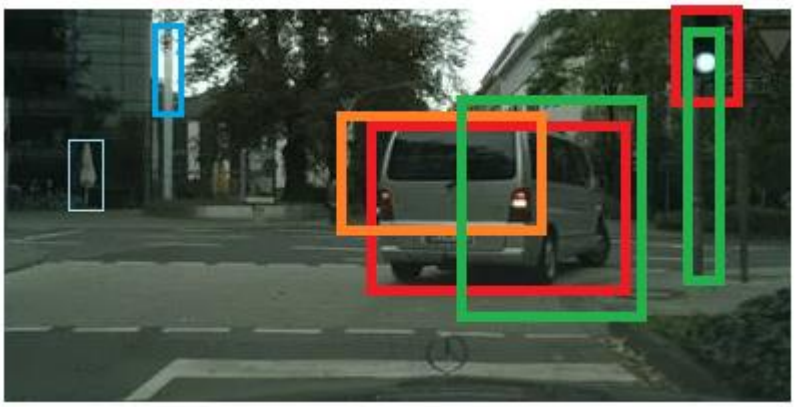

Figure 3.10. D. Emphasizing the bounding boxes using adjacent bounding boxes and intersection between them (low confidence appears in thin rectangles while the most important bounding boxes appear in thick rectangles).

Figure 3.11.E. Detected bounding boxes using a non-maximum suppression operation to find the most appropriate bounding boxes and eliminate outliers

Figure 3.12. F. Final detection results with confidence levels

3.3.1 YOLO V8 model layers (diving in deep concept)

The detailed architecture of the YOLO V8 model I shown in Figure 3.13. [44, 45]. The model consists of three main parts: backbone, neck and head.

The head part of YOLOV8 model consists of many convolutional layers and fully-connected layers responsible of predicting the bounding boxes, the occurrence of the objects and their corresponding confidence scores.

The YOLOV8 model utilizes the well-known mechanism called “attention” in his head part to ensure focusing on the most essential objects inside the image according to the relevant studied problem.

The backbone part is a pyramid-based architecture used to detect objects at different scales making the model more robust against changes in sizes (i.e., YOLOV8 model can detect nearby and faraway cars).

However, the following paragraph describes in detail the architecture of YOLOV8 layers:

In the first part of YOLO V8 model, there are many convolutional layers extract the main features of the input images and called the initial layers. They are built in an increasing way so layer-by-layer, the number of filters are increased, however, the spatial dimension of the activation maps is minimized (pooled).

The C2F block between convolutional layers aims to unify the high-level and contextual features of the image for more precise detection operation [46]. It consists of two convolutional layers in the input and two ones in the output. These layers have kernel size 1 and a stride of 1 with no padding. Between the two convolutional layers

there are a split layer, two bottleneck layers, and one concatenation layer. After many combinations of convolutional and C2F layers, there will be a SPPF (Spatial Pyramid Pooling Feature) layer which always generates a fixed-size outputs whatever is the input image size.

Within the architecture of YOLO model, there are many upsmapling and concatenation parts by which the feature maps are up sampled and concatenated with previous layers. This will allow the model to capture richer information at different scales. After sampling and concatenation blocks, the C2F block provides further processing. The last part of the YOLO V8 model is the detection layer which creates the bounding boxes. The feature maps are scaled to allow for objects of different size to be detected, these scales are from layers with 128, 256, and 512 channels. There are ten classes of objects to be detected.

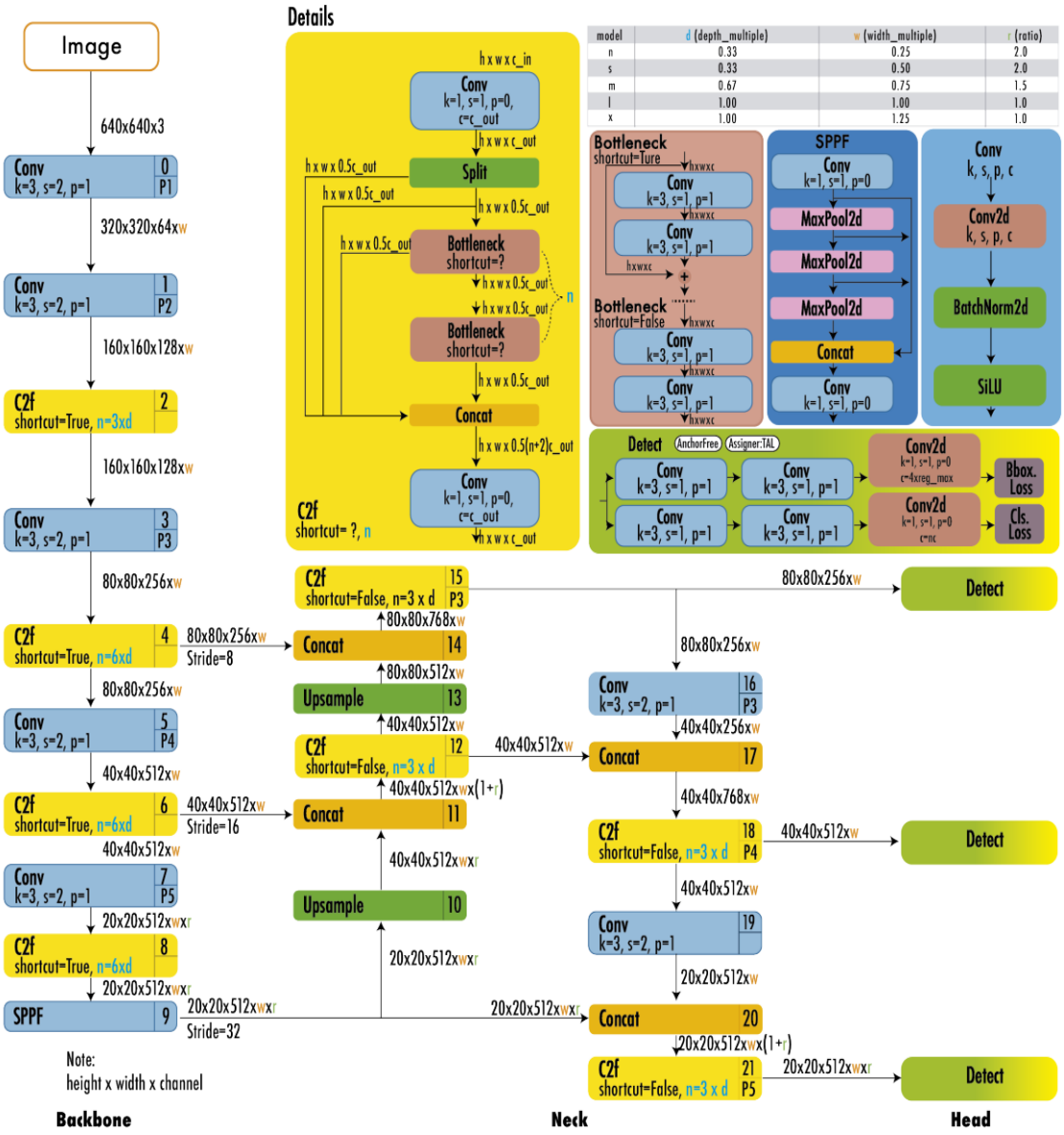

Figure 3.13. YOLOV8 model detailed architecture [43].

3.3.2 YOLO V8 Training

In the training process of YOLOV8 model, the model tries to match the ground truth (Bounding boxes coordinates) with the appropriate cell selected to be tested at the detection process.

The center of the bounding box is defined and then each cell will be compared until getting a match between the test cell and the bounding box. Then, the class prediction of that test cell is changed to match the ground truth appropriate class. The cell's bounding box will also be updated as shown in Figure 3.14.

Figure 3.14. Example of training of YOLOV8 model. (a) Center of bounding box defining.

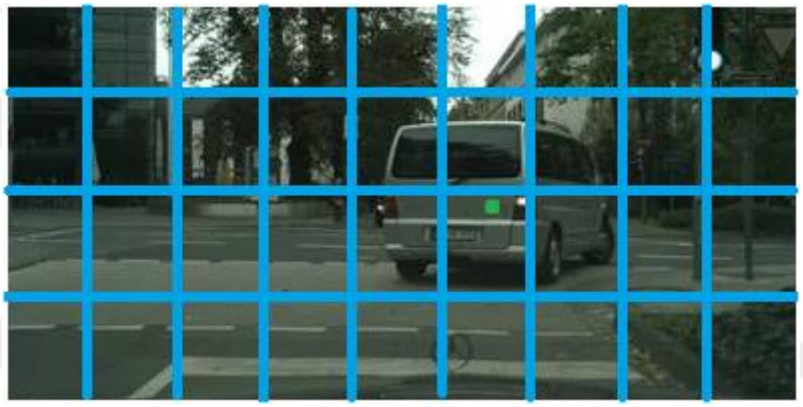

Figure 3.15. (b) Comparing with all cells.

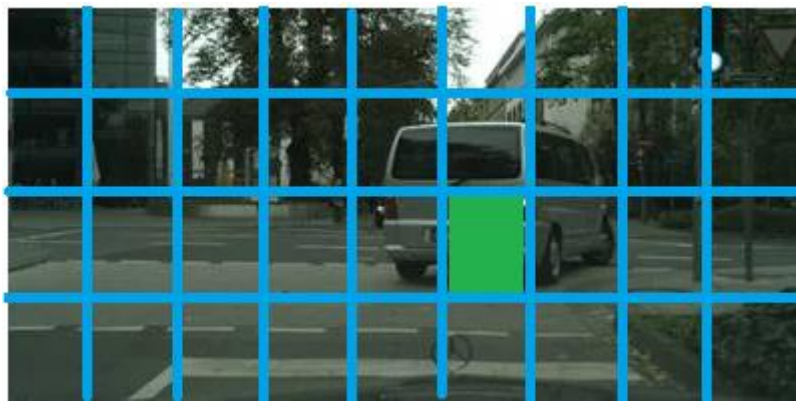

Figure 3.16. (c) Match the corresponding cell.

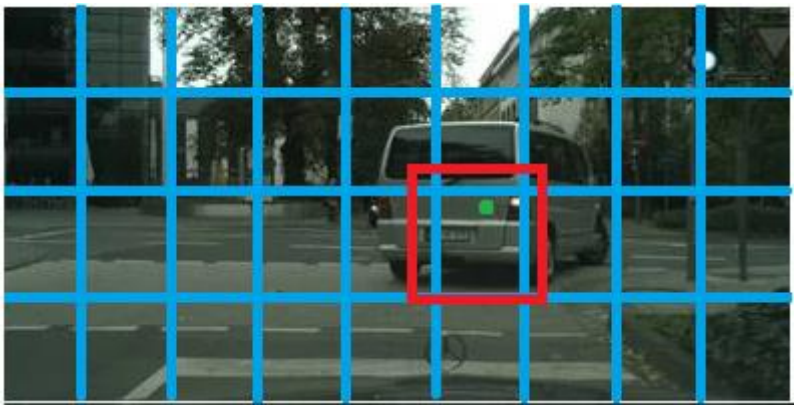

Figure 3.17. (d) Plot the bounding box.

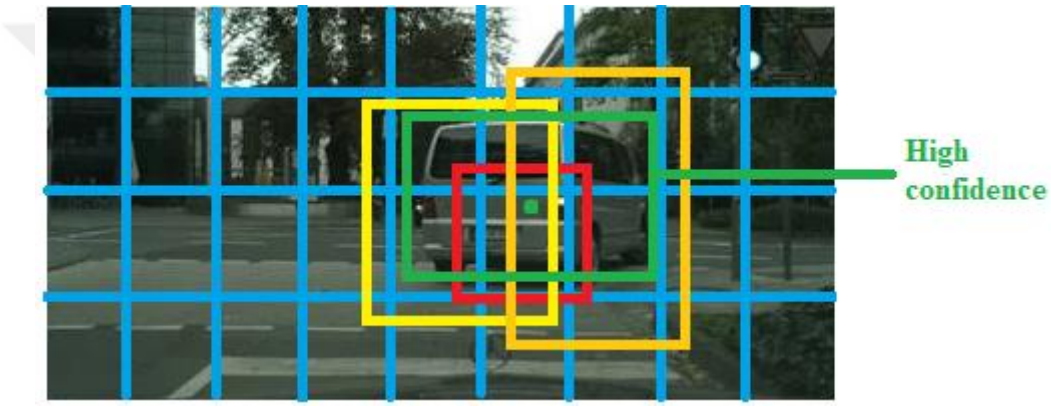

Figure 3.18. (e) Examples of high confidence cell matches.

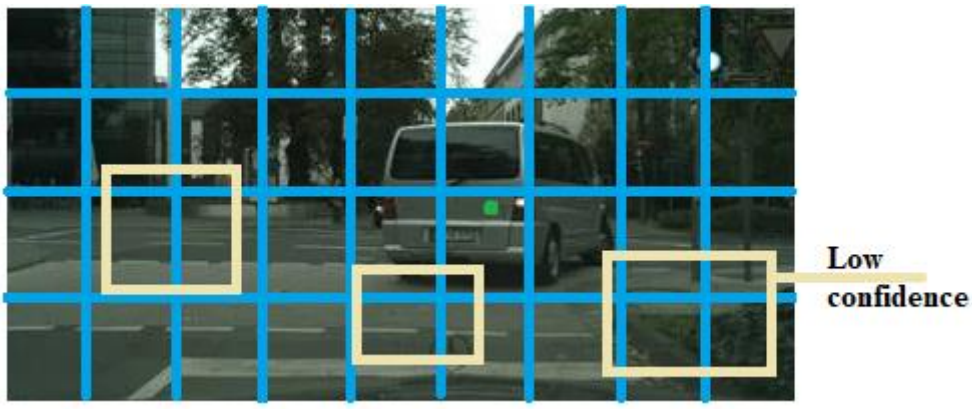

Figure 3.19. (f) Examples of low confidence cell matches.

The operation is repeated with all possible cells and many bounding boxes will be resulted, then, the bounding box with the biggest ***overlap*** with the original ground

truth (original bounding box) will get the highest confidence value. Although other bounding boxes that have less overlapping with the original bounding box will get low confidence value.

3.4 Performance metrics

For our trained YOLOV8 model, different evaluation metrics can be used to assess the models. Not only accuracy and loss for training and validation, but also other metrics can be used to evaluate the performance, and here are these parameters [47, 48]:

Box Loss: a loss that measures how "tight" the predicted bounding boxes (around each object) are to the original ground truth object (supplied with dataset).

Class Loss: Measures the accuracy of the predicted bounding box, class loss will be low if the predicted object inside the BBOX is the correct class.

Distribution Focal DF Loss is used to deal with class imbalance. In some cases, (as in our dataset), some classes will contain more instances than the others.

Precision: This metrics represents the ratio of the true positives TP to the total positive predictions and is denoted as TP/(TP+FP), where FP represents the false positives (incorrectly accepted regions inside the BBOX).

Recall: The ratio of true positives to the number of objects detected.

mAP50 mean Average Precision with a detection threshold of 50%. It includes trade-off between precision and recall since it takes into account both false positives and false negatives.

mAP50-95 mean Average Precision with detection thresholds from 50% to 95%.

3.5 The proposed support-decision algorithm

In the following, we will illustrate the decision-support algorithm used in this study steps:

The algorithm contains the following steps:

1- Load the trained YOLOV8 model and the test images.

```
model = YOLO('/content/drive/MyDrive/best.pt')
# Get a list of all validation image paths
val_image_paths = [f for f in
os.listdir(os.path.abspath(VAL_PATH)) if
f.endswith('.jpg')]
```

```
selected_images = random.sample(val_image_paths, 10)
```

2- Preprocess test images:

   a. resize to 640*640

   b. Normalize (divide by 255)

   c. Convert to tensor (to be fed into YOLOV8 model)

```
 img = cv2.imread(os.path.join(os.path.abspath(VAL_PATH),
image_path))
# Convert the image to RGB
img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img_rgb = cv2.resize(img_rgb,(640,640))
img_tensor = torch.from_numpy(img_rgb).permute(2, 0,
1).float() / 255.0
    img_tensor = img_tensor.unsqueeze(0)
```

3- Predict the model using YOLOV8 model

```
results = model.predict(img_tensor)
```

4- Initialize all results matrixes:

```
# Initialize a dictionary to store the class names and
probabilities
    class_probs = []
    # Initialize counters for cars and persons
    car_count = 0
    person_count = 0
    truck_count = 0
    motor_count = 0
    bus_count = 0
    bike_count = 0
    rider_count = 0
    # Initialize a dictionary to store the class names and
probabilities
    class_probs = []
    near_count = 0
    moderate_count = 0
    # Initialize counters for objects on the left and right
    left_count = 0
    right_count = 0
```

5- Run the decision-support code

   a. Iterate the algorithm as long as there is bounding boxes in the prediction results of the YOLO V8 model:

b. Extract the bounding box coordinates, the predicted class of the object, and the confidence level of the detected object.
c. If more than half of the width of the detected object is in the left part of the image, then the object is mostly located in the left part of the image so, the left_count is increased, else the right_count is increased.
d. Increase the counter of the class corresponding to the detected class by the YOLO V8 model.
e. Choose a color for each detected class so the bounding box and the confidence value will be colored with this color on the final results.
f. Compute the area of the bounding box, and the area percentage which is the area of the component (bounding box) divided by the total area of the image (height*width), to know the size of the component which will help us know if the component is near or far.
g. Decide if the object near or far: if the area percentage of the component is less than 0.01, it's considered far, else if it's less than 0.05 but higher than 0.01, it's considered as moderate, else, it will be considered as near (the area is big).
h. View image, plot the bounding box, and title it with the confidence score.

```
# Draw the bounding boxes on the image
    for result in results:
        # Each result is a 'Results' object containing the
detections for one image
        for box, conf, cls in zip(result.boxes.xyxy,
result.boxes.conf, result.boxes.cls):
            if ((box[0] + box[2]) / 2) <
img_tensor.shape[3] / 2:
                left_count += 1
            else:
                right_count += 1
            if model.names[int(cls)] == 'car':
              car_count += 1
            elif model.names[int(cls)] == 'person':
              person_count += 1
            elif model.names[int(cls)] == 'truck':
              truck_count += 1
            elif model.names[int(cls)] == 'bike':
              bike_count += 1
            elif model.names[int(cls)] == 'bus':
              bus_count += 1
```

```
            elif model.names[int(cls)] == 'motor':
              motor_count += 1
            elif model.names[int(cls)] == 'rider':
              rider_count += 1
            # Get the color for this class
            color = colors[model.names[int(cls)]]
            # Draw the bounding box
            cv2.rectangle(img_rgb, (int(box[0]),
int(box[1])), (int(box[2]), int(box[3])), color, 2)
            # Draw the label and score
            cv2.putText(img_rgb, f'[model.names[int(cls)]]:
[conf:.2f]', (int(box[0]), int(box[1])-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, color, 2)
            # Add the class and probability to the
dictionary
            if model.names[int(cls)] in class_probs:
                class_probs[model.names[int(cls)]].append(c
onf.item())
            else:
                class_probs[model.names[int(cls)]] =
[conf.item()]
            # If the class is a car or a person, increment
the appropriate counter
            if model.names[int(cls)] in ['car',
'person','truck','bus','bike','motor','rider']:
              # Compute the area of the bounding box
              box_area = (box[2] - box[0]) * (box[3] -
box[1])
              # Compute the area as a percentage of the
total image area
              area_percentage = box_area /
(img_tensor.shape[2] * img_tensor.shape[3])
              # Estimate the distance based on the area
percentage
              if area_percentage < 0.01:  # This is just an
example threshold
                  distance = 'far'
              elif area_percentage < 0.05:
                  distance = 'moderate'
                  moderate_count+=1
              else:
                  distance = 'near'
                  near_count+=1
              print(f'[model.names[int(cls)]] detected:
area = [box_area:.2f] pixels, area percentage =
[area_percentage:.2%], estimated distance = [distance]')
```

6- Define the final traffic status and the final driving decision:

a. Compute the total count of all components in the scene as total_count

i. If total_count>15 the traffic is considered to be heavy

ii. Else if the total_count >8 and total_count<=14 the traffic is considered to be moderate.

iii. Else the traffic is light.

```
    # Determine the traffic status based on the counts
    total_count=car_count + person_count + rider_count +
truck_count + bus_count + bike_count
    if total_count > 15:  # These are just example
thresholds
        traff_stat = 'heavy'
    elif total_count > 8:
        traff_stat = 'moderate'
    else:
        traff_stat = 'light'
    print(f'Traffic status: [traff_stat]')
```

b. Produce the final driving decision:

i. If number of near components >2 then the decision is to “stop now”.

ii. Else if the number of moderate components >5 or number of near components >0 and <2 or the traffic status is heavy then decision will be “***slow down immediately***”.

iii. Else if the traffic status is moderate then the decision is “slow down a little”.

iv. Else if the traffic status is light, then the decision will be “speed up a little”.

```
    if near_count>2:
        decision = 'stop now'
    elif moderate_count>5 or near_count>0 or
traff_stat=='heavy':
        decision='slow down immediately'
    elif traff_stat=='moderate':
        decision='slow down a little'
    elif traff_stat=='light':
        decision='speed up a little'
    print(f'Decision is to : [decision]')
```

c. Produce the preferred direction to be directed to by the driver by examining the left_count and the right_count variables. If the left_count is bigger than the right_count, then the notice that will be delivered to the driver will be “right has less traffic” else the notice will be “left has less traffic”.

```
    if left_count > right_count:
        decision = 'right has less traffic'
    elif right_count > left_count:
        decision = 'left has less traffic'
    else:
        decision = 'move forward'

    print(f'Decision is to: [decision]')
```

7- Show the image with all bounding boxes, their classes, their confidence values. Besides this, print all information about number of detected component of each class, area percentage, statement describes if the component is near or far, the traffic status, the final driving decision, and the notice about the left and right directions.

```
    # Display the image
    plt.imshow(cv2.cvtColor(img_rgb, cv2.COLOR_BGR2RGB))
    plt.show()
    # Print the probabilities for each class
    for cls, probs in class_probs.items():
        print(f'[cls]:')
        print('probabilities:', ', '.join([f'[prob:.2f]'
for prob in probs]))
```

The decision-support algorithm is the main contribution of our work which is the most essential benefit of computer vision detection methodologies.

The algorithms will be tested using test images to ensure the accuracy of detection and decision-support.

CHAPTER 4. IMPLEMENTATION AND RESULTS

4.1 Test Conditions

In this chapter, the implementation of the proposed methodology, the experiments, and the results will be introduced.

The main training scenario can be described as follows:

1- The training set will be used to train the most recent version of YOLO model (YOLOV8) to learn distinguishing between multiple classes: “Bike”, “Bus”, “Car”, “Motor”, “Person”, “Rider”, “Traffic Light”, “Traffic Sign”, “Train”, and “Truck”.

2- The trained YOLOV8 model will be used then as the core part of our decision support algorithm.

3- Scene capturing is the next step in which multiple images of the scene in front of the car will be captured and fed into the YOLOV8 model.

4- The prediction of each image resulted from the YOLOV8 model will be taken for the ***traffic analysis*** step.

5- The decision support algorithm will be then applied to help the drivers making their driving decisions based on the ***traffic analysis*** of the output predictions of the YOLOV8 model.

6- Scenario steps will be repeated for different ***test cases*** and the results will be compared and judged to see the efficiency of the proposed decision support system.

First, the training and validation results of the trained YOLOV8 model will be presented, then the decision support system results will be introduced and discussed. Finally, for more reliability, the current study's outcomes will be compared with the related state-of-art methodologies.

4.2 Results of training YOLOV8 model

Table 4.1 shows the training and validation loss (class loss: cls_loss, dfl_loss: distribution focal loss, and box loss). The loss is decreased epoch by epoch until reaching a minimum value. The validation box loss proves that the YOLOV8 model is able to accurately localize the objects (car, bus, person, etc.) within the bounding boxes. Moreover, the validation DFL loss focusses on the class imbalance of the dataset (which is actually exists in our dataset since some classes contain huge number of samples, while others don't), so the DFL loss of the trained YOLOV8 model is decreased epoch by epoch indicating that the model tries to improve the minor classes detection operation. For the class loss, which is also decreased through the training process, it indicates that the trained YOLOV8 model is getting better in predicting the right class of the detected objects. In total, the trained YOLOV8 model losses decreased through training indicating a good training process.

Table 4.1 Training and validation.

Epoch	Train box_loss	Train cls_loss	Train dfl_loss	Val box_loss	Val cls_loss	Val dfl_loss
1	1.4169	1.2516	1.0454	1.3922	0.93175	1.0171
2	1.3885	0.93359	1.0332	1.3745	0.87064	1.0076
3	1.3813	0.91551	1.027	1.38	0.8808	1.0188
4	1.3774	0.90348	1.026	1.3638	0.8527	1.0061
5	1.3604	0.87937	1.0184	1.3405	0.83254	0.99866
6	1.3436	0.8576	1.014	1.3403	0.80647	0.99717
7	1.3299	0.83879	1.0067	1.3242	0.79473	0.98827
8	1.3203	0.82713	1.0025	1.3137	0.78771	0.98518
9	1.3098	0.81176	0.99729	1.3122	0.77096	0.98107

10	1.3021	0.80146	0.99419	1.3054	0.76545	0.97874
11	1.3398	0.80441	1.0012	1.3044	0.77087	0.98152
12	1.3341	0.79421	0.99829	1.2969	0.76278	0.97558
13	1.3248	0.78192	0.99324	1.296	0.75154	0.97466
14	1.3173	0.77166	0.99102	1.2894	0.74864	0.97091
15	1.3082	0.75983	0.9876	1.284	0.73922	0.9701
16	1.3017	0.75076	0.98508	1.2828	0.73704	0.96999
17	1.2956	0.74146	0.98196	1.2776	0.73062	0.96827
18	1.2886	0.73086	0.978	1.2766	0.72655	0.96617
19	1.2804	0.7223	0.97599	1.2706	0.72203	0.96489
20	1.2716	0.71205	0.9723	1.2685	0.71887	0.9638

For more declarations, the focal loss, class loss and box loss curves are illustrated in Figure 4.1-Figure 4.3.

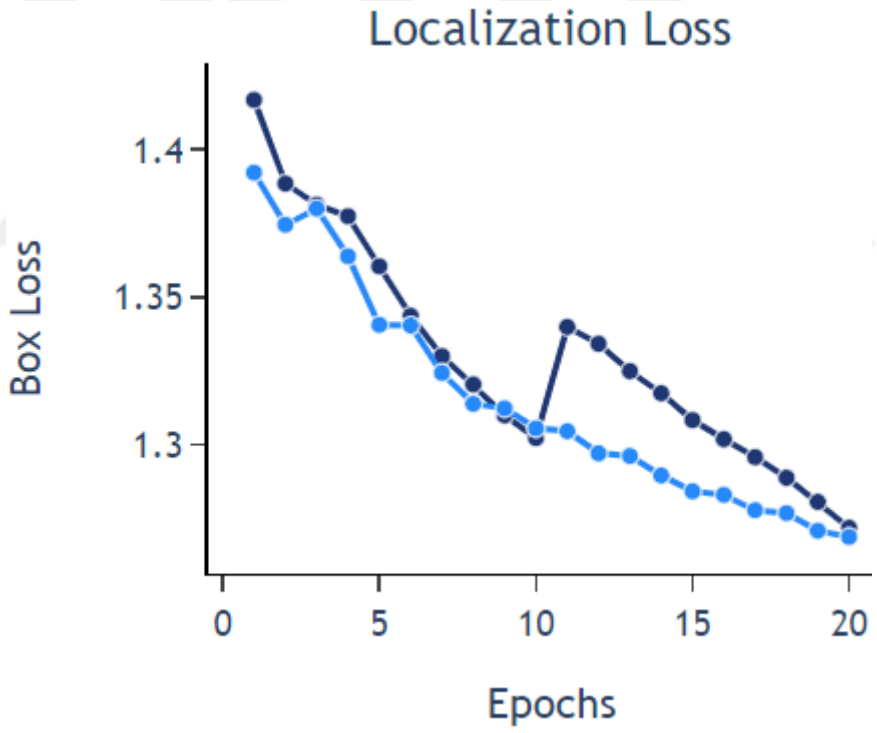

Figure 4.1. Box loss of the trained YOLOV8 model

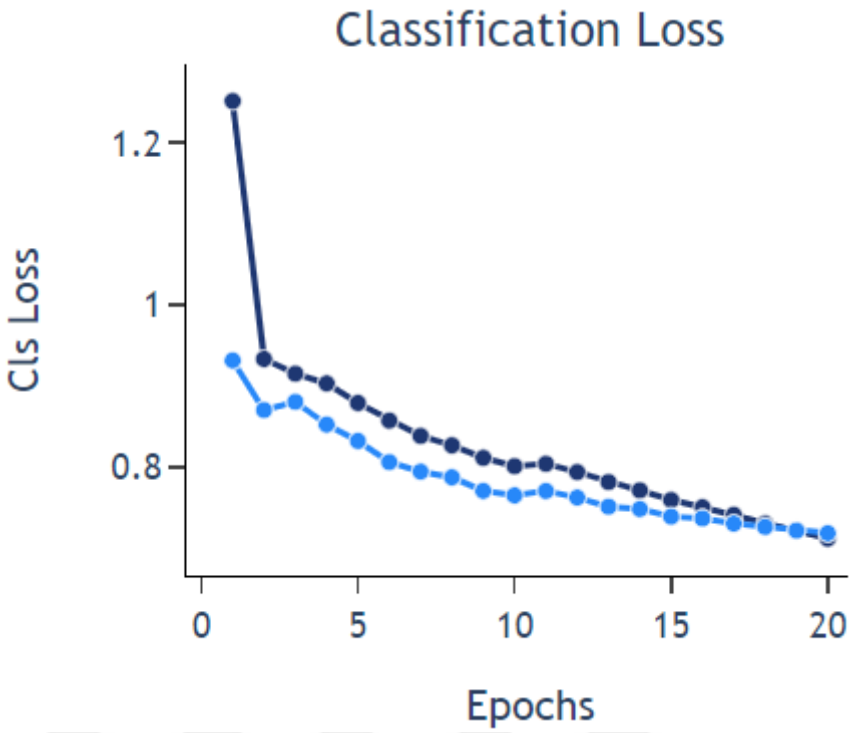

Figure 4.2. Classification loss of the trained YOLOV8 model

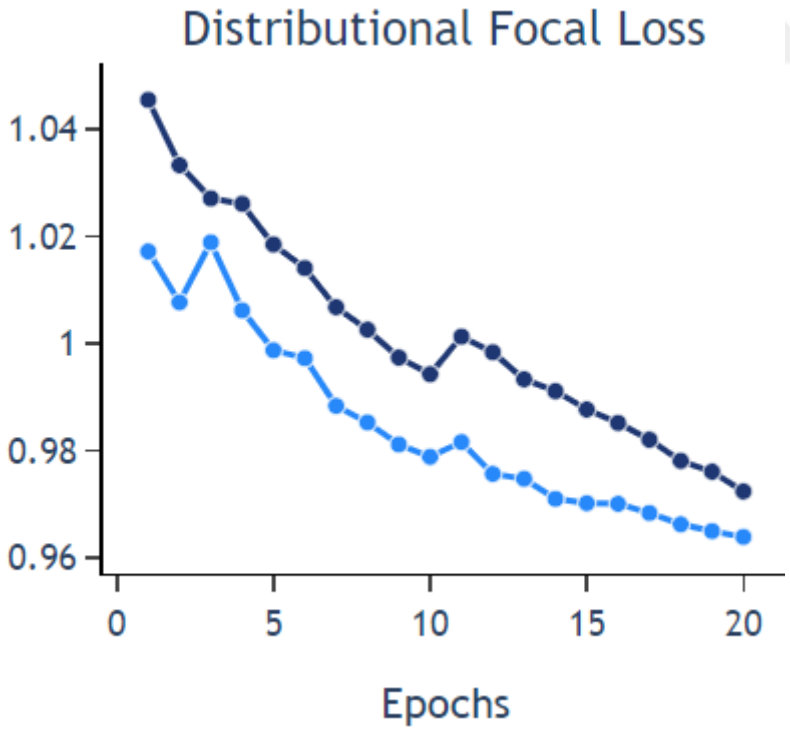

Figure 4.3. DFL loss of the trained YOLOV8 model

The precision, recall, mAP-50 and mAP50-95 are also computed and shown in Table 4.2 and Figure 4.4-Figure 4.7.

Table 4.2 Validation results.

Epoch	Train box_loss	Train cls_loss	Train dfl_loss
1	0.56622	0.34438	0.35174
2	0.50201	0.38158	0.40051
3	0.528	0.36966	0.39584
4	0.53153	0.38109	0.41151
5	0.65163	0.39886	0.42243
6	0.66171	0.40858	0.43631
7	0.66646	0.39707	0.44382
8	0.67194	0.42286	0.45357
9	0.6717	0.42034	0.46203
10	0.69143	0.43155	0.46837
11	0.68862	0.43508	0.47072
12	0.72265	0.42987	0.47769
13	0.71169	0.43584	0.48074
14	0.69442	0.43537	0.48269
15	0.70819	0.44124	0.49245
16	0.68951	0.44661	0.49254
17	0.71585	0.45011	0.49506
18	0.6843	0.45504	0.49474
19	0.72553	0.44892	0.50249
20	0.71167	0.45761	0.50076

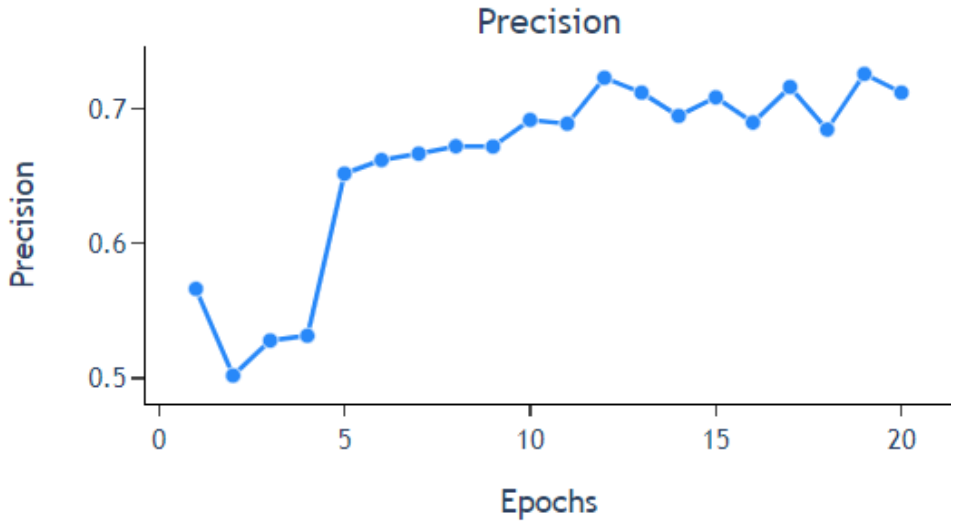

Figure 4.4. Precision of the trained YOLOV8 model

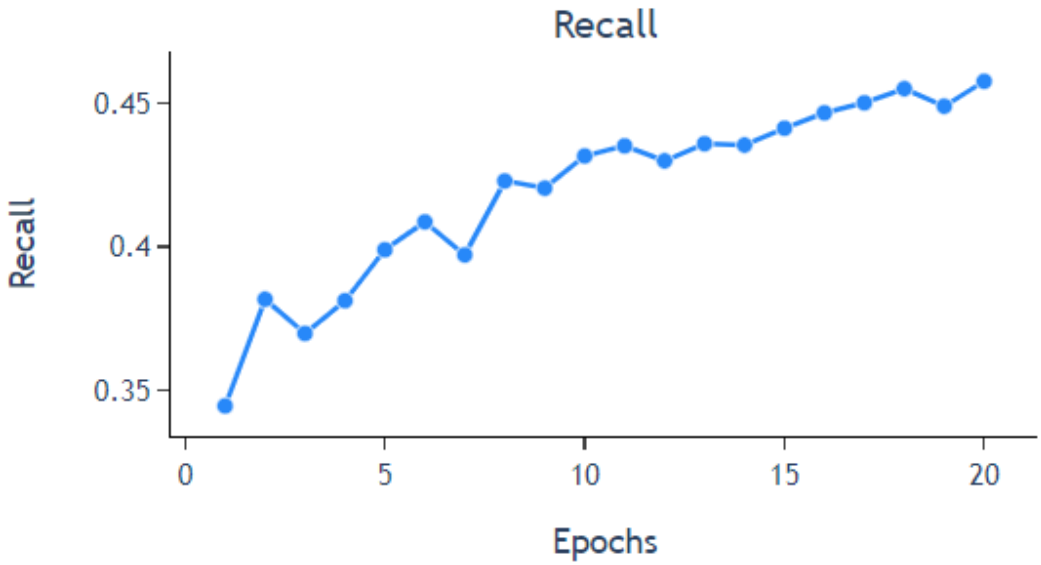

Figure 4.5. Recall of the trained YOLOV8 model

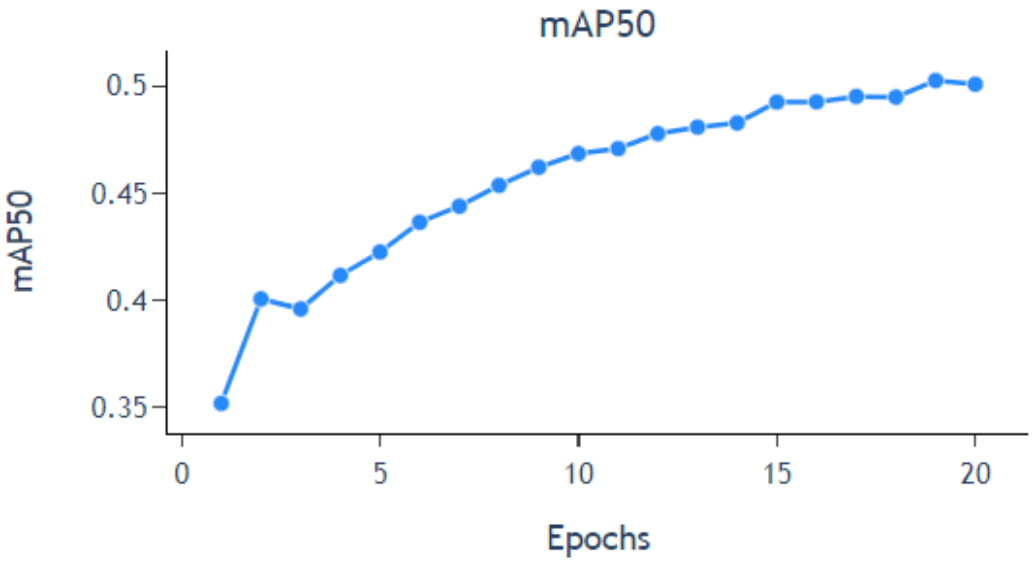

Figure 4.6. mAP50 of the trained YOLOV8 model

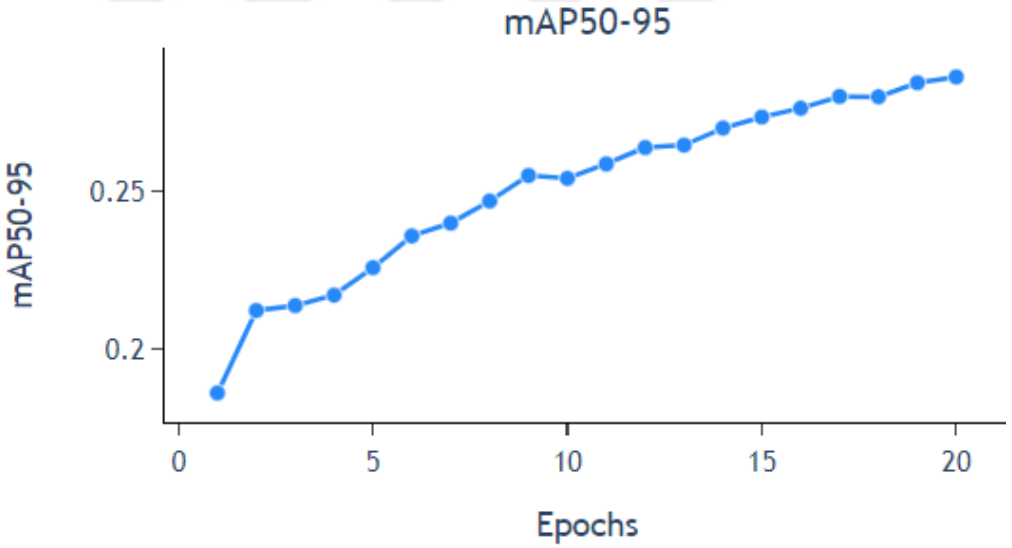

Figure 4.7. mAP50-95 of the trained YOLOV8 model

Figure 4.4-Figure 4.7 show that the precision, recall, mAP50 and mAP50-95 are increased epoch by epoch. The final values are 0.71167, 0.45761, 0.5, and 0.286, respectively.

4.2.1 Detailed calculations of the performance metrics

For more declarations, the precision, recall, mAP50 and mAP50-95 are computed for each class in the dataset (for each detected object). The detailed results of each class is illustrated in Table 4.3.

Table 4.3 Per-class precision, recall, mAP50 and mAP50-95 values

Class	Images	Instances	Precision	Recall	mAP50	mAP50-95
ALL	2000	37075	0.713	0.45	0.502	0.287
Bike	2000	208	0.577	0.447	0.446	0.23
Bus	2000	305	0.672	0.496	0.548	0.423
Car	2000	20525	0.794	0.695	0.77	0.48
Motor	2000	82	0.63	0.415	0.44	0.249
Person	2000	2607	0.738	0.506	0.601	0.306
Rider	2000	116	0.637	0.448	0.46	0.239
Traffic Light	2000	5361	0.721	0.517	0.576	0.217
Traffic Sign	2000	7021	0.72	0.543	0.621	0.33
Train	2000	3	1	0	0	0
Truck	2000	847	0.642	0.511	0.557	0.395

As shown in Table 4.3, classes like “Car”, “Person”, and “Traffic light” are the classes with the top-3 mAP50 values. Indeed, these classes are the most essential classes in our decision support algorithm. Note that the number of images are 2000 of each class, while in each image, there are different number of instances of each class. For example, class “car” contains 20525 instances per through 2000 images (There are many cars within one image).

The confusion matrix (CM) of the detailed correct classifications and misclassifications of each class of the dataset is illustrated in Figure 4.8.

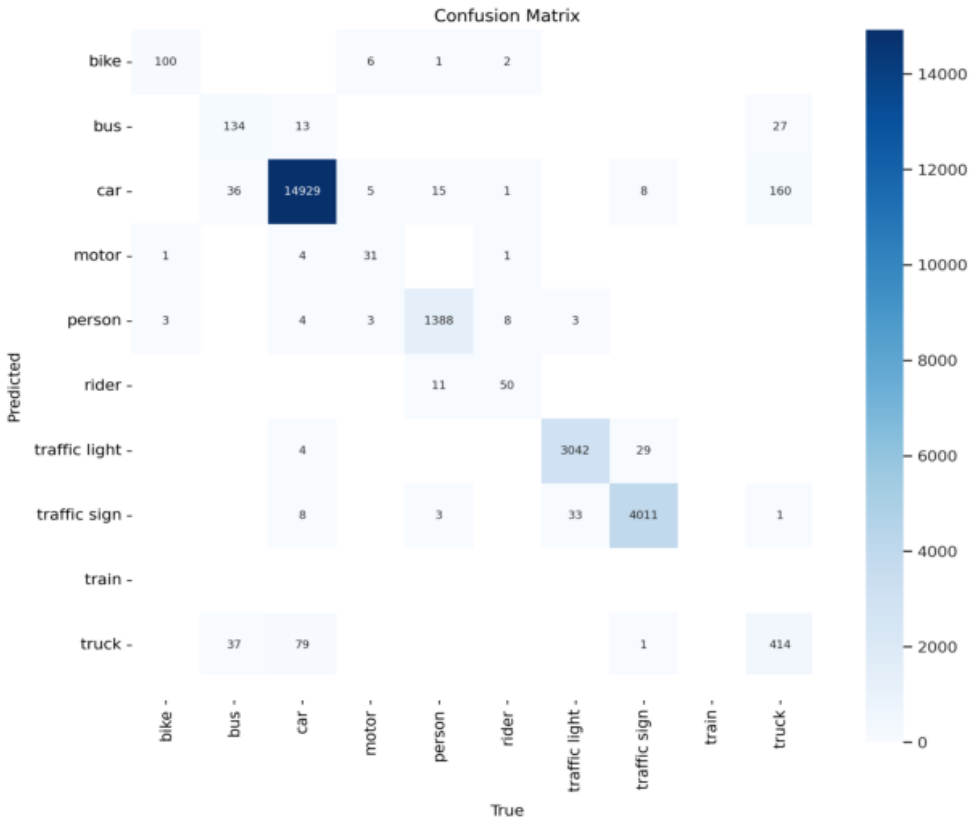

Figure 4.8. Confusion matrix of the test samples of all classes

The precision confidence, recall confidence and precision-recall curves are all illustrated in Figure 4.9-Figure 4.11.

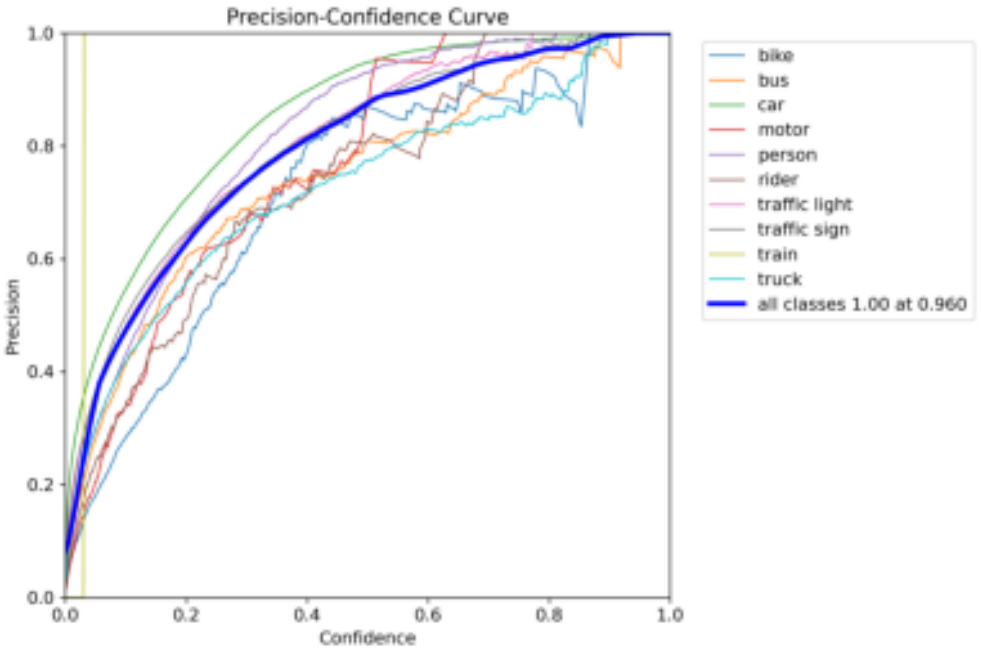

Figure 4.9. Precision-confidence curve.

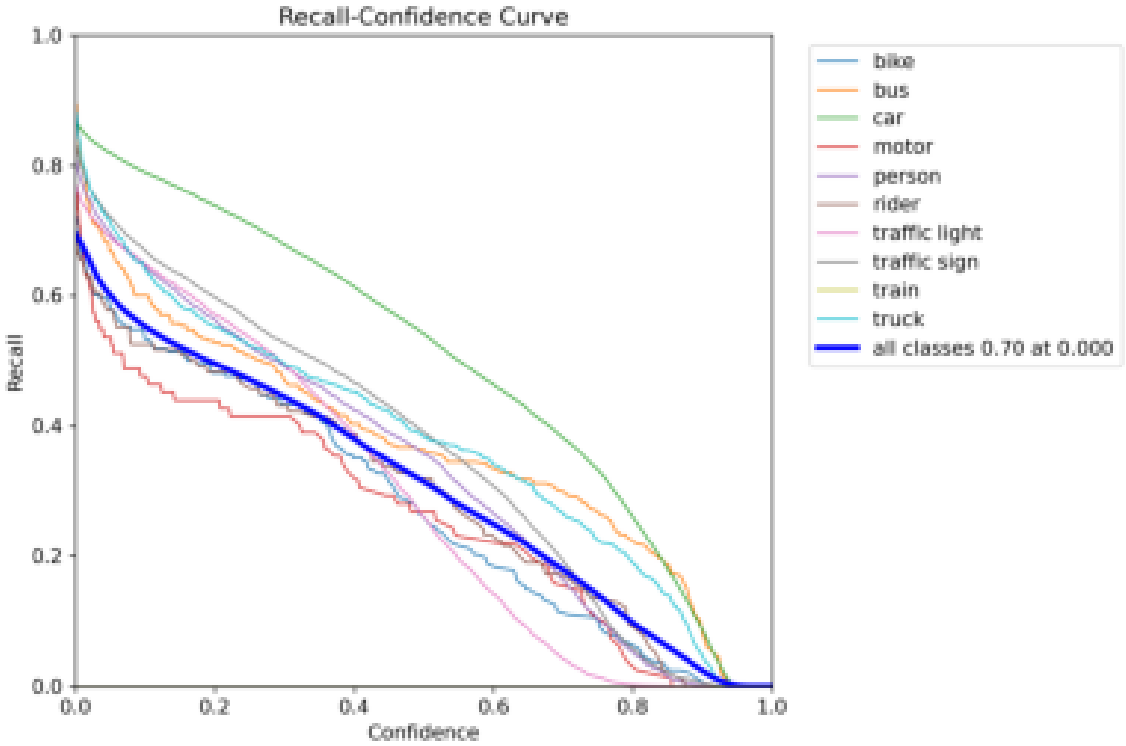

Figure 4.10. Recall-confidence curve.

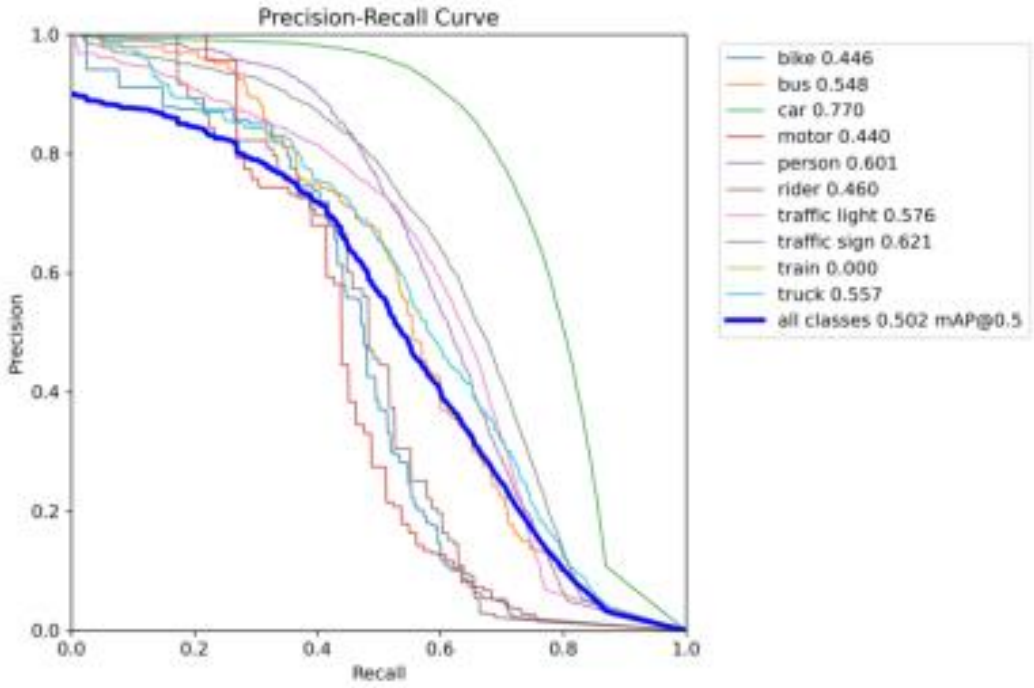

Figure 4.11. Precision-Recall curve.

Figures 4.9-Figure 4.11 illustrate the confidence scores of both precision, recall and precision-recall. The precision confidence curve is an increasing curve, while the recall curve is a decreasing one. The threshold confidence score of precision is perfect 0.9, the confidence score of the recall is 0.7 meaning that the model may miss some true positives objects while detecting objects in the scene. However, this is due to the problem of unbalancing (some classes have too few samples, for example the “train” class has only 3 samples).

4.3 Visual detection results of some samples of the test dataset

Examples of visual detection from the data test set are given below.

Figure 4.12. Test example 5 results for visual detection of test samples.

Figure 4.13. Test example 6 results for visual detection of test samples.

Figure 4.14. Test example 7 results for visual detection of test samples.

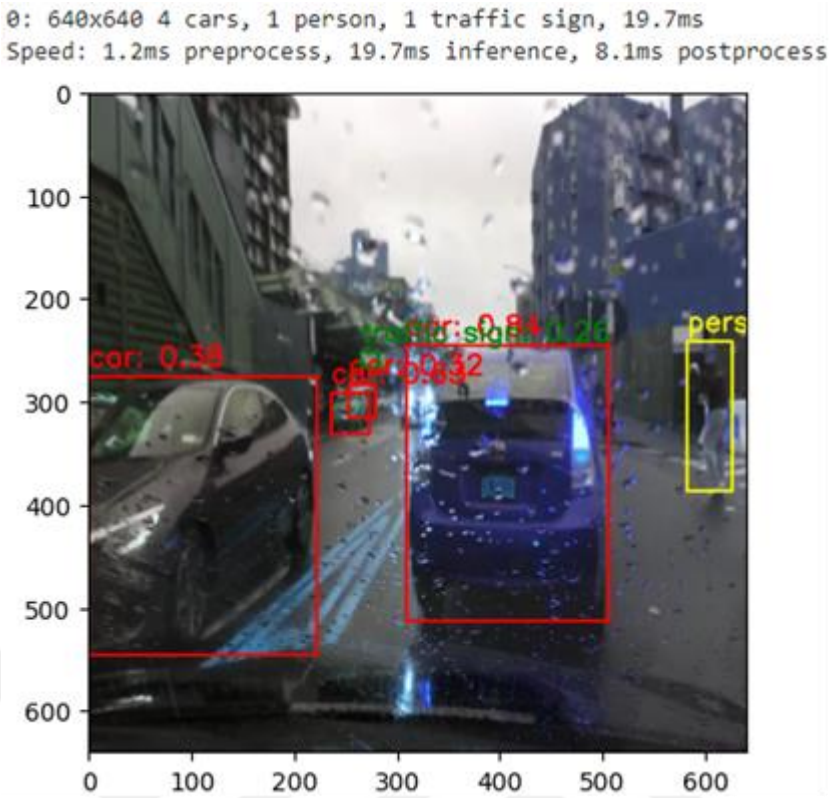

Figure 4.15. Test example 8 results for visual detection of test samples.

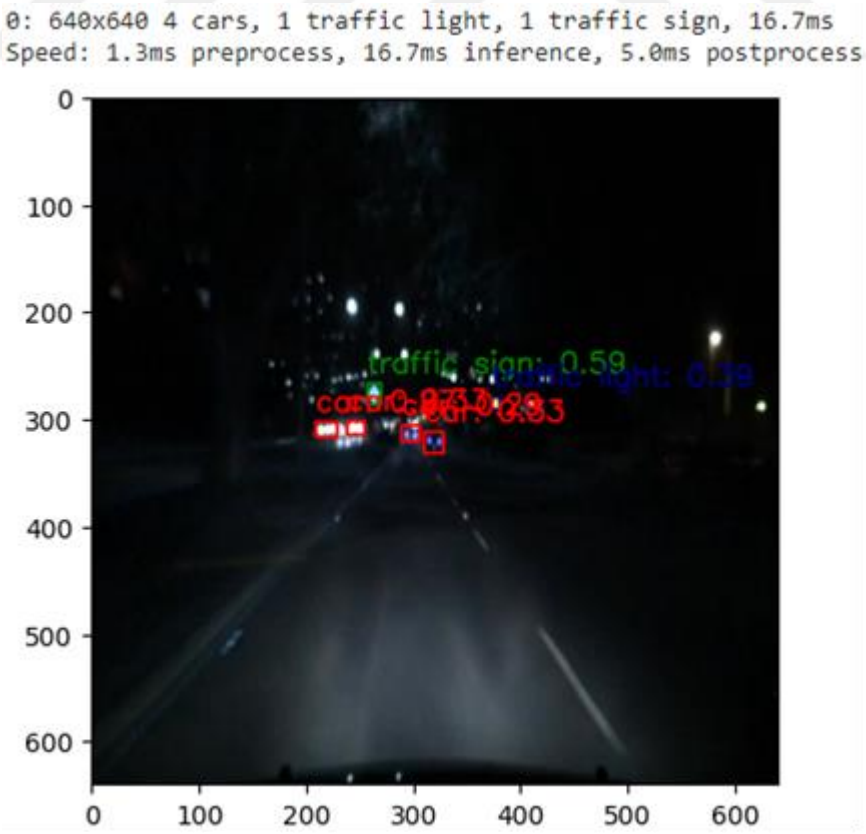

Figure 4.16. Test example 9 results for visual detection of test samples.

Figure 4.12-Figure 4.16 show some of the detection results of some test samples of the test set.

Figure 4.17. Test example 1 results for visual detection of test samples.

Figure 4.18. Test example 2 results for visual detection of test samples.

Figure 4.19. Test example 3 results for visual detection of test samples.

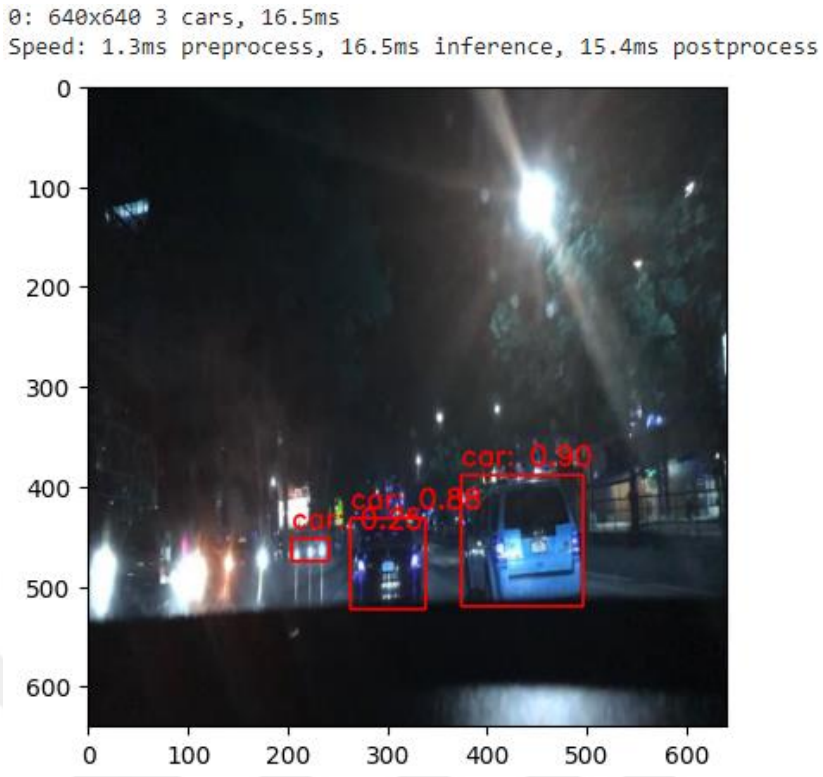

Figure 4.20. Test example 4 results for visual detection of test samples.

Figure 4.21. Test example 5 results for visual detection of test samples.

```
0: 640x640 10 cars, 3 persons, 3 traffic lights, 1 truck, 16.5ms
Speed: 1.2ms preprocess, 16.5ms inference, 5.5ms postprocess per
```

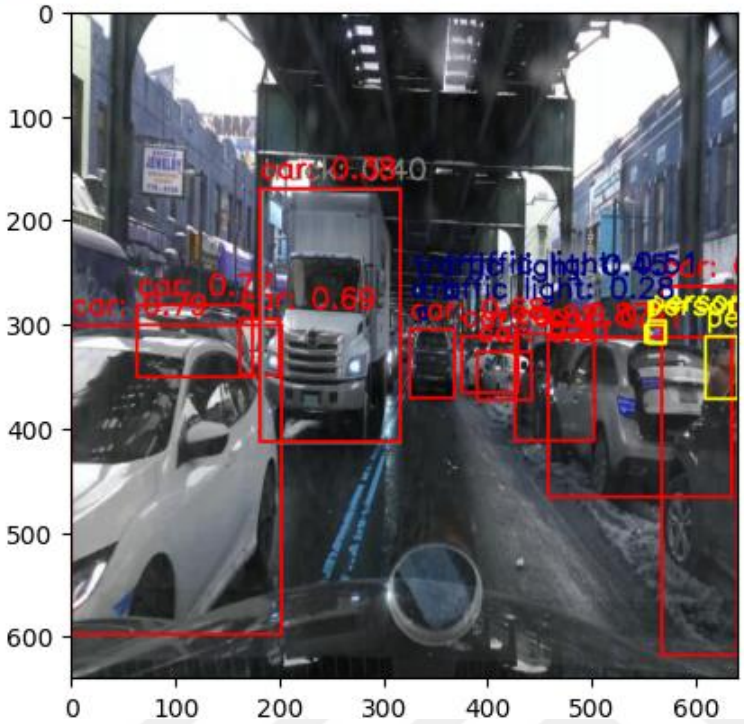

Figure 4.22. Test example 6 results for visual detection of test samples.

```
0: 640x640 5 cars, 16.6ms
Speed: 1.3ms preprocess, 16.6ms inference, 4.7ms postprocess
```

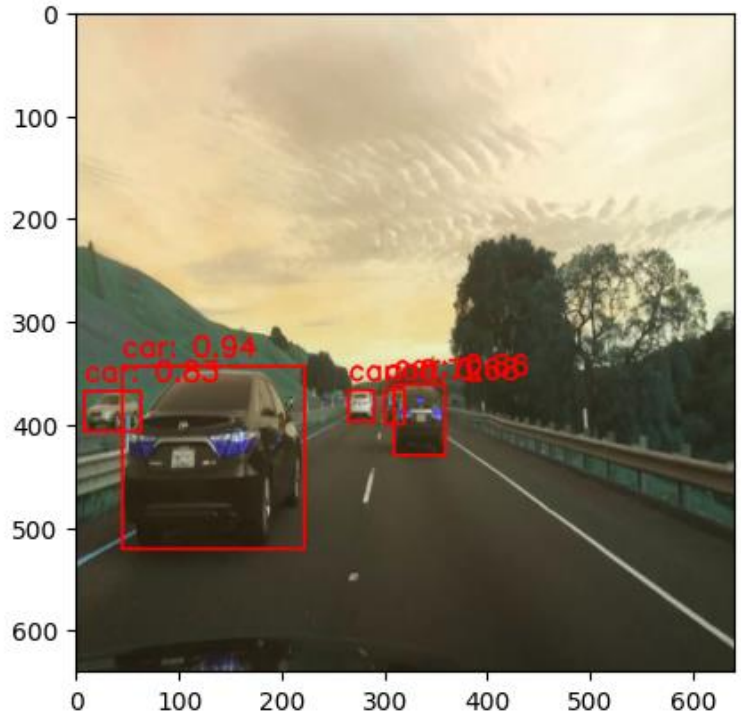

Figure 4.23. Test example 7 results for visual detection of test samples.

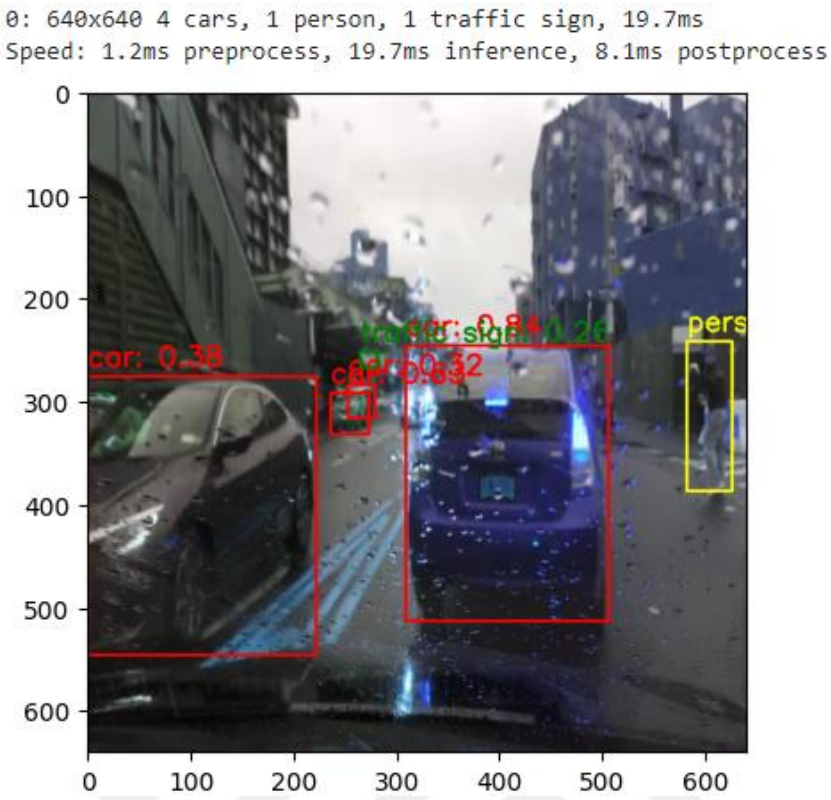

Figure 4.24. Test example 8 results for visual detection of test samples.

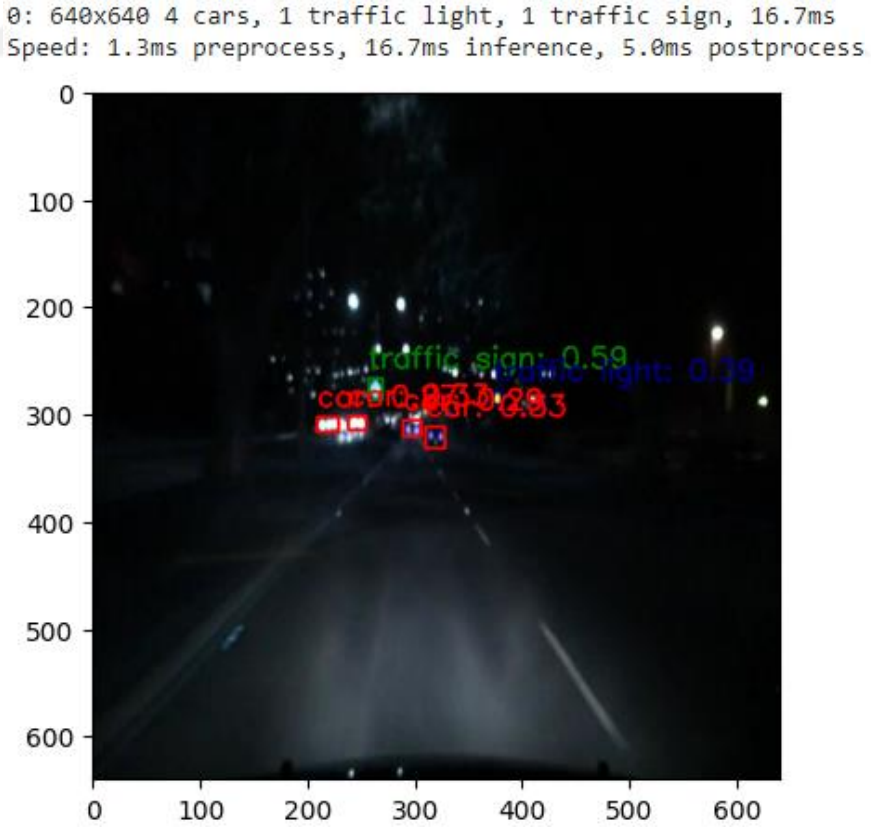

Figure 4.25. Test example 9 results for visual detection of test samples.

Figure 4.17-Figure 4.25 illustrate that the trained YOLOV8 model is capable of detecting all possible objects in front of the camera even in bad environment (low illumination or dark, far distance, busy traffic, etc.).

4.4 Evaluating the decision support algorithm

After detecting objects by the YOLOV8 model, the decision support algorithm is executed on the detection results of the YOLOV8 model.

4.4.1 Example1:

In this example Figure 4.26-Figure 4.27, the detection model decides that the scene contains 8 cars, 6 traffic lights, and 4 traffic signs. The decision support algorithm decides the percentage of the occurrence of each car in the scene. As a result, the decision support algorithm decides that there are 7 faraway cars, and one moderate-distance cars. The algorithm also detects that the traffic light is red. Besides that, the algorithm gives probabilities of each object detected in the scene. The algorithm decides that the traffic status is light. As a result, the algorithm suggests that the driver can speed up a little, and he can go right since the right direction of the street has less traffic.

```
0: 640x640 8 cars, 6 traffic lights, 4 traffic signs, 676.9ms
Speed: 0.0ms preprocess, 676.9ms inference, 5.2ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 5147.53 pixels, area percentage = 1.26%, estimated distance = moderate
car detected: area = 1144.38 pixels, area percentage = 0.28%, estimated distance = far
Traffic light detected: dominant color = red
car detected: area = 623.65 pixels, area percentage = 0.15%, estimated distance = far
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
car detected: area = 1138.94 pixels, area percentage = 0.28%, estimated distance = far
Traffic light detected: dominant color = red
car detected: area = 159.73 pixels, area percentage = 0.04%, estimated distance = far
car detected: area = 189.07 pixels, area percentage = 0.05%, estimated distance = far
car detected: area = 798.20 pixels, area percentage = 0.19%, estimated distance = far
car detected: area = 506.43 pixels, area percentage = 0.12%, estimated distance = far
Traffic light detected: dominant color = red
Traffic status: light
Decision is to : speed up a little
Decision is to: right has less traffic
 car:
 probabilities: 0.71, 0.59, 0.44, 0.37, 0.34, 0.33, 0.32, 0.30
 traffic sign:
 probabilities: 0.59, 0.52, 0.47, 0.34
 traffic light:
 probabilities: 0.53, 0.44, 0.43, 0.37, 0.37, 0.29
```

Figure 4.26. A visual detection results of a test sample (example1) Decision-support system output.

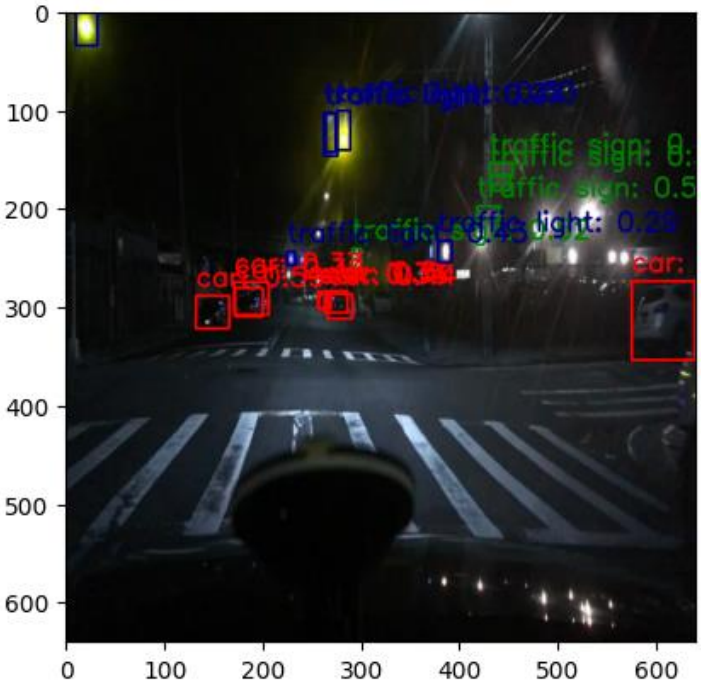

Figure 4.27. A visual detection results of a test sample (example1) Output of detection system (yolo).

4.4.2 Example 2:

In the second example, the algorithm detects 5 cars, 2 traffic lights, and 2 traffic signs. There are two faraway cars, one near car and two moderate-distance cars. The traffic is light but the algorithm advices driver to slow down since there is a near car and he can make an accident. The algorithm also suggests that the left direction has less traffic as shown in Figure 4.28-Figure 4.29.

```
0: 640x640 5 cars, 2 traffic lights, 2 traffic signs, 655.4ms
Speed: 0.0ms preprocess, 655.4ms inference, 5.1ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 18092.80 pixels, area percentage = 4.42%, estimated distance = moderate
car detected: area = 12636.33 pixels, area percentage = 3.09%, estimated distance = moderate
car detected: area = 64343.44 pixels, area percentage = 15.71%, estimated distance = near
car detected: area = 3141.63 pixels, area percentage = 0.77%, estimated distance = far
car detected: area = 1889.39 pixels, area percentage = 0.46%, estimated distance = far
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic status: light
Decision is to : slow down immediately
Decision is to: left has less traffic
car:
probabilities: 0.90, 0.87, 0.85, 0.56, 0.53
traffic sign:
probabilities: 0.64, 0.51
traffic light:
probabilities: 0.39, 0.31
```

Figure 4.28. A visual detection results of a test sample (example2) Decision-support system output.

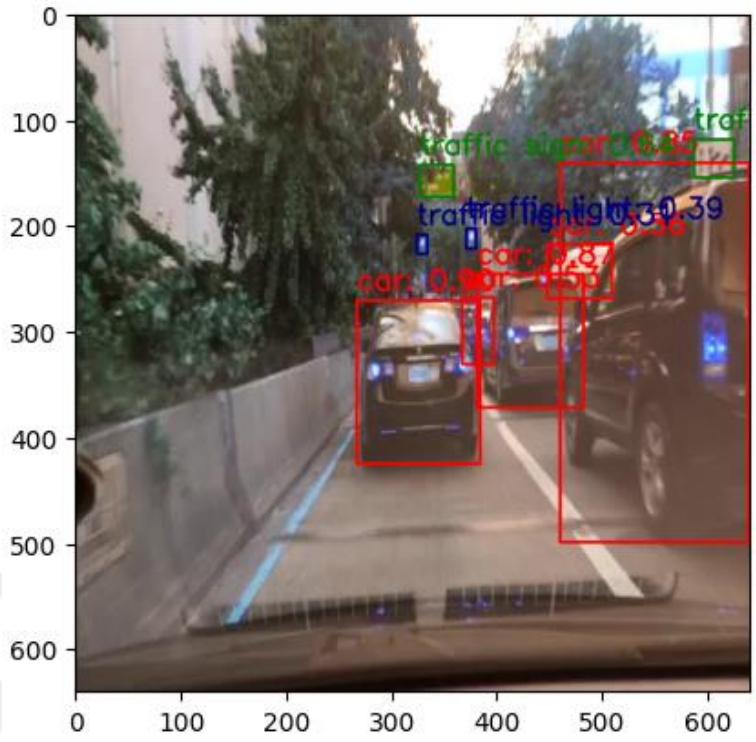

Figure 4.29. A visual detection results of a test sample (example2) Output of detection system (yolo).

4.4.3 Example 3:

In the third example, the algorithm detects 1 car, and 3 traffic lights. There are two faraway cars, one nearby car and two moderate-distance cars. The traffic is light and the algorithm suggests to speed up a little and move forward since the right and left parts of the road both contain cars and trucks Figure 4.30-Figure 4.31.

```
0: 640x640 1 car, 3 traffic signs, 674.0ms
Speed: 0.0ms preprocess, 674.0ms inference, 5.4ms postprocess per image at shape (1, 3, 640, 640
car detected: area = 13057.93 pixels, area percentage = 3.19%, estimated distance = moderate
Traffic status: light
Decision is to : speed up a little
Decision is to: move forward

traffic sign:
probabilities: 0.88, 0.59, 0.47
car:
probabilities: 0.69
```

Figure 4.30. A visual detection results of a test sample (example3) Decision-support system output.

Figure 4.31. A visual detection results of a test sample (example3) Output of detection system (yolo).

4.4.4 Example 4:

In the fourth example, the algorithm detects 8 cars, 4 traffic lights, 6 traffic signs, and 2 trucks. There are 7 faraway cars, one moderate-distance car and two moderate-distance trucks. The traffic status is moderate and the algorithm suggests to slow down a little and move to the right part of the road since it contains less cars and trucks than the left Figure 4.32-Figure 4.33.

```
0: 640x640 8 cars, 4 traffic lights, 6 traffic signs, 2 trucks, 633.1ms
Speed: 0.0ms preprocess, 633.1ms inference, 5.2ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 4437.05 pixels, area percentage = 1.08%, estimated distance = moderate
car detected: area = 1617.09 pixels, area percentage = 0.39%, estimated distance = far
car detected: area = 1044.51 pixels, area percentage = 0.26%, estimated distance = far
car detected: area = 1198.96 pixels, area percentage = 0.29%, estimated distance = far
truck detected: area = 6770.97 pixels, area percentage = 1.65%, estimated distance = moderate
car detected: area = 3110.53 pixels, area percentage = 0.76%, estimated distance = far
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
truck detected: area = 8053.42 pixels, area percentage = 1.97%, estimated distance = moderate
car detected: area = 207.31 pixels, area percentage = 0.05%, estimated distance = far
car detected: area = 795.10 pixels, area percentage = 0.19%, estimated distance = far
car detected: area = 159.36 pixels, area percentage = 0.04%, estimated distance = far
Traffic light detected: dominant color = red
Traffic status: moderate
Decision is to : slow down a little
Decision is to: right has less traffic
car:
probabilities: 0.86, 0.72, 0.71, 0.70, 0.61, 0.30, 0.29, 0.28
traffic sign:
probabilities: 0.66, 0.51, 0.49, 0.45, 0.27, 0.27
truck:
probabilities: 0.65, 0.32
traffic light:
probabilities: 0.45, 0.36, 0.34, 0.27
```

Figure 4.32. A visual detection results of a test sample (example4) Decision-support system output.

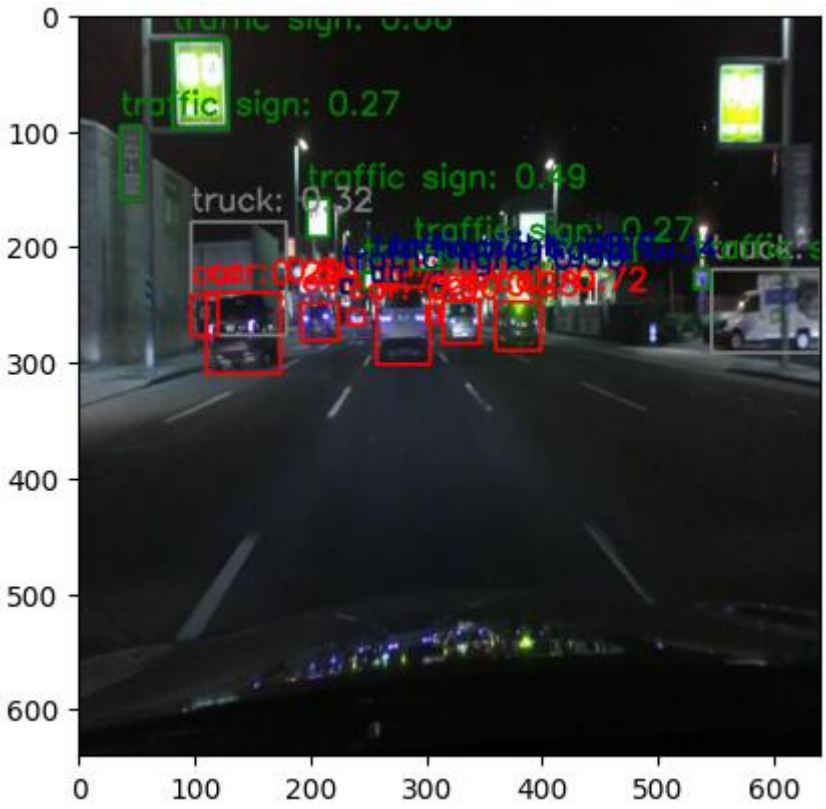

Figure 4.33. A visual detection results of a test sample (example4) Output of detection system (yolo).

4.4.5 Example 5:

In the fifth example, the algorithm detects 7 cars, 1 traffic light, and 1 traffic sign. There are 3 nearby cars, 1 faraway car, and 3 moderate-distance cars.

Although the traffic status is light, but the algorithm suggests to stop the car immediately since there are 3 nearby cars and the driver can extremely make an accident Figure 4.34-Figure 4.35.

```
car:
probabilities: 0.86, 0.72, 0.71, 0.70, 0.61, 0.30, 0.29, 0.28
traffic sign:
probabilities: 0.66, 0.51, 0.49, 0.45, 0.27, 0.27
truck:
probabilities: 0.65, 0.32
traffic light:
probabilities: 0.45, 0.36, 0.34, 0.27

0: 640x640 7 cars, 1 traffic light, 1 traffic sign, 630.9ms
Speed: 0.0ms preprocess, 630.9ms inference, 5.2ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 34133.15 pixels, area percentage = 8.33%, estimated distance = near
car detected: area = 30926.03 pixels, area percentage = 7.55%, estimated distance = near
car detected: area = 13690.66 pixels, area percentage = 3.34%, estimated distance = moderate
car detected: area = 34779.71 pixels, area percentage = 8.49%, estimated distance = near
car detected: area = 10400.80 pixels, area percentage = 2.54%, estimated distance = moderate
car detected: area = 4447.99 pixels, area percentage = 1.09%, estimated distance = moderate
Traffic light detected: dominant color = red
car detected: area = 3199.69 pixels, area percentage = 0.78%, estimated distance = far
Traffic status: light
Decision is to : stop now
Decision is to: right has less traffic
car:
probabilities: 0.92, 0.91, 0.86, 0.73, 0.69, 0.64, 0.32
traffic light:
probabilities: 0.47
traffic sign:
probabilities: 0.26
```

Figure 4.34. A visual detection results of a test sample (example5) Decision-support system output.

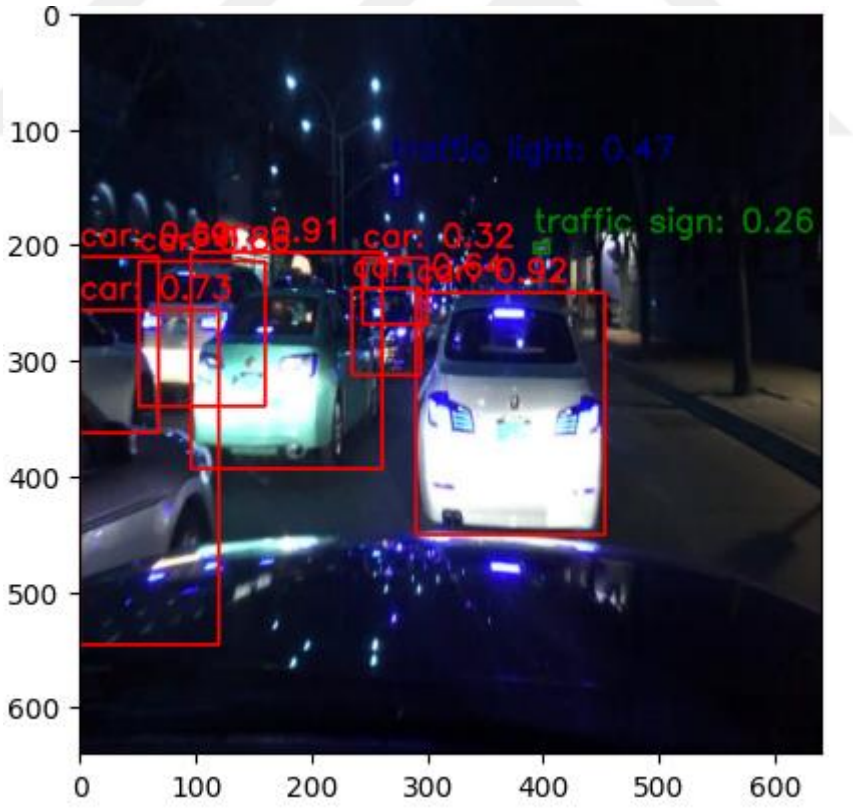

Figure 4.35. A visual detection results of a test sample (example5) Output of detection system (yolo).

4.4.6 Example 6:

In the sixth example, the algorithm detects 6 cars, 2 traffic light, and 1 truck. There are 5 faraway cars and trucks, and 2 moderate-distance cars. The traffic status is light, and the algorithm suggests to speed up a little since there are no nearby cars. The algorithm also suggests that the left has less traffic than the right side of the road (Figure 4.36-Figure 4.37).

```
0: 640x640 6 cars, 2 traffic signs, 1 truck, 618.8ms
Speed: 0.0ms preprocess, 618.8ms inference, 4.8ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 13294.97 pixels, area percentage = 3.25%, estimated distance = moderate
car detected: area = 4817.88 pixels, area percentage = 1.18%, estimated distance = moderate
truck detected: area = 2216.38 pixels, area percentage = 0.54%, estimated distance = far
car detected: area = 2056.95 pixels, area percentage = 0.50%, estimated distance = far
car detected: area = 1188.10 pixels, area percentage = 0.29%, estimated distance = far
car detected: area = 1338.34 pixels, area percentage = 0.33%, estimated distance = far
car detected: area = 160.17 pixels, area percentage = 0.04%, estimated distance = far
Traffic status: light
Decision is to : speed up a little
Decision is to: left has less traffic
```

Figure 4.36. A visual detection results of a test sample (example6). Decision-support system output.

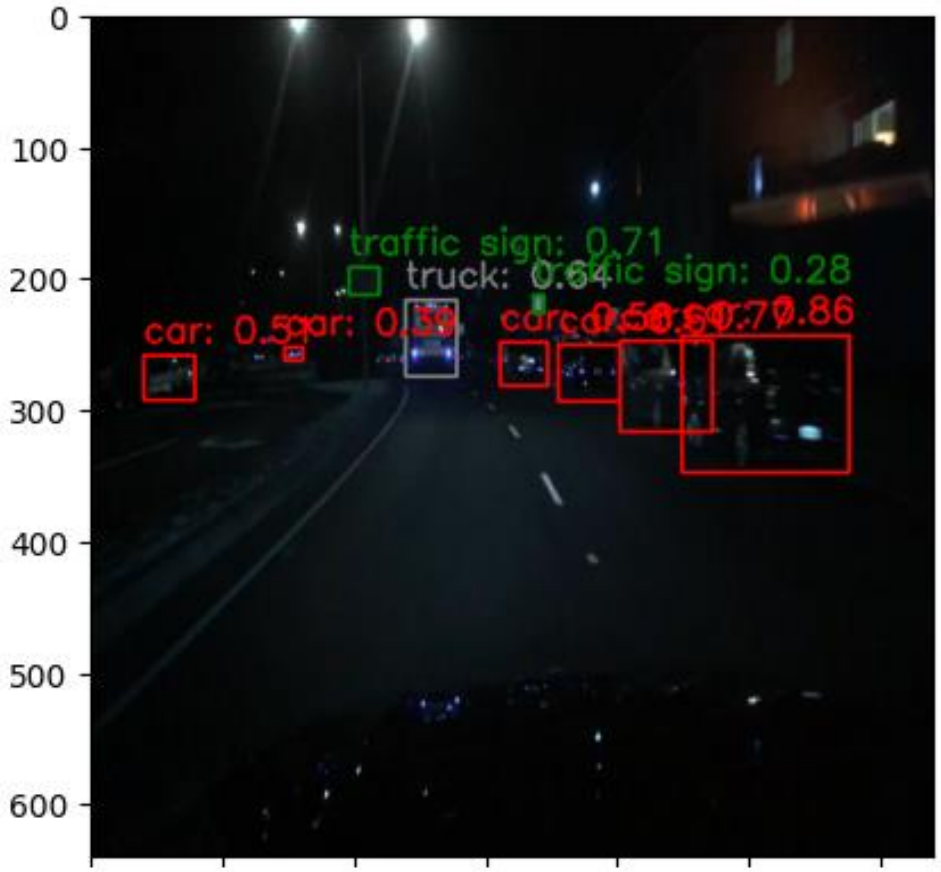

Figure 4.37. A visual detection results of a test sample (example6). Output of detection system (yolo).

4.4.7 Other test samples

This section shows some other test samples and the corresponding responses of the proposed decision support algorithm.

```
0: 640x640 1 car, 7 traffic lights, 3 traffic signs, 602.5ms
Speed: 0.0ms preprocess, 602.5ms inference, 3.9ms postprocess per image at shape (1, 3, 640, 640)
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
car detected: area = 558.96 pixels, area percentage = 0.14%, estimated distance = far
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic light detected: dominant color = red
Traffic status: light
Decision is to : speed up a little
Decision is to: right has less traffic
traffic sign:
probabilities: 0.81, 0.55, 0.44
traffic light:
probabilities: 0.71, 0.64, 0.46, 0.43, 0.41, 0.26, 0.25
car:
probabilities: 0.44
```

Figure 4.38. A visual detection results of a test sample (example7) Decision-support system output.

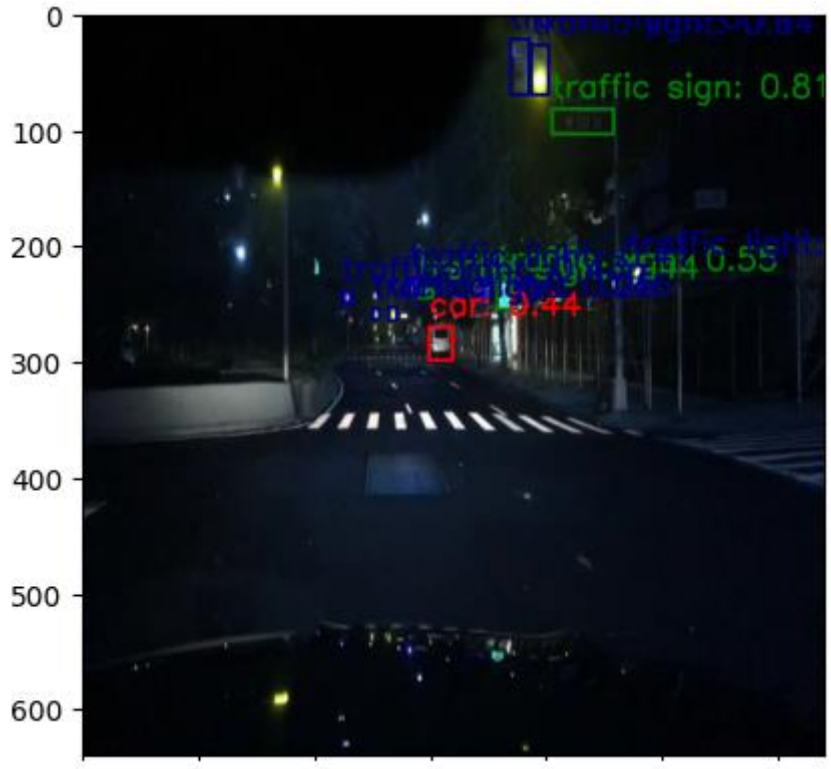

Figure 4.39. A visual detection results of a test sample (example7) Output of detection system (yolo).

```
0: 640x640 1 car, 2 traffic signs, 763.3ms
Speed: 0.0ms preprocess, 763.3ms inference, 5.1ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 13975.67 pixels, area percentage = 3.41%, estimated distance = moderate
Traffic status: light
Decision is to : speed up a little
Decision is to: left has less traffic
car:
probabilities: 0.78
traffic sign:
probabilities: 0.77, 0.66
```

Figure 4.40. A visual detection results of a test sample (example8) (a) Decision-support system output

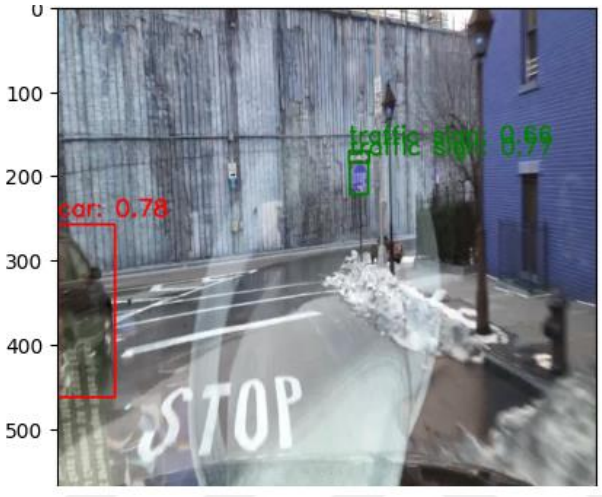

Figure 4.41. A visual detection results of a test sample (example8) (b) Output of detection system (yolo)

```
0: 640x640 9 cars, 1 traffic light, 936.9ms
Speed: 0.0ms preprocess, 936.9ms inference, 5.0ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 6529.69 pixels, area percentage = 1.59%, estimated distance = moderate
car detected: area = 2973.42 pixels, area percentage = 0.73%, estimated distance = far
car detected: area = 2606.38 pixels, area percentage = 0.64%, estimated distance = far
car detected: area = 1168.29 pixels, area percentage = 0.29%, estimated distance = far
car detected: area = 1227.91 pixels, area percentage = 0.30%, estimated distance = far
car detected: area = 1727.83 pixels, area percentage = 0.42%, estimated distance = far
car detected: area = 341.68 pixels, area percentage = 0.08%, estimated distance = far
Traffic light detected: dominant color = red
car detected: area = 721.43 pixels, area percentage = 0.18%, estimated distance = far
car detected: area = 1154.00 pixels, area percentage = 0.28%, estimated distance = far
Traffic status: moderate
Decision is to : slow down a little
Decision is to: right has less traffic

car:
probabilities: 0.78, 0.73, 0.63, 0.60, 0.48, 0.42, 0.32, 0.30, 0.27
traffic light:
probabilities: 0.30
```

Figure 4.42. A visual detection results of a test sample (example9) (a) Decision-support system output

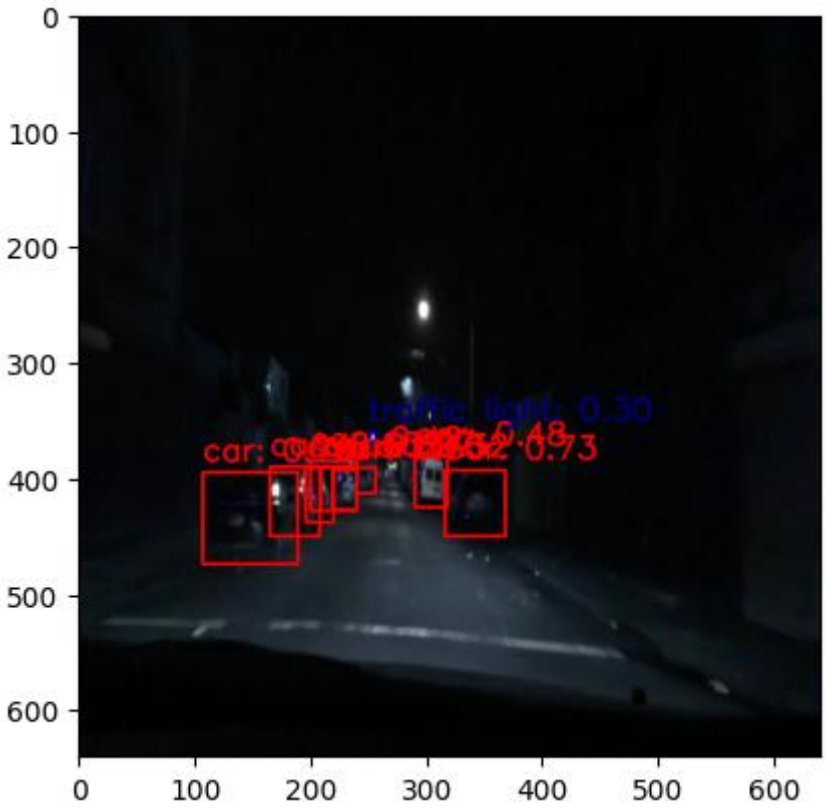

Figure 4.43. (b) Output of detection system (yolo)

```
0: 640x640 6 cars, 2 traffic lights, 3 trucks, 957.3ms
Speed: 0.0ms preprocess, 957.3ms inference, 5.3ms postprocess per image at shape (1, 3, 640, 640)
car detected: area = 5530.12 pixels, area percentage = 1.35%, estimated distance = moderate
car detected: area = 11500.18 pixels, area percentage = 2.81%, estimated distance = moderate
car detected: area = 19494.55 pixels, area percentage = 4.76%, estimated distance = moderate
truck detected: area = 44374.26 pixels, area percentage = 10.83%, estimated distance = near
car detected: area = 3162.88 pixels, area percentage = 0.77%, estimated distance = far
truck detected: area = 21109.80 pixels, area percentage = 5.15%, estimated distance = near
car detected: area = 1191.50 pixels, area percentage = 0.29%, estimated distance = far
Traffic light detected: dominant color = red
truck detected: area = 847.81 pixels, area percentage = 0.21%, estimated distance = far
car detected: area = 583.03 pixels, area percentage = 0.14%, estimated distance = far
Traffic light detected: dominant color = red
Traffic status: moderate
Decision is to : slow down immediately
Decision is to: right has less traffic
car:
probabilities: 0.90, 0.90, 0.89, 0.83, 0.74, 0.32
truck:
probabilities: 0.85, 0.81, 0.47
traffic light:
probabilities: 0.48, 0.32
```

Figure 4.44. Output of A visual detection results of a test sample (example10) (a) Decision-support system output

Figure 4.45. (b) Output of detection system (yolo)

CHAPTER 5. CONCLUSIONS AND FUTURE WORK

5.1 Summary

In this study, a new decision-support algorithm for traffic monitoring and driving assistant is introduced. The proposed system in general consists of two main steps; the first one is the deep learning part in which a YOLO V8 model is trained using the DBB100k dataset for traffic monitoring, while the second one is the decision-support algorithm by which the prediction output of the YOLO V8 model is analyzed and the traffic status besides the driving decision are made.

First, the dataset is obtained and the training set is used to train the YOLO V8 model and the model is evaluated using the test set.

The performance metrics, including precision, recall, and mAP are utilized to evaluate the model.

The decision-support algorithm is the second part of the proposed framework. First, the test samples are fed into the trained and evaluated YOLO V8 model and the prediction output of the model is then fed into the decision-support algorithm.

The decision-support algorithm analyzes the prediction which will contain the detected objects, the recognized classes and the confidence values of the detected objects. Depending on these information, the algorithm will:

1- Count the number of instances of each detected object.

2- Compute the area or percentage area within the entire image.

3- Define the location of the objects (left or right side of the road).
4- Define the traffic status (light, heavy or moderate).
5- Define the number of nearby, faraway (cars, trucks, trains, etc.)
6- Define the less-traffic side of the road.

Based on previous mentioned decisions, the driver can safely drive the car.

The experiments are applied to the test set and then all prediction results are forwarded to the decision-support algorithm which make the final driving decision and deliver it to the driver.

The evaluated YOLO V8 model achieves the following results:

A precision value of 0.713

A recall value of 0.45

Map value of 0.50

The best detected class is "car" with:

Precision value of 0.794

Recall value of 0.695

mAP value of 0.77

5.2 Future perspectives

The current research tries to solve the problem of previous traffic monitoring DL based frameworks.

The study takes into consideration not only detecting the traffic main parts (car, truck, traffic light, traffic light, etc.), but also make the driving decision that can help driver to safely drive his car.

Although the current research solved many problems and produced a new decision-support algorithm, but it still needs enhancements, including:

1- Try other datasets to ensure the generalization of the study.
2- Develop new DL models or fuse other models with YOLO V8 model to improve the detection accuracy.
3- Expand the decision-support algorithm to take into consideration monitoring the driver behavior and make audible warning in case of drowsiness.

4- Expand the decision-support algorithm to take into consideration recognizing the traffic sign (stop, reduce speed, or change direction).

REFERENCES

[1] I. Contreras and J. Vehi, "Artificial intelligence for diabetes management and decision support: literature review," *Journal of medical Internet research,* vol. 20, no. 5, p. e10775, 2018.

[2] S. Balasubramani, J. Aravindhar, P. Renjith, and K. Ramesh, "DDSS: Driver decision support system based on the driver behaviour prediction to avoid accidents in intelligent transport system," *International Journal of Cognitive Computing in Engineering,* vol. 5, pp. 1-13, 2024.

[3] H. Dui, S. Zhang, M. Liu, X. Dong, and G. Bai, "IoT-Enabled Real-Time Traffic Monitoring and Control Management for Intelligent Transportation Systems," *IEEE Internet of Things Journal,* 2024.

[4] S. Gershov, A. Raz, E. Karpas, and S. Laufer, "Towards an autonomous clinical decision support system," *Engineering Applications of Artificial Intelligence,* vol. 127, p. 107215, 2024.

[5] G. T. Berge, O.-C. Granmo, T. O. Tveit, B. E. Munkvold, A. Ruthjersen, and J. Sharma, "Machine learning-driven clinical decision support system for concept-based searching: a field trial in a Norwegian hospital," *BMC Medical Informatics and Decision Making,* vol. 23, no. 1, p. 5, 2023.

[6] S. Ainslie, D. Thompson, S. Maynard, and A. Ahmad, "Cyber-threat intelligence for security decision-making: a review and research agenda for practice," *Computers & Security,* p. 103352, 2023.

[7] H. Meerveld, R. Lindelauf, E. Postma, and M. Postma, "The irresponsibility of not using AI in the military," *Ethics and Information Technology,* vol. 25, no. 1, p. 14, 2023.

[8] B. K. Russell *et al.*, "The value of a spaceflight clinical decision support system for earth-independent medical operations," *npj Microgravity,* vol. 9, no. 1, p. 46, 2023.

[9] B. Sadeghi and D. R. Cohen, "Decision-making within geochemical exploration data based on spatial uncertainty–A new insight and a futuristic review," *Ore Geology Reviews,* vol. 161, p. 105660, 2023.

[10] W. N. Caballero, D. Rios Insua, and D. Banks, "Decision support issues in automated driving systems," *International Transactions in Operational Research,* vol. 30, no. 3, pp. 1216-1244, 2023.

[11] Z. Fang, P. Yue, M. Zhang, J. Xie, D. Wu, and L. Jiang, "A service-oriented collaborative approach to disaster decision support by integrating geospatial resources and task chain," *International Journal of Applied Earth Observation and Geoinformation,* vol. 117, p. 103217, 2023.

[12] E. Dilek and M. Dener, "Computer vision applications in intelligent transportation systems: a survey," *Sensors,* vol. 23, no. 6, p. 2938, 2023.

[13] M. Hagl and D. R. Kouabenan, "Safe on the road–does advanced driver-assistance systems use affect road risk perception?," *Transportation research part F: traffic psychology and behaviour,* vol. 73, pp. 488-498, 2020.

[14] M. P. Capitaine and H. M. G. Cárdenas, "Artificial intelligence and advanced driver assistance systems absorption (ADAS) in Mexico," *Ciencia Nicolaita,* no. 88, 2023.

[15] R. Sethuraman, S. Sellappan, J. Shunmugiah, N. Subbiah, V. Govindarajan, and S. Neelagandan, "An optimized AdaBoost Multi-class support vector machine for driver behavior monitoring in the advanced driver assistance systems," *Expert Systems with Applications,* vol. 212, p. 118618, 2023.

[16] H. Nadeem *et al.*, "Road feature detection for advance driver assistance system using deep learning," *Sensors*, vol. 23, no. 9, p. 4466, 2023.

[17] V. Malligere Shivanna and J.-I. Guo, "Object Detection, Recognition, and Tracking Algorithms for ADASs—A Study on Recent Trends," *Sensors*, vol. 24, no. 1, p. 249, 2023.

[18] M. Flores-Calero *et al.*, "Traffic Sign Detection and Recognition Using YOLO Object Detection Algorithm: A Systematic Review," *Mathematics*, vol. 12, no. 2, p. 297, 2024.

[19] S.-K. Tai, C. Dewi, R.-C. Chen, Y.-T. Liu, X. Jiang, and H. Yu, "Deep learning for traffic sign recognition based on spatial pyramid pooling with scale analysis," *Applied Sciences*, vol. 10, no. 19, p. 6997, 2020.

[20] L. Wang, K. Zhou, A. Chu, G. Wang, and L. Wang, "An improved light-weight traffic sign recognition algorithm based on YOLOv4-tiny," *IEEE Access*, vol. 9, pp. 124963-124971, 2021.

[21] S. Ahmed, U. Kamal, and M. K. Hasan, "DFR-TSD: A deep learning based framework for robust traffic sign detection under challenging weather conditions," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 6, pp. 5150-5162, 2021.

[22] W. Min, R. Liu, D. He, Q. Han, Q. Wei, and Q. Wang, "Traffic sign recognition based on semantic scene understanding and structural traffic sign location," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 9, pp. 15794-15807, 2022.

[23] X. Wang *et al.*, "Real-time and efficient multi-scale traffic sign detection method for driverless cars," *Sensors*, vol. 22, no. 18, p. 6930, 2022.

[24] Y. Fan *et al.*, "Application of improved YOLOv5 in aerial photographing infrared vehicle detection," *Electronics*, vol. 11, no. 15, p. 2344, 2022.

[25] L. Kang, Z. Lu, L. Meng, and Z. Gao, "YOLO-FA: Type-1 fuzzy attention based YOLO detector for vehicle detection," *Expert Systems with Applications*, vol. 237, p. 121209, 2024.

[26] S. Mishra and D. K. Yadav, "Vehicle Detection in High Density Traffic Surveillance Data using YOLO. v5," *Recent Advances in Electrical & Electronic Engineering (Formerly Recent Patents on Electrical & Electronic Engineering)*, vol. 17, no. 2, pp. 216-227, 2024.

[27] M. Safaldin, N. Zaghden, and M. Mejdoub, "An Improved YOLOv8 to Detect Moving Objects," *IEEE Access*, 2024.

[28] L. Zhu, Y. Xiao, and X. Li, "Hybrid driver monitoring system based on Internet of Things and machine learning," in *2021 IEEE International Conference on Consumer Electronics and Computer Engineering (ICCECE)*, 2021: IEEE, pp. 635-638.

[29] N. Darapaneni *et al.*, "Distracted driver monitoring system using AI," in *2022 Interdisciplinary Research in Technology and Management (IRTM)*, 2022: IEEE, pp. 1-8.

[30] I. Jegham, I. Alouani, A. B. Khalifa, and M. A. Mahjoub, "Deep learning-based hard spatial attention for driver in-vehicle action monitoring," *Expert Systems with Applications*, vol. 219, p. 119629, 2023.

[31] P. Kunekar, Y. Narule, R. Mahajan, S. Mandlapure, E. Mehendale, and Y. Meshram, "Traffic Management System Using YOLO Algorithm," *Engineering Proceedings*, vol. 59, no. 1, p. 210, 2024.

[32] X. Song, Y. He, S. Dong, and Y. Gong, "Non-exemplar Domain Incremental Object Detection via Learning Domain Bias," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 2024, vol. 38, no. 13, pp. 15056-15065.

[33] T. Zhang, K. Kasichainula, Y. Zhuo, B. Li, J.-S. Seo, and Y. Cao, "Patch-based Selection and Refinement for Early Object Detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2024, pp. 729-738.

[34] X. Yu and X. Lu, "Domain Adaptation of Anchor-Free object detection for urban traffic," *Neurocomputing*, vol. 582, p. 127477, 2024.

[35] P. Cong, H. Feng, S. Li, T. Li, Y. Xu, and X. Zhang, "A visual detection algorithm for autonomous driving road environment perception," *Engineering Applications of Artificial Intelligence*, vol. 133, p. 108034, 2024.

[36] J. Lu, T. Huang, Q. Zhang, X. Chen, and J. Zhou, "A lightweight vehicle detection network fusing feature pyramid and channel attention," *Internet of Things*, vol. 26, p. 101166, 2024/07/01/ 2024, doi: https://doi.org/10.1016/j.iot.2024.101166.

[37] F. Yu *et al.*, "Bdd100k: A diverse driving dataset for heterogeneous multitask learning," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 2636-2645.

[38] L. Nanni, M. Paci, S. Brahnam, and A. Lumini, "Comparison of different image data augmentation approaches," *Journal of imaging*, vol. 7, no. 12, p. 254, 2021.

[39] K. Szyc, "An impact of data augmentation techniques on the robustness of CNNs," in *International Conference on Dependability and Complex Systems*, 2022: Springer, pp. 331-339.

[40] A. M. Pour, H. Seyedarabi, S. H. A. Jahromi, and A. Javadzadeh, "Automatic detection and monitoring of diabetic retinopathy using efficient convolutional neural networks and contrast limited adaptive histogram equalization," *IEEE Access*, vol. 8, pp. 136668-136673, 2020.

[41] S. A. Khan, S. Hussain, and S. Yang, "Contrast enhancement of low-contrast medical images using modified contrast limited adaptive histogram equalization," *Journal of Medical Imaging and Health Informatics*, vol. 10, no. 8, pp. 1795-1803, 2020.

[42] O. Russakovsky *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211-252, 2015.

[43] J. Terven and D. Cordova-Esparza, "A comprehensive review of YOLO: From YOLOv1 to YOLOv8 and beyond," *arXiv preprint arXiv:2304.00501*, 2023.

[44] B. Selcuk and T. Serif, "A Comparison of YOLOv5 and YOLOv8 in the Context of Mobile UI Detection," in *International Conference on Mobile Web and Intelligent Information Systems*, 2023: Springer, pp. 161-174.

[45] M. Hussain, "YOLO-v1 to YOLO-v8, the Rise of YOLO and Its Complementary Nature toward Digital Manufacturing and Industrial Defect Detection," *Machines*, vol. 11, no. 7, p. 677, 2023.

[46] S. Akhtar, M. Hanif, and H. Malih, "Automatic Urine Sediment Detection and Classification Based on YoloV8," in *International Conference on Computational Science and Its Applications*, 2023: Springer, pp. 269-279.

[47] W. M. Elmessery *et al.*, "YOLO-Based Model for Automatic Detection of Broiler Pathological Phenomena through Visual and Thermal Images in Intensive Poultry Houses," *Agriculture,* vol. 13, no. 8, p. 1527, 2023.

[48] H. Orchi, M. Sadik, M. Khaldoun, and E. Sabir, "Real-Time Detection of Crop Leaf Diseases Using Enhanced YOLOv8 algorithm," in *2023 International Wireless Communications and Mobile Computing (IWCMC),* 2023: IEEE, pp. 1690-1696.

APPENDIX A

Code of training and evaluating the YOLO V8 model

```
import torch
import os
import numpy as np
import cv2
import pandas as pd
import matplotlib.pyplot as plt
from glob import glob
import yaml
!pip install opendatasets
import opendatasets as ods
ods.download("https://www.kaggle.com/datasets/aayusmaanjain/bdd100k-for-self-driving-cars")
IMG_HEIGHT = 720
IMG_WIDTH = 1280
labels = [
    "bike",
    "bus",
    "car",
    "motor",
    "person",
    "rider",
    "traffic light",
    "traffic sign",
    "train",
    "truck"
]
TRAIN_PATH = '/content/bdd100k-for-self-driving-cars/Data/train'
VAL_PATH = '/content/bdd100k-for-self-driving-cars/Data/val'
MODEL_PATH = '/content/bdd100k-for-self-driving-cars/Data/best.pt'
```

```
INFERENCE_PATH = '/content/bdd100k-for-self-driving-
cars/Data/inference_vid.mp4'
STEERING_WHEEL_PATH = '/content/bdd100k-for-self-driving-
cars/Data/steering_wheel_image.jpg'
train_images = glob(f'{TRAIN_PATH}/*.jpg')
val_images = glob(f'{VAL_PATH}/*.jpg')
n_samples = 5#view example of the dataset just for view
train_sample = np.random.choice(train_images, size=n_samples)
val_sample = np.random.choice(val_images, size=n_samples)
for i in range(n_samples):
    _, ax = plt.subplots(figsize=(16,6))
    train_path = train_sample[i]
    img_id = train_path.split(os.path.sep)[-1].split('.')[0]
    label_file = os.path.join(TRAIN_PATH, f'{img_id}.txt')
    with open(label_file, 'r') as f:
        lines = f.readlines()
    img = cv2.imread(train_path)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    for label in lines:
        splits = label.split()
        category = labels[int(splits[0])]
        x_center = float(splits[1]) * IMG_WIDTH
        y_center = float(splits[2]) * IMG_HEIGHT
        width = float(splits[3]) * IMG_WIDTH
        height = float(splits[4]) * IMG_HEIGHT
        pt1_x = x_center - width/2
        pt1_y = y_center - height/2
        pt2_x = x_center + width/2
        pt2_y = y_center + height/2
        pt1 = (int(pt1_x), int(pt1_y))
        pt2 = (int(pt2_x), int(pt2_y))
        img = cv2.rectangle(img, pt1=pt1, pt2=pt2,
color=(0,255,0), thickness=2)
        img = cv2.putText(img, category, org=pt1,
                          color=(0,255,0), fontFace =
cv2.FONT_HERSHEY_COMPLEX,
                          fontScale=1, thickness=2)
    ax.imshow(img)
    ax.axis('off')
    plt.show()
# Creating data.yaml file
data = {
    'train':os.path.abspath(TRAIN_PATH),
    'val':os.path.abspath(VAL_PATH),
    'names':labels,
    'nc':len(labels)
}
```

```
with open('data.yaml', 'w+') as f:
    yaml.safe_dump(data, f)
!pip install GPUtil
import gc
import torch
from numba import cuda
from GPUtil import showUtilization as gpu_usage
%pip install ultralytics
import ultralytics # YOLO V8 Library
ultralytics.checks()
!yolo train model=yolov8s.pt data="/content/data.yaml" epochs=20
verbose=True batch=32 imgsz=736 augment=True
os.listdir("/content/runs/detect/train")
training_results =
pd.read_csv("/content/runs/detect/train/results.csv")
list_column = [col.strip() for col in training_results.columns]
training_results.columns = list_column
training_results
# Training and Validation Loss
# Data Visualization dependencies
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import plotly
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots
from IPython.display import Image, display
# Create Subplot
fig = make_subplots(
        rows=1, cols=3,
        subplot_titles=["Localization Loss", "Classification
Loss", "Distributional Focal Loss"],
)
# Configuration Plot
class PlotCFG:
    marker_size = 6
    line_size = 2
    train_color = "#1e3772"
    valid_color = "#2788f9"

loss_list = ["box_loss", "cls_loss", "dfl_loss"]
for i, loss in enumerate(loss_list):
    fig.add_trace(
        go.Scatter(
            x=np.arange(1, training_results.shape[0]+1),
y=training_results["train/{}".format(loss)],
```

```
            mode="markers+lines",
            marker=dict(
                color=PlotCFG.train_color,
size=PlotCFG.marker_size, line=dict(color="White", width=0.5)
            ),
            line=dict(color=PlotCFG.train_color,
width=PlotCFG.line_size),
            name="Training"
        ), row=1, col=i+1
    )
    fig.add_trace(
        go.Scatter(
            x=np.arange(1, training_results.shape[0]+1),
y=training_results["val/{}".format(loss)],
            mode="markers+lines",
            marker=dict(
                color=PlotCFG.valid_color,
size=PlotCFG.marker_size, line=dict(color="White", width=0.55)
            ),
            line=dict(color=PlotCFG.valid_color,
width=PlotCFG.line_size),
            name="Validation"
        ), row=1, col=i+1
    )
ticklabels = ["Box Loss", "Cls Loss", "DFL Loss"]
for i, ticklabel in enumerate(ticklabels):
    fig.update_xaxes(title="Epochs", linecolor="Black",
ticks="outside", row=1, col=i+1)
    fig.update_yaxes(title=ticklabel, linecolor="Black",
ticks="outside", row=1, col=i+1)

# Update Layout
fig.update_layout(
    title="Training and Validation Loss", title_x=0.5,
font_family="Trebuchet MS",
    width=950, height=350,
    showlegend=False,
    plot_bgcolor="White",
    paper_bgcolor="White"
)
# Show results
fig.show(iframe_connected=True)
# Validation Metrics

# Create Subplot
subplot_titles = ["Precision", "Recall", "mAP50", "mAP50-95"]
fig = make_subplots(
```

```
        rows=2, cols=2,
        subplot_titles=subplot_titles,
)
# Configuration Plot
class PlotCFG:
    marker_size = 7
    line_size = 2
    train_color = "#1e3772"
    valid_color = "#2788f9"
metrics = ["precision", "recall", "mAP50", "mAP50-95"]
for i, metric in enumerate(metrics):
    # Plot
    fig.add_trace(
        go.Scatter(
            x=np.arange(1, training_results.shape[0]+1),
y=training_results["metrics/{}(B)".format(metric)],
            mode="markers+lines",
            marker=dict(
                color=PlotCFG.valid_color,
size=PlotCFG.marker_size, line=dict(color="White", width=0.5)
            ),
            line=dict(color=PlotCFG.valid_color,
width=PlotCFG.line_size),
            name="Validation"
        ), row=(i//2)+1, col=(i%2)+1
    )
    # Update Axes
    fig.update_xaxes(title="Epochs", linecolor="Black",
ticks="outside", row=(i//2)+1, col=(i%2)+1)
    fig.update_yaxes(title=subplot_titles[i], linecolor="Black",
ticks="outside", row=(i//2)+1, col=(i%2)+1)

# Update Layout
fig.update_layout(
    title="Validation Metrics", title_x=0.5,
font_family="Trebuchet MS",
    width=950, height=600,
    showlegend=False,
    plot_bgcolor="White",
    paper_bgcolor="White"
)
# Show results
fig.show(iframe_connected=True)
!yolo val model="/content/runs/detect/train/weights/best.pt"
data="/content/data.yaml" split="val"
os.listdir("/content/runs/detect/val")
#Testing Metrics
```

```
# Confusion Matrix (CM)
fig = plt.figure(figsize=(10, 9))
cm_img =
mpimg.imread("/content/runs/detect/val/confusion_matrix.png")
plt.imshow(cm_img)
plt.axis("off")
fig.show()
# Testing Curve
fig, axs = plt.subplots(2, 2, figsize=(13, 7.5))
curve_list = ["P_curve", "R_curve", "F1_curve", "PR_curve"]
for i, curve in enumerate(curve_list):
    curve_path = "/content/runs/detect/val/{}.png".format(curve)
    curve_img = mpimg.imread(curve_path)
    axs[i//2, i%2].imshow(curve_img)
    axs[i//2, i%2].axis('off')
plt.suptitle("Testing Curve", x=0.55, y=0.93)
plt.show()
```