

SMART TRAFFIC MONITORING SYSTEM USING YOLO

Submitted in partial fulfilment of the requirements for the award of
Bachelor of Technology degree in Information Technology

By

CHERUKU NITHESH REDDY (40120021)

NALLA UDAY KIRAN (40120061)



DEPARTMENT OF INFORMATION TECHNOLOGY

SCHOOL OF COMPUTING

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Category-1 University by UGC

Accredited with Grade "A++" by NAAC | 12B Status by UGC |

Approved by AICTE

JEPPIAAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI – 600119

APRIL 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

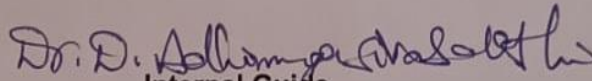
Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

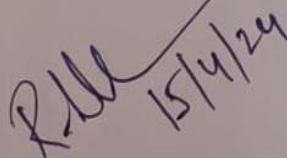
DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Cheruku Nithesh Reddy (40120021)** and **Nalla Uday Kiran (40120061)**, who carried out Project Phase-1 entitled "**SMART TRAFFIC MONITORING SYSTEM USING YOLO**" under my supervision from January 2024 to April 2024.


Internal Guide

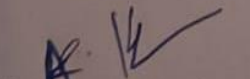
Dr. D. ADHIMUGA SIVASAKTHI

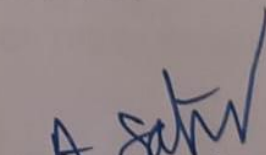

15/4/24

Head of the Department

Dr. R. SUBHASHINI, M.E., Ph. D.,

Submitted for Viva voce Examination held on 17-04-2024


Internal Examiner


External Examiner

DECLARATION

I, **CHERUKU NITHESH REDDY**, hereby declare that the Project Report entitled "**SMART TRAFFIC MONITORING SYSTEM USING YOLO**" done by me under the guidance of **Dr. D. Adhimuga Sivasakthi** is submitted in partial fulfilment the requirements for the award of Bachelor of Technology degree in **Information Technology**.

DATE: 17-04-2024

PLACE: Chennai

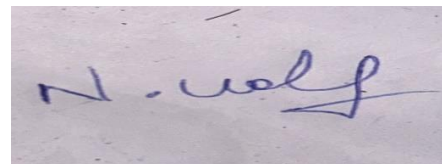
C. Nithish Reddy
SIGNATURE OF THE CANDIDATE

DECLARATION

I, **NALLA UDAY KIRAN**, hereby declare that the Project Report entitled “**SMART TRAFFIC MONITORING SYSTEM USING YOLO**” done by me under the guidance of **Dr. D. Adhimuga Sivasakthi** is submitted in partial fulfilment the requirements for the award of Bachelor of Technology degree in **Information Technology**.

DATE: 17/04/2024

PLACE: Chennai

A photograph of a handwritten signature in blue ink on a light-colored surface. The signature appears to be 'N. Uday Kiran' written in a cursive style.

SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to the **Board of Management of SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala, M.E., Ph.D., Dean**, School of Computing, and **Dr. R. Subhashini, M.E., Ph.D.**, Head of the Department of Information Technology, for providing me with the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my project guide, **Dr. D. Adhimuga Sivasakthi** for the valuable guidance, suggestions, and constant encouragement that paved the way for the successful completion of the project work.

I wish to express my thanks to all teaching and non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

ABSTRACT

The world's population is growing; as a result, the number of vehicles on the road is increasing tremendously daily, which leads to heavy traffic. It is a hard task to manage and monitor the traffic moments. Traffic monitoring is essential for preventing accidents on roads.

There is a need for a model which can monitor, detect, and classify vehicles. Counting the number of vehicles also plays a crucial role in the traffic moments, as it helps the authorities to avoid traffic jams and accidents due to heavy traffic. Detection and classification of vehicles are the two most challenging tasks of a video-based intelligent transportation system.

Traditional detection and classification methods are computationally highly expensive and become unsuccessful in many cases, such as collisions among vehicles and when differences between pixel intensities of vehicles and backgrounds are small. There are several applications of vehicle classification and detection, including traffic monitoring, surveillance, and parking lot management. In traffic monitoring, vehicle classification and detection can be used to monitor the flow of traffic and identify congestion points. In parking lot management, it can be used to count the number of vehicles entering and leaving the parking lot, and to identify parking violations.

This project presents a way to detect vehicles, classify them by using image processing techniques like OpenCV, and YOLO. This process of detection of vehicles is divided into several parts: 1) Data Collection, 2) Data Preprocessing, 3) Detecting of the vehicles, 4) Classifying the type of vehicles, 5) Counting of the vehicles, 6) Display the output. The classification of vehicles is done by the size of the object detected by the algorithm.

TABLE OF CONTENTS

Chapter No	Title	Page No.
	ABSTRACT	vi
	LIST OF FIGURES	ix
	LIST OF TABLES	ix
	LIST OF ABBREVIATIONS	x
1	INTRODUCTION	1
	1.1 Area of Research	2
2	LITERATURE SURVEY	3
	2.1 Inferences from Literature Survey	6
	2.2 Open problems in Existing System	7
3	REQUIREMENTS ANALYSIS	9
	3.1 Risk Analysis of the Project	9
	3.2 Software Requirements Specification Document	10
	3.2.1 Introduction	10
	3.2.1.1 Purpose of this document	10
	3.2.1.2 Scope of this document	10
	3.2.1.3 Overview	10
	3.2.2 General description	10
	3.2.3 Functional Requirements	11
	3.2.4 Performance Requirements	11
	3.2.5 Design Constraints	11
	3.2.6 Non-Functional Attributes	11
	3.2.7 Appendices	12
	3.3 Software and Hardware Requirements	12
4	DESCRIPTION OF THE PROPOSED SYSTEM	13

4.1	Selected methodology	14
4.2	Description of Software for Implementation and Testing plan of the Proposed Model	20
4.3	Project Management Plan	22
5	IMPLEMENTATION DETAILS	23
5.1	Algorithms	23
5.2	Testing of the model	32
6	RESULTS AND DISCUSSION	39
7	CONCLUSION	41
7.1	Conclusion	41
7.2	Future Work	42
7.3	Research Issues	43
7.4	Implementation issues	44
	REFERENCES	45
	APPENDIX	47
	A. SOURCE CODE	47
	B. SCREENSHOTS	62
	C. RESEARCH PAPER	70

LIST OF FIGURES

FIGURE NO.	FIGURE NAME	PAGE NO.
4.1	Flow Diagram	14
4.2	Training the Model with 80 Epochs.	19
5.1	YOLOV8-Architecture	24
5.2	P-R curve of training	34
5.3	Confusion Matrix of different classes.	36
5.4	Precision-Confidence Curve	37
5.5	F1-Confidence Curve	38
6.1	Performance of the model.	39
6.2	Final Output	40

LIST OF TABLES

TABLE NO.	TABLE NAME	PAGE NO.
3.2.1	APPENDICES	12

LIST OF ABBREVIATIONS

SHORT FORMS	FULL FORMS
OpenCV	OPEN-SOURCE COMPUTER VISION
YOLO	YOU ONLY LOOK ONCE
R-CNN	REGION-BASED CONVOLUTIONAL NEURAL NETWORK
LSTM	LONG SHORT-TERM MEMORY
CNN	CONVOLUTIONAL NEURAL NETWORK
SVM	SUPPORT VECTOR MACHINE
KNN	K-NEAREST NEIGHBOURS
XML	EXTENSIBLE MARKUP LANGUAGE
IOU	INTERSECTION OVER UNION

CHAPTER 1

INTRODUCTION

The number of vehicles has increased tremendously over the past few years. Vehicle detection and counting are becoming increasingly important in the field of intelligent highway management systems. A vehicle detection system is an application of an object detection system in which the primary focus is on tracking the vehicle using its coordinates in images or videos. The efficiency and accuracy of vehicle detection plays a vital role in traffic management systems. Video surveillance systems have long been in use to monitor security-sensitive areas. With the proper installation of traffic surveillance cameras, vast data of traffic video footage has been obtained for analysis. Detection and tracking of vehicles using the traffic surveillance system give a more promising way to manage and control road traffic. Due to the different sizes of vehicles, their detection remains a challenge that directly affects the accuracy of vehicle count. Vehicle detection is still an important challenge in computer vision. The research of intelligent transportation systems is very important to ensure the smoothness and safety of roads. Nowadays, Object detection is being deployed across various industries. The use cases range from video surveillance to sports broadcasting to robot navigation. The traditional methods of vehicle detection and classification did not give as good results as expected. Due to advancement in technology, various machine learning and deep learning algorithms like computer vision are introduced, which efficiently detects objects and improves the accuracy of the detection of the models. Several sub-tasks are needed to perform this project, such as object detection, counting the number of objects, finding the relative size of the objects, or finding the relative distance between the objects.

It is challenging for some individuals to record all the vehicles with them because the cars pass by in real-time. It's not like when one is watching the video, and they can pause it and have a note of it, so to remove this limitation, this application can be very well-versed to attain time-saving quality and be automated. Vehicle detection systems can keep track of individual vehicles and overcome the challenges of the inability to locate them in images collected from a moving platform.

First, the moving vehicles are detected in the frames of the video sequence by combining some deep learning and object detection techniques. Vehicle classification is a very important process in traffic management systems. Outcomes of the vehicle classification can be used for different aims, such as reporting and statistical analysis. Detecting the number of vehicles plays a vital role in traffic flow analysis for identifying traffic areas, and thereby, the traffic department can take necessary actions to decrease the accidents and reduce the traffic. Traffic flow analysis can be used for detecting traffic rule violators and thereby taking necessary actions leads to reducing the total number of accidents caused, and thereby the life of people can be saved.

1.1 Area of Research

Machine learning (ML) and Deep Learning (DL) investigate computer science algorithmic calculations that can work naturally through experience and data. It is part of artificial intelligence. Machine learning algorithms fabricate a model dependent on example data, known as “Training data,” to make predictions or decisions without programming for every application. Machine learning algorithms are utilized in various applications, such as medication, email sorting, speech recognition, and computer vision. Developing standard algorithms to perform the required tasks is troublesome or unworkable. A subset of machine learning is rigidly identified with computational statistics, which centers around making computer predictions, yet not all machine learning is statistical learning. The investigation of mathematical optimization conveys strategies, hypotheses, and application areas to the field of machine learning. Data mining is a related field of study, focusing on exploring data investigation through unsupervised learning. Some executions of machine learning use data and neural networks that impersonate the working of an actual brain. Machine learning is likewise referred to as proactive analysis in its application across business issues.

CHAPTER 2

LITERATURE SURVEY

This report focuses on constructing a traffic monitoring system for estimating traffic parameters, such as vehicle counting and classification. Vision-based vehicle detection and classification play an important role in real-time traffic management systems. Real-time vehicle detection in video streams relies heavily on image processing techniques, such as motion segmentation, edge detection, digital filtering, etc. The vehicle detection techniques developed by researchers have been given in this literature survey.

Ali Tourani and Asadollah Shahbahrami, [3] presented a Vehicle Counting Method Based on Digital Image Processing Algorithms to estimate important traffic parameters from video sequences using video. They introduced a new “linearity” feature in vehicle representation. This method is different from traditional methods, and it detects vehicles with good accuracy. The programming language they used is C++, which doesn’t have all the necessary libraries compared to Python, and there is mention of speed estimation and nighttime evaluation, and there is no masking method used for detecting shadows in the traffic video. These Problems can be solved by using backgroundSubtractorMOG2(), which efficiently detects shadows.

Amit Kumar, Mansour H. Assaf, Sunil R. Das, Satyendra N. Biswas, Emil M. Petriu, and Voicu Groza [4] presented an Image Processing Based System for the Classification of Vehicles for Parking Purposes where they only detected the type of vehicle that is being parked, and there is no mention of vehicle counting, speed estimation, and they used MATLAB for image processing, and the technique they used is different from other traditional approaches, and the accuracy is comparatively low, and the detection of a class of vehicles of this system is limited to images as the real-world data is of the form is of video form, so there are many more advanced techniques for object detection.

S. Kul, S. Eken, and A. Sayer [14] presented Distributed and collaborative real-time

vehicle detection and classification over the video streams, which is based on detecting vehicles and classifying them, and they have used BackgroundsubtractorMOG2() for the elimination of shadows till some extent. The Algorithms they used are ANN, SVM, and AdaBoost for the Classification of vehicles they are pretty much good algorithms that can better classify and detect them, like YOLO, R-CNN, CNN, LSTM, and many more, and there is no mention of detecting the speed of the vehicle and counting them.

Pandu Ranga, H.T; Ravi Kiran, M; Raja Shekar, S; Naveen Kumar [12] proposed Vehicle Detection and Classification based on Morphological Technique which can detect, classify, and count the number of vehicles using image thresholding techniques without any masking method and this methodology was good of detecting the specific size of vehicles as they are using vertical edge detection method for identifying the vehicles. And there is no mention of speed detection, and this system was limited to images and cannot be further extended by this methodology.

Jyotsna Tripathi, Kavita Chaudhary, Akanksha Joshi, and Prof. J B Jawale [9] proposed an Automatic Vehicle Counting and Classification to detect and classify the vehicles the algorithms used Adaptive Background subtraction for background elimination and Color Based Classification for detecting and classification of vehicles. This model can classify vehicles like small, medium, and heavy, not identify the type of vehicles. There is no mention of speed estimation and nighttime data, and accuracy is not that good as the methodology was a little bit outdated and much more efficient algorithms are available.

A.H.S. Lai, G.S.K. Fung, N.H.C. Yung [2] proposed Classification of Vehicles from Video Using Multiple Time-Spatial Images,” which is used to detect and classify them accordingly and used background subtractor for detecting the shadows. A new method, MVDL-based detection, was used for detection and classification accordingly, this methodology was used to distinguish the correspondence of vehicles, and there is no mention of speed detection and nighttime data, and the KNN algorithm is not an efficient algorithm to classify the vehicles.

Zhao, R.; Wang, X [15] Proposed Counting Vehicles from Semantic Regions, Intelligent Transportation which is used for counting the vehicles. Their methodology is based on tracking and clustering feature points, and a traffic scene is segmented into local regions and then processed. By counting vehicles on each path separately, it breaks the challenging problem into simpler problems. In extremely crowded scenes, vehicles are not perfectly located and detected. In extremely crowded scenes, vehicles on adjacent lanes may be very close in space and move side by side at the same speed. It poses difficulty in trajectory clustering. This problem can be alleviated to some extent by first excluding trajectories outside a lane before clustering.

R. Gonzalez, R. E. Woods [13] proposed Vehicle Detection and Classification using Image processing. The vehicle is successfully detected from the video frames. The counter works with high accuracy in daylight situations however accuracy decreases if lighting conditions are not good. The centroid for each car that is traced is detected, the classification of vehicles is restricted, and the classification is restricted to a few classes. BackgroundSubtractorMOG2() is used to extract the frames and detect the shadows.

Prem Kumar Bhaskar Suet-Peng Yon [11] proposed the Image Processing Based Vehicle Detection and Tracking Method. They developed a unique algorithm for vehicle data recognition and tracking using the Gaussian mixture model and blob detection methods. They differentiated the foreground from the background in frames by learning the background. The foreground detector detects the object, and a binary computation defines rectangular regions around every detected object. To detect the moving object correctly and to remove the noise, some morphological operations have been applied to improve the performance of the model, and thereby, detection has become an easy part and perfect.

2.1 INFERENCES FROM LITERATURE SURVEY

[3] The methodology that is used is not that efficient, and the accuracy is that good, and python and its integrated libraries make object detection and image processing faster and more efficient and faster while compared to C++. There is no mention of nighttime vehicles, and speed estimation here; the number of frames is limited to 29 frames per second, and the resolution is limited to 960*540.

[4] The methodology they used was not as efficient as new methodologies came into existence, and it is limited to images and cannot be further extended by using the same methodology MATLAB is used for image processing limited to detecting the specific vehicles, and classification is also restricted to certain futures only.

[14] The process that they developed is not applicable for real-time detections because of scalability and performance problems. Since Video data is streaming data and is very large in size, the processing of data is quite difficult, so they have not done these for real-time object detection functionalities and no background noise is removed for the perfection of accuracy.

[12] The algorithm they proposed is based on different techniques, including image differencing, thresholding, edge detection, and binary morphological process. To reduce the noise in an output image multiple thresholding, they performed with different thresholding values. No masking methodology was used for detecting the shadows.

[9] They used Adaptive Background Subtraction Color Based Classification for detecting and classifying the vehicles. This Single system provides multiple domain outputs. Tracking criminal vehicles with the help of color and type of vehicle. More work is needed to reduce occlusions. Only classify the vehicles like small, medium, and heavy, not identify the type of vehicles. There are many more advanced algorithms for identifying and classification of objects which give more performance compared to this methodology.

[2] A new method, MVDL-based detection, was used for the detection and classification of them accordingly. TSI methodology was used to distinguish the

correspondence of vehicles, and there is no mention of speed detection and nighttime data, and the KNN algorithm is not an efficient algorithm to classify the vehicles.

[15] Their methodology breaks the problem into simpler problems, which count vehicles on each path separately. The model of each path and its source and sink add strong regularization on the motion and the sizes of vehicles and can thus significantly improve the accuracy of vehicle counting. This approach has some limitations and can be improved in several aspects. A semantic region could be detected if pedestrians frequently walk through a zebra crossing. Pedestrian paths and vehicle paths can be distinguished by simple human intervention or the distributions of object sizes and speed along the paths.

[13] The approach they used to detect the vehicles is pretty much good, and they also mentioned their research on nighttime data and their analysis of night and daytime analysis was mentioned and then can be further extended by adding some features to it like adding estimation of speed and adding some more features for selection methods. The number of classes of prediction can also be extended by using YOLO pretrained algorithm.

[11] The methodology they used is quite interesting and impressive, and they used noise elimination methods to improve the accuracy of the model feature extraction methodologies are also performed to detect necessary features and vehicle data recognition and tracking using the Gaussian mixture model and blob detection methods are used to identify the vehicles and classify them, and there is no mention of speed estimation.

2.2 OPEN PROBLEMS IN THE EXISTING SYSTEM

The main problems in the existing system are mainly the model's accuracy and not using advanced deep learning frameworks that efficiently detect and classify them as per their classes. The classification of a few models is limited to few classes only, but there can be a further extension of classes for classification of vehicles, and the majority of researchers have done detection on images as the real-world data is of

the video format, so it can't be used for solving real-world problems to monitor the traffic flow analysis the researchers have not much concentrated on removing the noise from the traffic video to improve the accuracy of detection of vehicles. The algorithms like KNN, SVM, and AdaBoost like algorithms are used for the detection of objects and classifying them, but there are pretty good algorithms like R-CNN, YOLO, and LSTM, and many more performance of can be used to detect the classes of vehicles and the accuracy during night time data is not that accurate, many researchers had not concentrated on the speed of vehicles which plays a vital role for identifying the traffic rule violators, and there are many more advancements in technology which improves the performance the model. This project aims to solve a few problems in the existing system by using various deep learning and machine learning methodologies which can be used for monitoring traffic.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 RISK ANALYSIS OF THE PROJECT

This study of detecting the type of vehicles, calculating the vehicle count, has raised many questions. Collecting the proper traffic video was an essential task for the project. After obtaining the video removing the noise from the video played a crucial role as it affects the model's accuracy. By using BackgroundSubtractorKNN () method, the noise in the video can be removed. Resolving all the above issues was a tough task. As soon as the issues were resolved, it was observed that the model was detecting the shadows of vehicles which had led to detect multiple images at a time, which would later disturb the count of the vehicles when calculated for removing the shadows BackgroundSubtractorMOG2() method is used. The noise and shadows of the vehicles are cleaned at a nice rate. Adjusting the frames for objects(vehicles) that were not being well detected by the BackgroundSubtractorMOG2() is resolved by using YOLO (You Only Look Once) algorithm. YOLO is one of the most efficient algorithms in the field of object detection as the name suggests, it takes the whole input once, and in one forward pass, it gives us the predictions with the help of a pre-trained dataset called coco. The YOLO algorithm can take every image from the video and classifies them according to their respective classes. Although YOLO is faster it takes lots of computational resources to train and develop the model. Adjusting the imaginary line for counting the vehicles was tough. The proper determining of the imaginary line is important as this line is used to count the number of vehicles which is an important task in traffic monitoring. Integrating the Bytetrack with Deepsort will be another challenging task. After integration of those two algorithms, it should further integrate to YOLO model they have been developed it takes lots of computational resources to integrate them.

3.2 SOFTWARE REQUIREMENTS SPECIFICATION DOCUMENT

3.2.1 INTRODUCTION

3.2.1.1 Purpose of this document

The purpose of this document is to identify the essential requirements of the software and define the requirements to make sure the development is moving in the right path.

3.2.1.2 Scope of this document

This project's main objective is to help the traffic department to understand the traffic flow analysis of a particular area. Vehicle detection, counting, and speed estimation help them to monitor and control the traffic.

3.2.1.3 Overview

Vehicle detection based on computer vision is an essential part of the smart traffic monitoring system based on computer vision is an essential part of the smart traffic monitoring system. Smart traffic and information systems require the collection of traffic data from respective sensors for the regulation of traffic. Surveillance cameras have been installed to monitor and control traffic in the last few years. Several studies have been carried out in video surveillance technologies using image processing techniques for traffic management. Video processing of traffic data obtained through surveillance cameras is an instance of applications for advanced cautioning or data extraction for real-time analysis of vehicles. To detect and classify the vehicles, YOLO V8 is one of the best choices because of its speed and accuracy as it is pre-trained on a custom dataset called the "coco dataset," which consists of pre-trained labels, giving us more performance, which increases the accuracy of the model.

3.2.2 General description

There is a need for a smart traffic management system for controlling and understanding traffic flow analysis. Vehicle detection using deep learning can be carried out using various deep learning algorithms. YOLO is one of them. YOLO is fast and accurate. The YOLO algorithm uses neural networks to provide real-time object detection. This Smart traffic management system helps the traffic police to identify the vehicle count and speed of each individual vehicle so that they can keep

track of the number of vehicles and speed of every vehicle and find the number of vehicles that cross the specified speed limit set by them.

3.2.3. Functional Requirements

We are developing a deep learning model that can detect the vehicles and count the number of vehicles and estimate the speed of the vehicles. Functional Requirements of the smart traffic monitoring System are:

1. Collecting the proper video consisting of vehicles.
2. The Collected video can be used to pass as input to the model.
3. The developed model needs to detect, count, and classify the vehicles accordingly.
4. YOLO is an efficient algorithm to detect and classify them with the help of pre-trained weights.

3.2.4. Performance Requirements

With the help of Graphical Processing Units (GPUs), the processing power increases, and performance also increases. With the help of the YOLO algorithm, a pre-trained model can be used to improve the performance of the model. YOLO is famous because of its speed and accuracy. So, using the YOLO algorithm improves the model's performance and gives good results.

3.2.5. Design Constraints

With the help of the software present, the system can detect only a few things like count and classifying the vehicles Smart Traffic monitoring system comes as a new system adding a few more additional features and improved accuracy, making an easy-to-use interface for people to use the software.

3.2.6. Non-Functional Attributes

The system must be designed so that background tasks can continue while the user performs foreground tasks. The response time of the model should be low. The system must give stability and the possibility to reuse the results in the future.

The system must be working in real-time, which allows the users to view it 24/7. The system shall be able to be maintainable. The system shall be able to be extended in the future. Deployment of the smart traffic management application should be done on the cloud with cloud computing capabilities with GPUs to process the real-time data for the traffic monitoring system.

3.2.7. Appendices

OpenCV	OPEN-SOURCE COMPUTER VISION
YOLO	YOU ONLY LOOK ONCE
R-CNN	REGION-BASED CONVOLUTIONAL NEURAL NETWORK
LSTM	LONG SHORT-TERM MEMORY
CNN	CONVOLUTIONAL NEURAL NETWORK
SVM	SUPPORT VECTOR MACHINE
KNN	K-NEAREST NEIGHBOURS

Table 3.2.1 APPENDICES

3.3 Software and Hardware Requirements

Software Requirements

Installing Anaconda Individual edition 64-bit (PY 3.8)

Use Jupiter notebook in Anaconda Navigator for running project Python notebook.

Python Notebook also works in Google Collab and Kaggle Notebook editor.

Git-hub repositories are used for cloning the repositories and saving the results.

Hardware Requirements

License: Free use and redistribution under the terms of the EULA for Anaconda Individual Edition.

Operating system: Windows 8 or newer, 64-bit macOS 10.13+, or Linux, including Ubuntu, RedHat, CentOS 7+, and others.

System architecture: Windows- 64-bit x86, 32-bit x86; MacOS- 64-bit x86; Linux- 64-bit x86, 64-bit aarch64 (AWS Graviton2 / arm64), 64-bit Power8/Power9, s390x (Linux on IBM Z & Linux ONE).

Minimum 5 GB disk space to download and install.

CHAPTER 4

DESCRIPTION OF THE PROPOSED SYSTEM

The proposed system for smart traffic monitoring using YOLO (You Only Look Once) would involve the implementation of a deep learning model for object detection and tracking in real-time video streams. YOLO is an object detection algorithm that uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation. It is known for its fast and accurate detection of objects in images and videos. YOLO is an effective tool for vehicle classification and detection in smart traffic monitoring systems. The YOLO algorithm is designed to be able to detect and classify objects in real-time, making it an ideal choice for traffic monitoring applications. To detect vehicles using YOLO, the algorithm would be trained on a dataset of annotated images and videos of vehicles in various lighting conditions, angles, and viewpoints. The YOLO model would learn to detect the unique features of vehicles, such as their size, shape, and color, and classify them as cars, trucks, motorcycles, or other types of vehicles.

To count the number of vehicles ByteTrack is used, ByteTrack is a real-time object detection algorithm that can be used for vehicle counting in smart traffic monitoring systems. It uses a deep learning architecture to detect and track vehicles in live video streams. The algorithm can be trained on a dataset of annotated traffic images and videos, allowing it to detect and count vehicles accurately. Once deployed, ByteTrack can analyze video streams from traffic cameras and provide real-time vehicle counting data, including traffic volume and traffic patterns. This information can be used to optimize traffic flow, improve safety, and reduce congestion. Overall, the use of ByteTrack in vehicle counting can provide valuable insights for traffic management and urban planning.

4.1 SELECTED METHODOLOGY

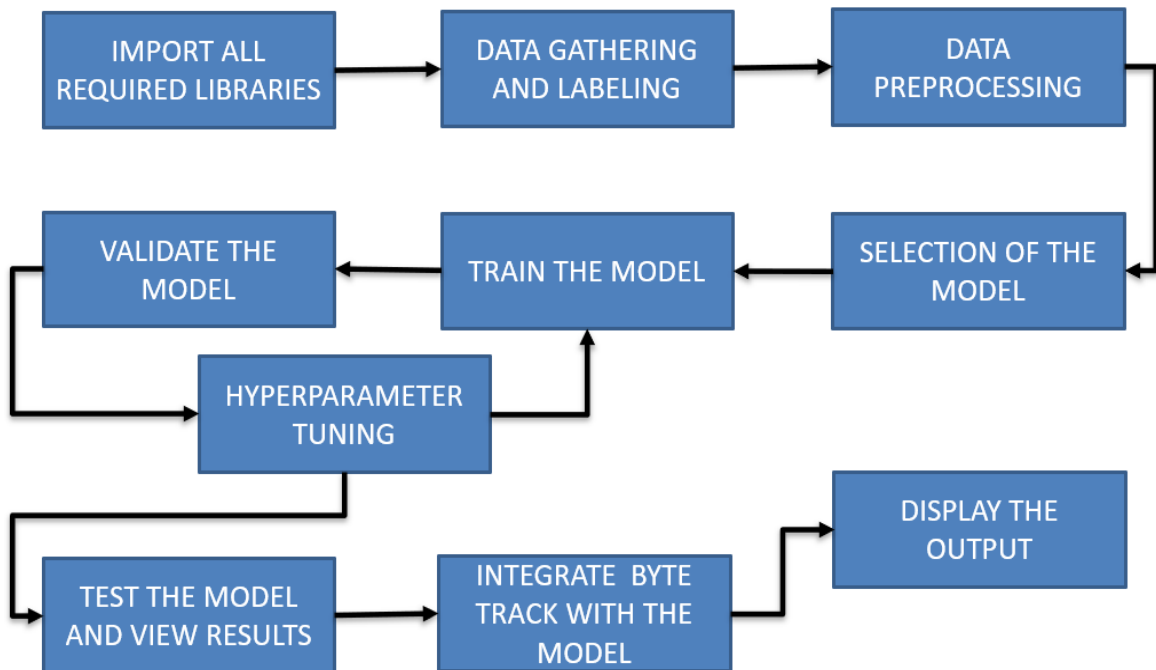


Fig 4.1 FLOW DIAGRAM FOR THE PROPOSED SYSTEM

This above diagram represents the flow of the project. Initially the data is collected from various resources and annotated with the help of “Roboflow” Software. The data collected is of different situations and different backgrounds. The annotated data is used to train the model after training the model is validated with validation dataset. Hyperparameter tuning is done to improve the performance of the model. It is done using trial and error method. Test the model with the testing data to test the performance of the model. After that the Bytetrack algorithm is integrated with the model to count the number of vehicles crossing the imaginary line. The integrated model is used to predict the outputs provided with valid traffic video and display the results accordingly which can be used detect, classify, and count the number of vehicles.

The following are steps involved in vehicle detection, classification and counting:

Vehicle Detection

The first step entails gathering a sizable dataset of images and videos that include vehicles. The dataset needs to be varied and include a wide range of vehicle sizes, orientations, and types. Bounding boxes around the vehicles should be annotated on each image in the dataset. The annotations point out where the vehicles are in the picture and what class they belong to. To get them ready for use as input in the YOLOv8 model, the images in the dataset are pre-processed. The images must be cleaned up, their brightness and contrast adjusted, and their size standardized. Pre-processing lessens the influence of unimportant features in the input images, which helps increase the model's accuracy. The YOLOv8 model must then be trained using the annotated dataset. During training, the model gains the ability to recognize characteristics of vehicles like their size, colour, texture, and shape. The model can be used to identify vehicles in new images and videos after being trained. During inference, the YOLOv8 model applies several convolutional filters to the input image to identify objects. The model generates a set of bounding boxes and corresponding class probabilities for each object that is detected. The YOLOv8 model's output is post-processed to clean up any duplicate or false-positive detections and to sharpen the bounding boxes that were found. Non-maximum suppression (NMS) is used to eliminate overlapping detections, a confidence threshold is set to eliminate low-confidence detections, and the bounding boxes are adjusted to better fit the detected objects.

Classification of vehicles

As there are so many different types of vehicles, classifying them all is a monumental task. As a result, the dataset is divided into vehicles of five categories: cars, buses, motorcycles, trucks, and ambulances. The steps involved in creating the model for the classification of vehicles are as follows:

Cloning the YOLO model from GitHub and installing all the prerequisites. To use the YOLO model, one must clone the repository from GitHub and install all the required libraries, such as NumPy, torch, SciPy, TensorFlow, and so on. Collection

of data and labeling them with their respective classes like cars, buses, trucks, motorcycles, and ambulances. It takes a lot of data and processing power to train a YOLO model for the classification of vehicles, which is a challenging task. Collecting data, annotating it, training a model, and evaluating it is important for the model's performance, as data plays a crucial role in the model's performance.

The object detector's level of certainty that there is an object present inside a specific bounding box is expressed by a scalar value known as the confidence value. The confidence value, one of the YOLO model's outputs, is used to eliminate. Low-confidence detections that could be false positives or have incorrect localization.

In YOLO, each bounding box is assigned a confidence value, and only those detections with a high enough confidence value are retained for further processing. This helps to ensure that the model only makes predictions for objects about which it is very confident and that it is not affected by false positive detections. YOLO outputs the bounding box's location, the object's class label, the box's width, and its height in addition to the confidence level. These outputs, which are also used for object detection, are used to draw the bounding boxes around the objects in an image.

Data Set Preparation

The first step in training a YOLO model is to collect a dataset of images that contains the objects you want to detect and classify. This dataset should be diverse, with images that vary in lighting conditions, camera angles, and object placements. Once you have your dataset, you need to label the images with bounding boxes indicating the location of the objects in the image and their corresponding class labels. This labeling process can be time-consuming and tedious, but it is crucial for training an accurate YOLO model. Annotation tools like "Roboflow" can be used to speed up the process.

Data Preprocessing

Data preprocessing is a crucial step in machine learning that involves cleaning, transforming, and normalizing the data to prepare it for model training. In the context

of vehicle detection and classification, data preprocessing is especially important to ensure that the models can accurately identify and classify vehicles in real-world scenarios. One important aspect of data preprocessing for vehicle detection and classification is data cleaning. This involves removing any invalid or incomplete data from the dataset, such as images that are corrupted or contain missing data. This ensures that the models are trained in high-quality data, which can improve their accuracy and robustness. Another key step in data preprocessing is data normalization.

Resizing

YOLO divides the input image into a grid of cells, with each cell responsible for forecasting a set of bounding boxes and associated class probabilities for the objects in the cell. For each object found in the image, the YOLO model produces a set of bounding boxes and associated class probabilities. The different objects in the image can be recognized categorized using these bounding boxes, such as the cars in a traffic scene.

Cropping is a common image processing technique used in computer vision, including in vehicle detection and classification. It involves removing parts of an image to create a smaller, focused region of interest, which can be useful for eliminating irrelevant background and improving the accuracy of the model.

There are several reasons why cropping may be used in vehicle detection and classification. For example, in a traffic surveillance system, the goal may be to identify and track individual vehicles within a larger scene. Cropping can be used to extract the region of the image that contains the vehicle, while eliminating the surrounding background. This can make it easier for the model to identify and classify the vehicle accurately. Another reason for cropping may be to standardize the size of images. In many datasets, images may have different sizes and aspect ratios, which can make it difficult to train a model that is sensitive to the size and orientation of objects.

Normalization

Normalization is necessary for the model to learn more quickly and with higher accuracy, normalizing the pixel values of the images is crucial for detecting the objects. This is carried out because normalizing the input enables the gradients to flow more effectively across the network as the model's weights are initialized using a tiny random number. In this manner, the model can learn more quickly and arrive at a better answer. In normalization, the input images are scaled between 0 and 1. Both Images and Bounding boxes are normalized which helped the model more to learn efficiently.

Train, Validate, and Test the model

Training and validation play an important role in measuring the performance of the model. From the collected data the model is trained with 70 percent of data, 20 percent for validation of the model, and the remaining for testing of the model's performance. The training process involves feeding the labeled training dataset into the YOLO model and adjusting the weights to minimize the detection error. During training, the YOLO model predicts bounding boxes and class probabilities for the objects in the image. These predictions are compared to the ground-truth bounding boxes and class labels in the training dataset. The loss function is then calculated based on the difference between the predicted and ground-truth values. The weights of the model are updated using backpropagation to minimize the loss. The training process is iterative, with the model being updated after each batch of images is processed. The goal of training is to reduce the loss function, which indicates the difference between the predicted and ground-truth values. A lower loss function indicates a more accurate model. Validation is employed to confirm the vehicles' classification. After the model is trained, tuning of parameters is necessary for better performance hence, parameters and hyperparameters are used to further increase the performance of the model and avoid overfitting problems.

Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
71/80	12.1G	0.04959	0.1783	0.9199	2	640: 100% 274/274 [03:18<00:00, 1.38it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 39/39 [00:25<00:00, 1.54it/s]
	all	1233	1233	0.878	0.862	0.931 0.929
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
72/80	12.1G	0.04346	0.1066	0.9136	2	640: 100% 274/274 [03:17<00:00, 1.39it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 39/39 [00:25<00:00, 1.54it/s]
	all	1233	1233	0.896	0.866	0.936 0.93
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
73/80	12.1G	0.04435	0.1013	0.9187	2	640: 100% 274/274 [03:17<00:00, 1.39it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 39/39 [00:25<00:00, 1.54it/s]
	all	1233	1233	0.911	0.84	0.934 0.926
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
74/80	12.1G	0.04178	0.08403	0.9157	2	640: 100% 274/274 [03:17<00:00, 1.39it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 39/39 [00:25<00:00, 1.54it/s]
	all	1233	1233	0.93	0.856	0.938 0.931
Epoch	GPU_mem	box_loss	cls_loss	dfl_loss	Instances	Size
75/80	12.1G	inf	0.08609	0.9209	2	640: 100% 274/274 [03:16<00:00, 1.39it/s]
	Class	Images	Instances	Box(P	R	mAP50 mAP50-95): 100% 39/39 [00:25<00:00, 1.54it/s]
	all	1233	1233	0.922	0.85	0.936 0.93

Fig.4.2 Training the Model With the collected data with 80 Epochs

Hyperparameter Tuning

Hyperparameter tuning is an important step in vehicle classification using deep learning algorithms. Hyperparameters are parameters that are not learned by the neural network during training, but instead are set by the user before training begins. Examples of hyperparameters in vehicle classification include the learning rate, number of epochs, batch size, and the number of hidden layers in the neural network. Hyperparameter tuning involves systematically adjusting these hyperparameters to find the combination that produces the best results. This is typically done using a combination of trial-and-error and automated optimization techniques, such as grid search, random search, or Bayesian optimization. The goal of hyperparameter tuning is to find the set of hyperparameters that produces the best performance on a validation set.

Counting of vehicles

Using YOLO to count vehicles can yield useful data that can be utilized to boost productivity, security, and safety in a variety of contexts. YOLO's real-time vehicle counting capability enables businesses to make data-driven decisions that can enhance operations and eventually result in cost savings. To count the vehicles in a video drawing a line is necessary and the count variable is declared with a value of zero.

Vehicles are counted whenever they cross the line and each time the vehicle is crossed the count variable is incremented and at the end count is returned. YOLO implements a technique to assure that, even if a vehicle appears in several frames, it is only recorded once. YOLO compares the bounding boxes of the detected vehicles across frames and only counts a vehicle as new if its bounding box is distinct from any other detected bounding boxes. Once the vehicles are detected in the video frames, the tracking network is used to track their movements across frames. The tracking network uses an algorithm called the Siamese network, which compares the features of the detected vehicles across frames and matches them to the same object. The tracking network can track vehicles even if they are partially obscured or if there are changes in lighting or other environmental factors.

4.2 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION AND TESTING PLAN OF THE PROPOSED SYSTEM

4.2.1 DESCRIPTION OF SOFTWARE FOR IMPLEMENTATION

GOOGLE COLLAB

Google Collab is a free cloud-based service provided by Google that allows users to write, run and share Python code in a Jupyter notebook environment. It provides users with a virtual machine that comes pre-installed with Python, various Python libraries such as TensorFlow, PyTorch, and OpenCV, and allows for GPU and TPU (Tensor Processing Unit) acceleration. Google Collab provides users with access to a range of powerful hardware configurations, including CPUs, GPUs, and TPUs, which can be used to train machine learning models and perform other computationally intensive tasks. It also allows users to save their work to Google Drive, collaborate with others in real-time, and share their notebooks with anyone.

GIT-HUB

We use Git-hub for cloning repositories like YOLO and uploading the files into Git-hub and saving them there. GitHub also supports the development of open-source

software, which is software with source code that anyone can inspect, modify, and enhance the models. We might also use Kaggle for training the model as it supports faster execution as it supports GPU.

KAGGLE NOTEBOOK

Kaggle Notebook is a cloud-based platform that allows data scientists and researchers to develop and share machine learning and data analysis projects. The platform is powered by Jupyter notebooks and provides access to powerful computing resources, including GPUs and TPUs, to help accelerate model training. Kaggle Notebooks are created and stored on Kaggle servers, which means that users can access their notebooks from anywhere with an internet connection. Users can also collaborate on notebooks with others and share their work publicly or privately.

4.2.2 TESTING PLAN OF THE PROPOSED SYSTEM

A testing plan for vehicle detection and classification would involve a series of steps to ensure the accuracy and effectiveness of the system. Firstly, the input data would need to be gathered and prepared for testing, which would involve collecting various images of vehicles in different scenarios. The dataset should include a wide range of vehicles, from small cars to large trucks, and should be representative of the types of vehicles the system will encounter in real-world scenarios. The data should also be properly labeled to indicate the ground truth information, such as the position, size, and class of each vehicle in the image. To test the performance of the model the dataset which consists of 1200 images is used which is approximately 20 percent of the total data collected. Then, the YOLO algorithm would be trained using this data to accurately detect and classify vehicles. Once the algorithm has been trained, the model needs to be tested to evaluate its performance. This would include testing the algorithm on different types of vehicles, in different lighting conditions, and in different environments. The system would also be tested for its ability to accurately detect and classify multiple vehicles at the same time, as well as its ability to detect and classify vehicles in motion. To evaluate the performance of the YOLO algorithm, a set of performance metrics should be defined. These

metrics should be selected based on the specific requirements of the system, and include measures such as precision, recall, F1 score, and mean average precision (mAP). Precision measures the proportion of true positives (correctly detected vehicles) out of all the vehicles that were detected by the system. Recall measures the proportion of true positives out of all the actual vehicles in the scene. F1 score is the harmonic mean of precision and recall. mAP measures the average precision of the system across all possible thresholds. These performance metrics will be used to evaluate the accuracy and effectiveness of the YOLO algorithm and to identify areas where the system may need improvement.

4.3 PROJECT MANAGEMENT PLAN

WEEKS	PLAN
WEEK 1	COLLECTION OF VEHICLE DATASET FROM VARIOUS SOURCES AND LOAD IT TO ROBOFLOW SOFTWARE.
WEEK 2 - 4	ANNOTATING AND LABELING DATA WITH THEIR RESPECTIVE CLASSES USING ROBOFLOW SOFTWARE.
WEEK 5-8	DEVELOPING, TRAINING THE MODEL AND TESTING, MEASURING THE PERFORMANCE OF THE TRAINED MODEL.
WEEK 9-12	FINALIZING THE BEST METHODOLOGIES AND IMPROVING THE PERFORMANCE OF THE MODEL BY TUNING THE PARAMETERS AND SAVING THE WEIGHTS.
WEEK 13-15	DEVELOPING, INTEGRATING BYTETRACK AND THE TRAINED MODEL IN ORDER TO COUNT VEHICLES AND DISPLAY THE RESULTS.
WEEK 16-18	PREPARING DOCUMENTATION, RESEARCH PAPER FOR PUBLISHING AND PUBLISHING THE PAPER.

CHAPTER 5

IMPLEMENTATION DETAILS

5.1 ALGORITHMS

5.1.1 OPEN-CV

OpenCV provides a wide range of functions and algorithms for image and video processing, including image filtering, edge detection, feature detection, object detection, tracking, and machine learning. These functions can be used to analyze and manipulate images and video streams in real-time and are commonly used in a range of applications, such as robotics, surveillance, medical imaging, and autonomous vehicles. One of the key challenges in vehicle detection and classification is processing large amounts of image and video data in real-time. OpenCV offers a range of image and video processing functions that can be used to preprocess the data and reduce the computational complexity of the algorithm. For example, image resizing can be used to reduce the resolution of the input image, which can significantly reduce the amount of processing required. Color conversion functions can also be used to convert the image to grayscale or other color spaces, which can help to highlight certain features or reduce noise in the image.

Once the image has been preprocessed, YOLO can be used to detect the vehicles in the image. YOLO is a deep learning algorithm that can detect objects in real-time from images and video streams. It uses a single neural network to predict the bounding boxes and class probabilities of the objects in an image. OpenCV provides functions for loading the pre-trained YOLO model and running inference on the input image. The output of this stage is a set of bounding boxes that represent the location of the detected vehicles in the image.

After the vehicles have been detected, OpenCV can be used to extract features from the bounding boxes. Features can be used to classify vehicles into different categories, such as cars, trucks, and motorcycles. OpenCV provides several functions for feature extraction and classification, such as Histogram of Oriented Gradients (HOG), Local Binary Patterns (LBP). HOG and LBP are both feature

Input Layer

The input layer of YOLO is responsible for taking the input image and resizing it to a fixed size. The fixed size is usually set to 416*416 or 640*640 pixels, but it can be adjusted based on the specific application. The input image is then divided into a grid of cells, with each cell responsible for predicting the objects that appear in its region. The size of the grid is determined by the hyperparameter, which is usually set to 19x19 or 13x13.

Convolutional Layers

The convolutional layers in YOLO are responsible for extracting features from the input image. The convolutional filters slide over the input image and perform a convolution operation to extract features. The size of the filters is typically set to 3x3 or 1x1, and the number of filters gradually increased as the image is passed through more convolutional layers. The convolutional layers are also responsible for reducing the spatial resolution of the features, which helps to reduce the computation required in the later stages of the network.

Downsampling Layers

The downsampling layers in YOLO are responsible for reducing the spatial resolution of the features extracted by the convolutional layers. The downsampling is typically achieved by applying max pooling or stride convolution operations. The downsampling helps to reduce the computation required in the later stages of the network and increases the receptive field of the network.

Fully Connected Layers

The fully connected layers in YOLO are responsible for predicting the location and class of the objects in the input image. The fully connected layers take the features extracted by the convolutional and downsampling layers and perform a series of

linear transformations. The output of the fully connected layers is then passed through an activation function to predict the object class probabilities and the bounding box coordinates.

Output Layer

The output layer of YOLO consists of a set of bounding boxes that represent the location of objects in the image. Each bounding box contains a predicted object class and a confidence score. The confidence score represents how confident YOLO is that the object is present in the bounding box. The predicted object class is one of the pre-defined classes that the system has been trained to recognize. YOLO uses a SoftMax activation function to predict the object class probabilities.

The Softmax function is used to convert the output of the YOLO neural network into a probability distribution over a set of discrete categories. In the context of YOLO, the output of the neural network is a set of bounding boxes and associated confidence scores, each of which represents the location and probability of an object in the input image. The Softmax function is used to convert the confidence scores for each bounding box into a probability distribution over the set of discrete object categories that the network is trained to recognize. The Softmax function takes as input a vector of scores and outputs a vector of probabilities that sums to one. In YOLO, the Softmax function is applied to the confidence scores for each bounding box to obtain a probability distribution over the set of object categories. The bounding box with the highest probability for each category is then selected as the final output for the YOLO algorithm.

5.1.3 BYTE TRACK

ByteTrack is a deep learning-based object tracking algorithm that uses a single-stage detector to detect and track objects in video sequences. The algorithm is based on the idea of using temporal information to improve the accuracy of object detection and tracking. In this section, we will explain how ByteTrack works and how it can be used for various applications.

Architecture of ByteTrack

The architecture of ByteTrack consists of three main components: the feature extraction network, the detection network, and the tracking module. The feature extraction network is used to extract features from the input video frames. The detection network is used to detect objects in the video frames, and the tracking module is used to track the objects across frames.

Feature Extraction Network

The feature extraction network is used to extract features from the input video frames. The features are used by the detection network to detect objects in the video frames. The feature extraction network is typically a convolutional neural network (CNN) that is pre-trained on a large dataset, such as ImageNet. The pre-trained CNN is used as a feature extractor, and the features are fed into the detection network.

Detection Network

The detection network is used to detect objects in the video frames. The detection network is a single-stage detector that uses a modified version of the YOLO (You Only Look Once) detector. The YOLO detector is a popular object detection algorithm that uses a single neural network to predict the bounding boxes and the class probabilities of the objects in the image.

The detection network takes the features extracted by the feature extraction network as input and predicts the bounding boxes and the class probabilities of the objects in the video frame. The detection network outputs a set of bounding boxes and their corresponding class probabilities for each object detected in the frame.

ByteTrack can be integrated into the trained model to track the movements of the vehicles which can be used for counting the number of vehicles passing through the region of interest.

Tracking Module

The tracking module is used to track the objects across frames. The tracking module uses a simple, yet effective tracking algorithm called Tracktor. Tracktor is based on the correlation filter and is used to estimate the position of the object in the next frame based on its position in the current frame.

The tracking module takes the bounding boxes predicted by the detection network as input and tracks the objects across frames. The tracking module estimates the position of the objects in the next frame based on their position in the current frame and the motion of the objects. The tracking module outputs the estimated position of the objects in the next frame.

Cosine Distance Metrics

Cosine distance is a commonly used distance metric in machine learning and computer vision for measuring the similarity between two vectors. In ByteTrack, a popular object detection and tracking algorithm, cosine distance is used to compare the features of objects in different frames to track their movement and estimate their velocity. ByteTrack uses a Siamese network architecture to extract features from the objects in each frame. The Siamese network consists of two identical networks with shared weights, where each network takes as input an image patch containing an object and produces a feature vector. The two feature vectors are then compared using cosine distance to compute a similarity score. If the similarity score is above a certain threshold, the object is the same as the object in the previous frame, and its position and velocity are updated accordingly.

Cosine distance is chosen as the distance metric because it is robust to changes in scale and lighting conditions, which are common challenges in object detection and tracking. Cosine distance measures the angle between two vectors, rather than their magnitude, which makes it insensitive to the overall scale of the vectors. This makes cosine distance a good choice for comparing features extracted from different

images, which may have different lighting conditions or scales.

In ByteTrack, cosine distance is also used to compute the appearance affinity matrix, which measures the similarity between the features of all pairs of objects in each frame. The appearance affinity matrix is used to group objects with similar appearance into the same track, and to associate objects with their corresponding tracks in subsequent frames. The use of cosine distance in computing the appearance affinity matrix helps to improve the accuracy of the object tracking algorithm by effectively handling changes in appearance due to lighting conditions, occlusions, and other factors.

5.1.4 DEEPSORT

Deport (Deep Learning-based SORT) is a tracking algorithm that is commonly used in vehicle counting and tracking applications. It is an improvement upon the SORT algorithm, which uses a Kalman filter to track the position and velocity of objects in a video stream. DeepSORT, on the other hand, uses a deep neural network to associate detections with existing tracks. This allows the algorithm to handle situations where multiple vehicles are present in the scene, and where vehicles may enter or leave the scene without being detected. The DeepSORT algorithm consists of three main stages: detection, feature extraction, and data association. In the detection stage, object detection algorithms such as Faster R-CNN, YOLO, and SSD are used to identify vehicles present in each frame of the video. These algorithms use convolutional neural networks (CNNs) to detect and localize objects in the image and output a set of bounding boxes and corresponding confidence scores for each object. In the feature extraction stage, features such as color, texture, and shape are extracted from each detected vehicle. These features are used to represent the vehicle in a high-dimensional feature space, where vehicles with similar features are close to each other. This is important for the data association stage, as it allows the algorithm to match detections with existing tracks based on the similarity of their features.

Finally, in the data association stage, the algorithm uses a nearest neighbor approach to associate detections with existing tracks. Specifically, for each detection in the current frame, the algorithm searches for the track that is closest to it in the feature space. If the distance between the detection and the track is below a certain threshold, the detection is associated with the track. If there is no track that is close enough to the detection, a new track is created for the detection.

One of the key advantages of DeepSORT over other tracking algorithms is its ability to handle situations where vehicles are occluded or temporarily leave the scene. When a vehicle becomes occluded, the algorithm can still track it using its previously learned features and can continue to track it once it reappears in the scene. Similarly, when a vehicle temporarily leaves the scene, the algorithm can predict its position and velocity based on its previous motion and can track it once it reappears. DeepSORT has a wide range of applications in vehicle counting and tracking. One common use case is in traffic monitoring, where it can be used to count the number of vehicles passing through a particular intersection or stretch of road. In this application, the algorithm can be used to differentiate between different types of vehicles, such as cars, buses, and trucks, and can provide real-time data on traffic flow and congestion.

Another application of DeepSORT is in surveillance, where it can be used to track vehicles of interest, such as those involved in criminal activity or suspicious behavior. In this application, the algorithm can be used to track the movement of the vehicle over time, and can provide data on its location, speed, and direction of travel. Overall, DeepSORT is an effective and robust algorithm for vehicle counting and tracking applications. Its ability to handle occlusions and track vehicles over time makes it well-suited for a wide range of applications in both industry and academia. As computer vision technology continues to advance, it is likely that DeepSORT and similar algorithms will become even more powerful and versatile, opening new possibilities for vehicle tracking and analysis.

5.1.5 KALMAN FILTER

Kalman filter is a mathematical algorithm used to estimate the state of a system in the presence of uncertain and noisy measurements. It is widely used in a variety of fields, including control systems, signal processing, robotics, and computer vision. The Kalman filter works by combining a prediction of the state of the system based on a model of its behavior with a measurement of the system's output to obtain an improved estimate of the system's state. The filter considers both the uncertainties in the model and the measurement and provides a statistically optimal estimate of the state.

The Kalman filter consists of two main steps: prediction and update. In the prediction step, the filter predicts the state of the system at the next time step based on the current state and a model of the system's behavior. The prediction is computed using a state transition matrix, which relates the state at the current time step to the state at the next time step, and a control input, which represents the effect of any external inputs on the system. The prediction step also computes the error covariance matrix, which represents the uncertainty in the state prediction. This matrix considers the uncertainty in the system model and the effect of any noise in the state and control inputs.

In the update step, the filter combines the predicted state with a measurement of the system's output to obtain an improved estimate of the state. The measurement is modeled as a linear function of the state plus an additive Gaussian noise term. The Kalman filter computes the Kalman gain, which determines the amount of weight given to the predicted state and the measurement in the updated estimate. The updated estimate is then computed as a weighted average of the predicted state and the measurement, with the weights determined by the Kalman gain. The update step also computes the error covariance matrix for the updated estimate, which considers the uncertainty in the measurement and the effect of the Kalman gain on the error covariance matrix.

One of the key advantages of the Kalman filter is its ability to provide an optimal estimate of the system state, even in the presence of noise and uncertainty. The filter considers both the uncertainty in the system model and the measurement and provides a statistically optimal estimate of the state. The filter is widely used in control systems, where it is used to estimate the state of a system based on noisy sensor measurements.

5.2 TESTING OF THE MODEL

Testing the model of vehicles is an important step in the development of vehicle detection and classification algorithms. The goal of testing is to evaluate the performance of the algorithm on a set of test data that is separate from the training data. The performance of a vehicle detection and classification model can be evaluated using several metrics. The most common evaluation metrics include:

Mean Average Precision (mAP)

It is a widely used evaluation metric in the field of computer vision, particularly for object detection. It is used to evaluate the accuracy of an object detection algorithm by comparing the predicted bounding boxes of objects in an image or video sequence to the ground truth bounding boxes.

mAP is calculated by first calculating the average precision (AP) for each class in the dataset. AP is a measure of the quality of the model's predictions for a given class, and it considers both the precision and recall of the model's predictions. Precision is the fraction of true positives (correctly predicted objects) among all predicted objects, while recall is the fraction of true positives among all actual objects in the image. To calculate the AP for a given class, the model's predictions for that class are sorted based on their confidence score (i.e., the probability that the predicted object is of that class). A threshold is then applied to the predictions to determine which predicted objects are considered true positives and which are considered false positives. A true positive is a predicted object that has an overlap

of at least 0.5 with a ground truth object (i.e., a correct bounding box). A false positive is a predicted object that does not have an overlap of at least 0.5 with any ground truth object.

Once the true positives and false positives have been determined for each prediction, the precision and recall can be calculated at each threshold. Precision is calculated as the number of true positives divided by the total number of predicted objects, while the recall is calculated as the number of true positives divided by the total number of ground truth objects.

The precision-recall curve for the class is then plotted by varying the threshold and calculating the precision and recall at each threshold. The AP for the class is then calculated as the area under the precision-recall curve. An AP of 1 indicates perfect precision and recall, while an AP of 0 indicates poor performance.

The mAP for the model is then calculated as the average of the APs for all classes in the dataset. The mAP gives an overall measure of the accuracy of the model across all classes.

P-R CURVE

The P-R curve (Precision-Recall curve) is a popular evaluation metric used in object detection tasks, including YOLO training. The P-R curve shows the relationship between the precision and recall of a model as the detection threshold varies. Precision is defined as the proportion of true positives (correctly detected objects) among all detected objects, and recall is defined as the proportion of true positives among all objects that should have been detected. In YOLO training, the P-R curve is used to evaluate the performance of the model by measuring the precision and recall at different confidence levels. The confidence level is the threshold at which YOLO decides whether an object is present in the image or not.

During training, the P-R curve can be used to tune the confidence level of the model

to maximize the overall precision and recall. This is done by adjusting the threshold level to find a balance between precision and recall that produces the best overall performance. To generate the P-R curve during training, the ground truth labels and the predicted labels from the YOLO model are compared. The precision and recall are then calculated for each threshold level, and the results are plotted on a graph. Ideally, the P-R curve should show a high precision and high recall at lower confidence levels, indicating that the model is correctly detecting most of the objects in the image while keeping false positives to a minimum.

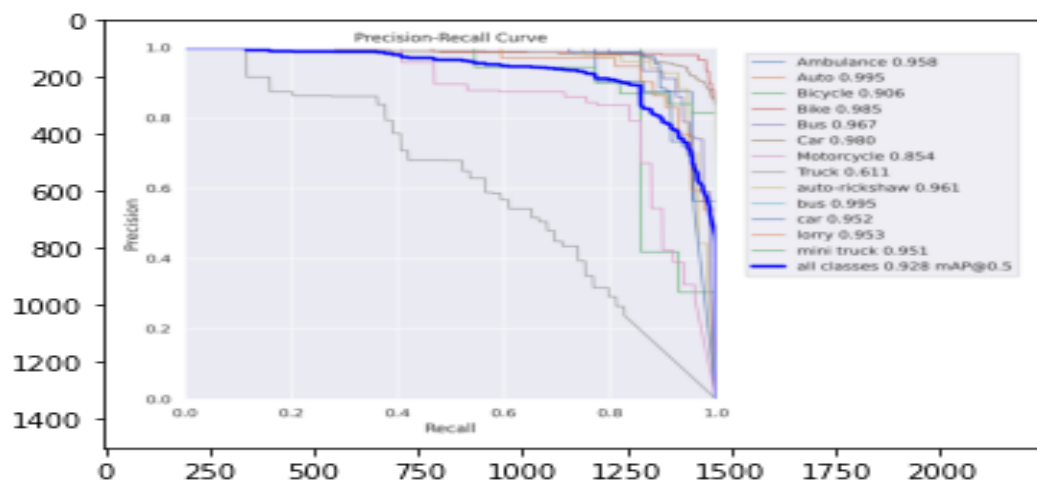


Fig 5.2 P-R CURVE FOR VEHICLE CLASSIFICATION

Confusion Matrix

Confusion matrix can be used to evaluate the performance of a classification model that is used to classify vehicles based on their type (e.g., car, truck, motorcycle, etc.). Here's an example of a confusion matrix for a binary classification problem that aims to distinguish between cars and non-cars:

True Positives (TP) The number of actual cars that were correctly classified as cars by the model.

False Positives (FP) The number of non-cars (e.g., trucks, motorcycles) that were incorrectly classified as cars by the model.

False Negatives (FN) The number of cars that were incorrectly classified as non-cars by the model.

True Negatives (TN) The number of non-cars that were correctly classified as non-cars by the model.

From this confusion matrix, several metrics can be computed to evaluate the performance of the classification model:

Accuracy The overall percentage of correct predictions made by the model. It is computed as $(TP + TN) / (TP + FP + TN + FN)$.

Precision The fraction of positive predictions that are correct. It is computed as $TP / (TP + FP)$.

Recall The fraction of actual positives that were correctly classified by the model. It is computed as $TP / (TP + FN)$.

F1 score The harmonic means of precision and recall, which provides a balanced measure of the model's performance. It is computed as $2 * (precision * recall) / (precision + recall)$.

These metrics can be used to assess the strengths and weaknesses of the classification model, and to identify areas where the model can be improved. For example, a high false positive rate may indicate that the model is incorrectly classifying non-cars as cars, while a high false negative rate may indicate that the model is failing to recognize certain types of cars. By analyzing the confusion matrix and associated metrics, improvements can be made to the classification model to optimize its performance for the specific task of vehicle classification.

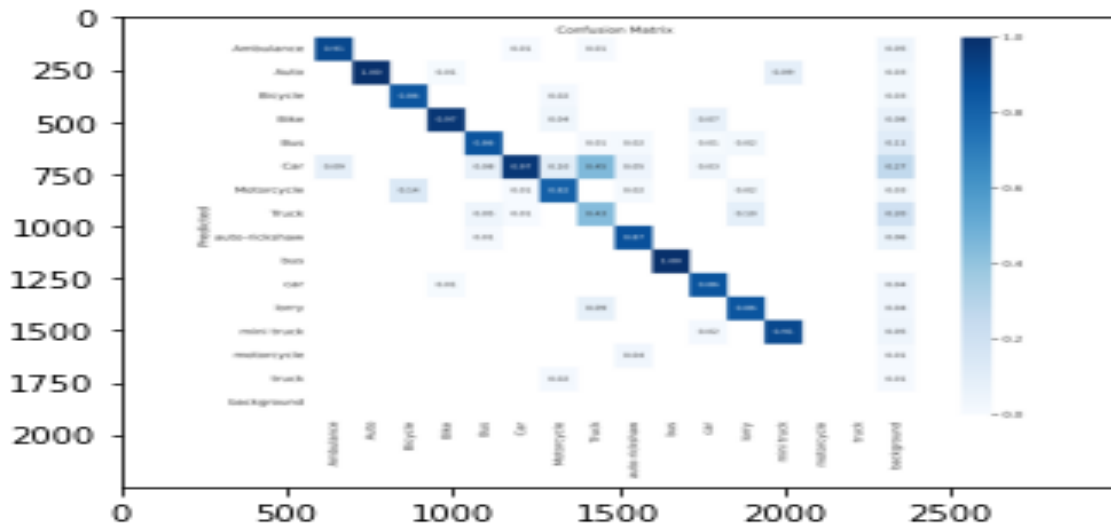


Fig.5.3 Confusion Matrix of different classes

Average Precision

Average Precision (AP) is a popular metric for evaluating the performance of object detection and classification models, including those used for vehicle classification. The AP is calculated by measuring the area under the Precision-Recall curve, which represents the trade-off between the true positive rate (TPR) and the false positive rate (FPR) of the model. In the context of vehicle classification, the classification model is applied to a set of test images, and the Precision-Recall curve is computed for each vehicle type by varying the classification threshold and computing the TPR and FPR at each threshold. The AP is then computed as the area under the Precision-Recall curve, using numerical integration techniques such as the trapezoidal rule. The AP values for each vehicle type are then averaged to obtain the Mean Average Precision (mAP), which is a summary measure of the model's overall performance. In general, a higher AP value indicates better performance, as it means the model has a higher TPR for a given FPR. However, it is important to note that AP is a relative measure of performance and should be interpreted in the context of the specific application and dataset being used.

P-CURVE

The p-curve is a useful tool for evaluating the statistical significance of a set of results in vehicle detection. It is commonly used to assess the evidence for the existence of a true effect in the data, and to identify potential publication bias or selective reporting of results. In the context of vehicle detection, the p-curve can be used to evaluate the statistical significance of the model's performance on a particular dataset. The model's performance is measured on a set of test images, and the results are recorded as a set of detection scores. The detection scores are then plotted on a histogram, with the x-axis representing the range of possible detection scores, and the y-axis representing the frequency of detections at each score. The p-curve is then generated by plotting the cumulative distribution function (CDF) of the detection scores on a log scale. This shows the proportion of true positive detections that have a detection score greater than a given threshold. The p-curve is then analyzed using statistical techniques to determine if there is evidence of a true effect.

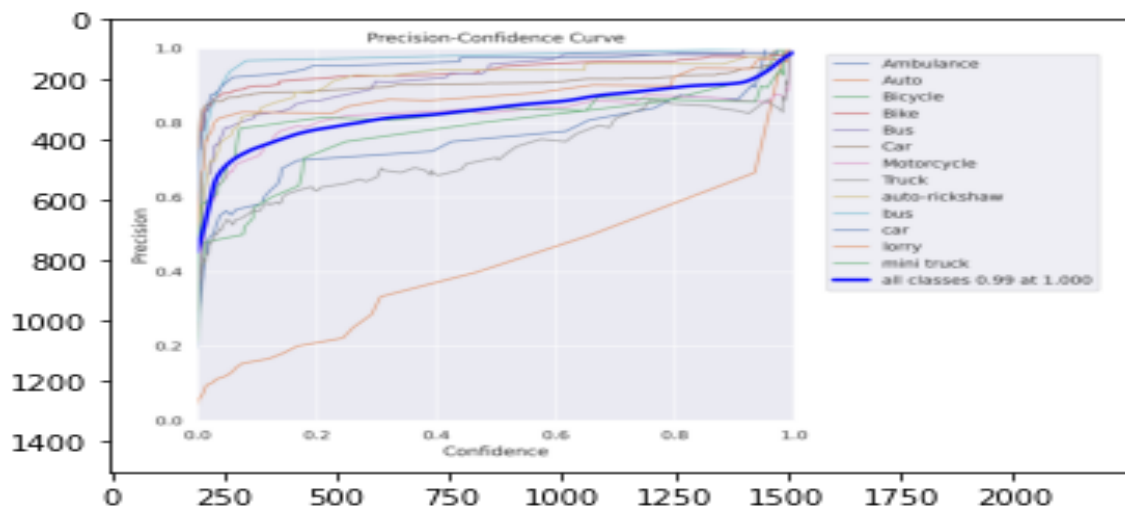


Fig.5.4 Precision-Confidence Curve

F1 SCORE

F1 score is a widely used metric in machine learning and statistics, particularly in binary classification tasks, including vehicle classification. It is a measure of the

balance between the precision and recall of a model and provides a way to evaluate the overall performance of the model. In binary classification, there are four possible outcomes: true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN). Precision is the proportion of true positives among all positive predictions, while recall (also called sensitivity) is the proportion of true positives among all actual positive samples. The F1 score is a single summary measure that combines precision and recall into a single value and is defined as the harmonic mean of precision and recall: The F1 score ranges from 0 (worst) to 1 (best). A high F1 score indicates that the model has both high precision and high recall, meaning it is making accurate predictions while also avoiding false positives and false negatives. In vehicle classification, the F1 score can be used to evaluate the overall performance of the model, particularly when the dataset is imbalanced (i.e., one class has many more examples than another).

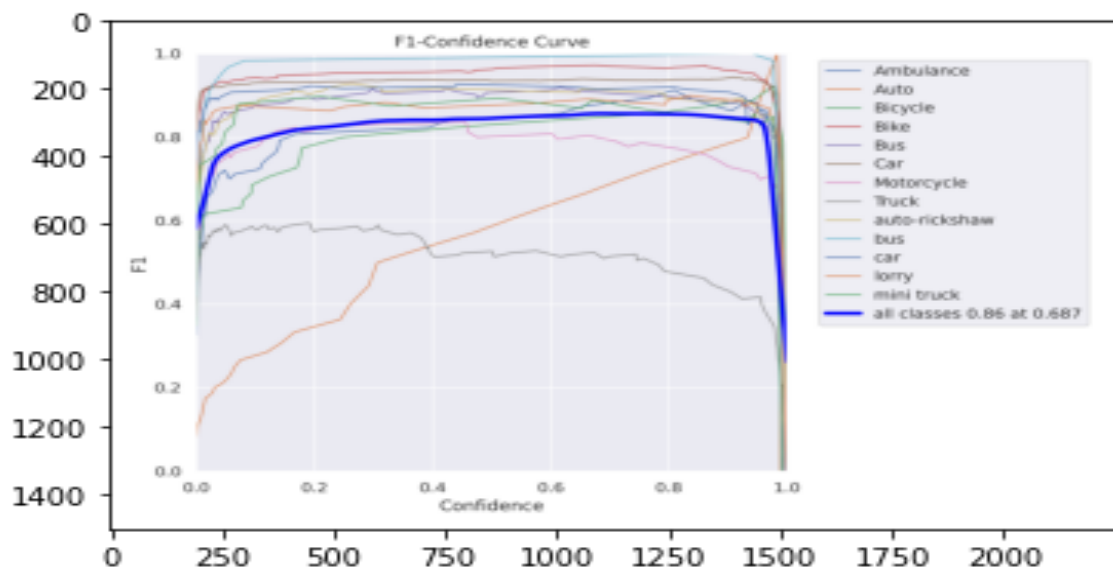
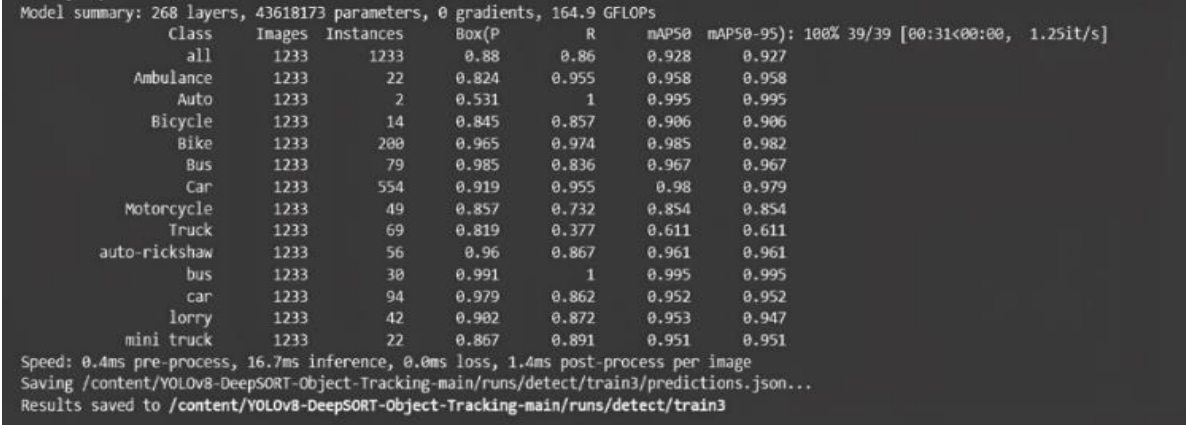


Fig 5.5 F1-Confidence Curve

CHAPTER 6

RESULTS AND DISCUSSION

An image dataset of vehicles is used for model training and evaluation in this study on YOLO-based vehicle detection and classification. The dataset included representations of various vehicle types and orientations. The model's overall accuracy was found to be 92.8%, with a mAP (50-95) of 92.7%. These outcomes show how accurately the model can identify and categorize vehicles. The YOLO model could process images in real time despite having a relatively high level of computational complexity. real-time. The computational complexity of the YOLO model was relatively high, but it was still capable of processing images in real time.



```

Model summary: 268 layers, 43618173 parameters, 0 gradients, 164.9 GFLOPs
Class Images Instances Box(P) R mAP50 mAP50-95: 100% 39/39 [00:31<00:00, 1.25it/s]
all 1233 1233 0.88 0.86 0.928 0.927
Ambulance 1233 22 0.824 0.955 0.958 0.958
Auto 1233 2 0.531 1 0.995 0.995
Bicycle 1233 14 0.845 0.857 0.906 0.906
Bike 1233 200 0.965 0.974 0.985 0.982
Bus 1233 79 0.985 0.836 0.967 0.967
Car 1233 554 0.919 0.955 0.98 0.979
Motorcycle 1233 49 0.857 0.732 0.854 0.854
Truck 1233 69 0.819 0.377 0.611 0.611
auto-rickshaw 1233 56 0.96 0.867 0.961 0.961
bus 1233 30 0.991 1 0.995 0.995
car 1233 94 0.979 0.862 0.952 0.952
lorry 1233 42 0.902 0.872 0.953 0.947
mini truck 1233 22 0.867 0.891 0.951 0.951
Speed: 0.4ms pre-process, 16.7ms inference, 0.0ms loss, 1.4ms post-process per image
Saving /content/YOLOv8-DeepSORT-Object-Tracking-main/runs/detect/train3/predictions.json...
Results saved to /content/YOLOv8-DeepSORT-Object-Tracking-main/runs/detect/train3

```

FIG 6.1 PERFORMANCE OF THE MODEL

The common metric used to evaluate the performance of vehicle detection and classification systems is mean average precision (mAP), which measures the accuracy of object detection algorithms.

The model performed well for all vehicle classes, with auto having the highest accuracy (99.5%) and trucks having the lowest (61.1%). Although the model could be improved in future studies, it had trouble detecting small vehicles, like motorcycles. When compared to other methods, the trained model outperformed them in terms of recall and accuracy. These findings show how well YOLO performs when used as a model for vehicle detection and classification tasks, and they suggest that many practical uses, including traffic monitoring and autonomous

driving, may be possible. The model will be tested on a larger and more varied dataset in the future work, and strategies to enhance the model's performance on small vehicles will be investigated.

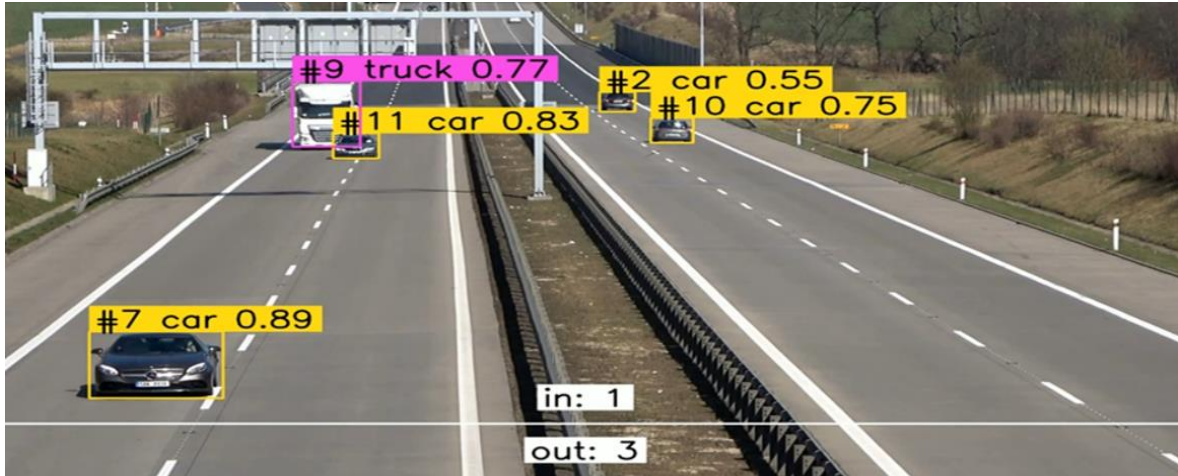


FIG 6.2 FINAL OUTPUT

The above figure shows the imaginary line that was drawn to count the number of vehicles crossing the specific line, the number of incoming and outgoing vehicles, and the classes of each group of vehicles. Despite these promising results, there are also some limitations and challenges associated with vehicle detection and classification systems. One issue is the potential for bias in the algorithms used, which may lead to inaccuracies or unfair outcomes. Another issue is the robustness of these systems to changes in lighting, weather, and environmental conditions, which may require the use of adaptive or machine learning techniques to improve performance. In addition, there are ethical considerations associated with the use of vehicle detection and classification systems, particularly regarding privacy and data protection. It is important for practitioners to carefully consider these issues and implement appropriate safeguards and solutions to address them.

Overall, vehicle detection and classification systems can achieve high levels of accuracy and performance and have the potential to improve a range of applications, including traffic management, surveillance, and autonomous driving. However, these systems require careful planning, design, and implementation to ensure their effectiveness, reliability, and ethical responsibility.

CHAPTER 7

CONCLUSION

7.1 CONCLUSION

In conclusion, computer vision tasks like vehicle classification and identification are important because they have many applications, including traffic monitoring, self-driving cars, and intelligent transportation systems. For the classification and detection of vehicles, several deep learning-based methods, including convolutional neural networks (CNNs), You Only Look Once (YOLO) algorithms, template matching, and feature-based techniques, have been proposed.

In vehicle detection and classification using YOLO, the algorithm is trained on a large dataset of images that contain different types of vehicles. During training, YOLO learns to identify the unique features of each type of vehicle, such as the shape and size of the vehicle, and use this information to accurately detect and classify vehicles in new images. Once the model is trained, it can be used to perform vehicle detection and classification in real-time on video streams or images. The YOLO algorithm can detect multiple vehicles in a single image and classify them into different categories, such as cars, trucks, and buses, with high accuracy.

Moreover, YOLO can be integrated with other technologies such as OpenCV and deep learning frameworks such as TensorFlow and PyTorch to enhance its capabilities in vehicle detection and classification. The use of GPU (Graphics Processing Unit) acceleration and cloud computing platforms such as Google Collab can also greatly speed up the training and testing processes.

Due to their ability to extract specific information from photos and tolerate variations in lighting and viewpoint, CNNs have been demonstrated to be particularly effective for tasks involving the classification and detection of vehicles. The "YOLO" real-time object recognition technique uses CNN to recognize objects in an image or video. YOLO can provide useful information that can be used in a variety of situations to improve productivity, safety, and security because it allows for real-time monitoring and decision-making.

Overall, the adoption of deep learning-based techniques over the past few years has resulted in a significant improvement in the performance of vehicle classification and detection. There is still room for improvement, especially in handling occlusions and recognizing vehicles in low-light situations. One of YOLO's problems is that it has trouble detecting many objects at once when they are close to one another. Future studies in this area will concentrate on finding solutions to these problems and enhancing the functionality of vehicle classification and detection systems.

7.2 FUTURE WORK

YOLO's real-time image processing capabilities make it ideal for vehicle detection and classification. In practical applications like traffic monitoring and autonomous driving, where quick processing times are crucial, this makes it a suitable model to use. The simultaneous object detection and classification capabilities of YOLO can reduce computational complexity and increase system effectiveness. While many current vehicle detection and classification algorithms can classify vehicles into broad categories (e.g., car, truck, bus), future work could focus on developing algorithms that can classify vehicles into more fine-grained categories (e.g., sedan, SUV, pickup truck). With the increasing importance of data privacy, future work in this area could focus on developing vehicle detection and classification algorithms that can operate on encrypted data, preserving the privacy of the data while still allowing for accurate detection and classification.

However, there are some disadvantages to using YOLO for vehicle detection and classification. There are some disadvantages to it, such as a high computational resource requirement that might prevent its use in environments with few resources. Another flaw is that the model might not be able to detect small vehicles, such as motorcycles; this is a problem that could be resolved in future studies. To produce the best results, the model must also be modified for the specific task and dataset, which could take some time. Overall, YOLO is a powerful model for identifying and classifying vehicles, but it has some drawbacks. The focus of future research may be on improving the model's performance on small vehicles and developing more

efficient versions of the model that can run on hardware with less processing power.

7.3 RESEARCH ISSUES

There are several research issues in vehicle detection and classification that are currently being studied by researchers in the field. Some of these issues include:

Occlusion: One of the biggest challenges in vehicle detection and classification is occlusion, where one or more parts of the vehicle are obscured by other objects in the scene. This can make it difficult for algorithms to accurately detect and classify the vehicle.

Scale variation: Vehicles can vary significantly in size, from small cars to large trucks and buses. This can make it difficult for algorithms to accurately detect and classify vehicles across a range of scales.

Lighting conditions: Lighting conditions can have a significant impact on the accuracy of vehicle detection and classification algorithms. For example, images captured in low light conditions may be more difficult to analyze than those captured in bright sunlight.

Viewpoint variation: Vehicles can appear differently depending on the viewpoint from which they are observed. This can make it difficult for algorithms to accurately classify vehicles based on their appearance.

Dataset bias: The accuracy of vehicle detection and classification algorithms is heavily dependent on the quality and diversity of the training dataset used to train the algorithm. If the dataset is biased towards certain types of vehicles or certain lighting conditions, this can negatively impact the algorithm's accuracy.

Computational efficiency: As the size of datasets and complexity of algorithms continues to increase, computational efficiency is becoming an increasingly important issue in vehicle detection and classification. Algorithms that are too computationally expensive may be impractical for use in real-world applications.

Generalization: Vehicle detection and classification algorithms need to be able to

generalize to new and unseen environments. This can be challenging, as the appearance of vehicles can vary significantly depending on the environment in which they are observed.

7.4 IMPLEMENTATION ISSUES

These are the implementation issues in vehicle detection, counting and classification:

Hardware constraints: The performance of vehicle detection and classification algorithms can be heavily dependent on the hardware used to run them. For example, real-time detection and classification require high-performance hardware, such as GPUs or specialized processors, which may not be available or feasible to use in all systems.

Integration with other systems: Vehicle detection and classification systems often need to be integrated with other systems, such as traffic management systems or surveillance systems. This requires careful coordination and may require the use of standardized interfaces or protocols to ensure interoperability.

Privacy concerns: Vehicle detection and classification systems may collect and process sensitive information, such as license plate numbers or vehicle movements. It is important to consider privacy concerns and ensure that appropriate safeguards are in place to protect this data.

Robustness to changing conditions: Vehicle detection and classification algorithms need to be able to perform reliably under a wide range of conditions, including changes in lighting, weather, and environmental conditions. This requires careful selection and tuning of algorithms and may involve the use of machine learning techniques to adapt to changing conditions.

REFERENCES: -

- [1] A. Blake, d Isard; Contour Tracking by Stochastic Propagation of Conditional Density. In P m. European Conference on Computer Vision 342-356 1996.
- [2] A.H.S. Lai, G.S.K. Fung, N.H.C. Yung,” Vehicle Type Classification from Visual-Based Dimension Estimation”, Intelligent Transportation Systems, 2001. Proceedings. 2001 IEEE, Oakland, CA, USA, pp. 201- 206, 25-29 Aug. 2001.
- [3] Ali Tourani Asadollah Shahbahrami, Vehicle Counting Method Based on Digital Image Processing Algorithms 2015 2nd International Conference on Pattern Recognition and Image Analysis (IPRIA 2015) March 11-12, 2015.
- [4] Amit Kumar, Mansour H. Assaf¹, Sunil R. Das, Satyendra N. Biswas, Emil M. Petriu, and Voicu Groza, Image Processing Based System for Classification of Vehicles for Parking Purposes, IEEE 2015 IEEE International Instrumentation and Measurement Technology Conference (I2MTC).
- [5] Bouvie, C.; Scharcanski, J.; Barcellos, P.; Lopes Escouto, F.,” Tracking and counting vehicles in traffic video sequences using particle filtering,” Instrumentation and Measurement Technology Conference (I2MTC), 2013 IEEE International, vol., no., pp.812,815, 6-9 May 2013.
- [6] Bouvie, Christiano; Scharcanski, Jacob; Barcellos, Pablo; Escout, Tracking and Counting Vehicles in Traffic Video Sequences Using Particle Filtering, IEEE 2013 IEEE International Instrumentation and Measurement Technology Conference (I2MTC).
- [7] J. Zhou, D. Gao, D. Zhang, Moving Vehicle Detection for Automatic Traffic Monitoring, IEEE Transactions on Vehicular Technology, Vol. 56, NO. 1,2007.
- [8] Jacinto C. Nascimento Arnaldo J. Abrantes Jorge S. Marques t, AN ALGORITHM FOR CENTROID-BASED TRACKING OF MOVING OBJECTS, 0 1999 IEEE

- [9] Jyotsna Tripathi, Kavita Chaudhary, Akanksha Joshi, and Prof. J B Jawale, Automatic Vehicle Counting and Classification, International Journal of Innovative and Emerging Research in Engineering Volume 2, Issue 4, 2015.
- [10] Mithun, N.C.; Rashid, N.U.; Rahman, S.M.M.,” Detection and Classification of Vehicles from Video Using Multiple Time-Spatial Images,” Intelligent Transportation Systems, IEEE Transactions on, vol.13, no.3, pp.1215, 1225, Sept. 2012.
- [11] Prem Kumar Bhaskar Suet-Peng Yong, Image Processing Based Vehicle Detection and Tracking Method, 2014 IEEE.
- [12] Pandu ranga, H.T; Ravi kiran, M; Raja shekar, S; Naveen kumar, Vehicle Detection and Classification based on Morphological Technique, IEEE 2010 International Conference on Signal, and Image Processing (ICSIP).
- [13] R. Gonzalez, R. E. Woods, Digital Image Processing, 2nd Edition, Prentice-Hall, 2002
- [14] S. Kul, S. Eken, and A. Sayar Distributed and collaborative real-time vehicle detection and classification over the video streams, Int. J. Adv. Robot. Syst., vol. 14, no. 4, p. 172988141772078, Jul. 2017.
- [15] Zhao, R.; Wang, X.,” Counting Vehicles from Semantic Regions,” Intelligent Transportation Systems, IEEE Transactions on, vol.14, no.2, pp.1016, 1022, June 2013.

APPENDIX:

A. SOURCE CODE:

```
from google.colab import drive
drive.mount('/content/drive/')
```

```
!git clone https://github.com/kalvaakhil/akhil-yolov8.git
```

```
%cd /content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-main
```

```
!pip install -e '.[dev]'
```

```
%cd /content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-main/ultralytics/yolo/v8/detect
```

```
!pip install roboflow
```

```
from roboflow import Roboflow
rf = Roboflow(api_key="ncEaFKHsT7P97J36cNmb")
project = rf.workspace("yolo-trials").project("vehicle-detection-eckrb")
dataset = project.version(3).download("yolov8")
```

```
import torch
class Loss:

    def __init__(self, model): # model must be de-paralleled

        device = next(model.parameters()).device # get model device
        h = model.args # hyperparameters

        m = model.model[-1] # Detect() module
        self.bce = nn.BCEWithLogitsLoss(reduction='none')
        self.hyp = h
        self.stride = m.stride # model strides
        self.nc = m.nc # number of classes
        self.no = m.no
        self.reg_max = m.reg_max
        self.device = device

        self.use_dfl = m.reg_max > 1
```

```

        self.assgner = TaskAlignedAssgner(topk=10, num_classes=self.nc,
alpha=0.5, beta=6.0)
        self.bbox_loss = BboxLoss(m.reg_max - 1,
use_dfl=self.use_dfl).to(device)
        self.proj = torch.arange(m.reg_max, dtype=torch.float,
device=device)

    def preprocess(self, targets, batch_size, scale_tensor):
        if targets.shape[0] == 0:
            out = torch.zeros(batch_size, 0, 5, device=self.device)
        else:
            i = targets[:, 0] # image index
            _, counts = i.unique(return_counts=True)
            out = torch.zeros(batch_size, counts.max(), 5,
device=self.device)
            for j in range(batch_size):
                matches = i == j
                n = matches.sum()
                if n:
                    out[j, :n] = targets[matches, 1:]
            out[..., 1:5] = xywh2xyxy(out[..., 1:5].mul_(scale_tensor))
        return out

    def bbox_decode(self, anchor_points, pred_dist):
        if self.use_dfl:
            b, a, c = pred_dist.shape # batch, anchors, channels
            pred_dist = pred_dist.view(b, a, 4, c //
4).softmax(3).matmul(self.proj.type(pred_dist.dtype))
            # pred_dist = pred_dist.view(b, a, c // 4,
4).transpose(2,3).softmax(3).matmul(self.proj.type(pred_dist.dtype))
            # pred_dist = (pred_dist.view(b, a, c // 4, 4).softmax(2) *
self.proj.type(pred_dist.dtype).view(1, 1, -1, 1)).sum(2)
            return dist2bbox(pred_dist, anchor_points, xywh=False)

    def __call__(self, preds, batch):
        loss = torch.zeros(3, device=self.device) # box, cls, dfl
        feats = preds[1] if isinstance(preds, tuple) else preds
        pred_distri, pred_scores = torch.cat([xi.view(feats[0].shape[0],
self.no, -1) for xi in feats], 2).split(
            (self.reg_max * 4, self.nc), 1)

        pred_scores = pred_scores.permute(0, 2, 1).contiguous()
        pred_distri = pred_distri.permute(0, 2, 1).contiguous()

        dtype = pred_scores.dtype
        batch_size = pred_scores.shape[0]

```

```

        imgsz = torch.tensor(feats[0].shape[2:], device=self.device,
dtype=dtype) * self.stride[0] # image size (h,w)
        anchor_points, stride_tensor = make_anchors(feats, self.stride, 0.5)

        # targets

```

```

import numpy as np
import torch
import torch.nn as nn

# BaseTrainer python usage
class DetectionTrainer:

    def get_dataloader(self, dataset_path, batch_size, mode='train',
rank=0):
        # TODO: manage splits differently
        # calculate stride - check if model is initialized
        gs = max(int(de_parallel(self.model).stride.max() if self.model else
0), 32)
        return create_dataloader(path=dataset_path,
                                imgsz=self.args.imgsz,
                                batch_size=batch_size,
                                stride=gs,
                                hyp=vars(self.args),
                                augment=mode == 'train',
                                cache=self.args.cache,
                                pad=0 if mode == 'train' else 0.5,
                                rect=self.args.rect or mode == 'val',
                                rank=rank,
                                workers=self.args.workers,
                                close_mosaic=self.args.close_mosaic != 0,
                                prefix=colorstr(f'{mode}: '),
                                shuffle=mode == 'train',
                                seed=self.args.seed)[0] if
self.args.v5loader else \
            build_dataloader(self.args, batch_size, img_path=dataset_path,
stride=gs, rank=rank, mode=mode,
                                rect=mode == 'val',
names=self.data['names'])[0]

    def preprocess_batch(self, batch):
        batch['img'] = batch['img'].to(self.device,
non_blocking=True).float() / 255
        return batch

```

```

def set_model_attributes(self):
    self.model.nc = self.data['nc'] # attach number of classes to model
    self.model.names = self.data['names'] # attach class names to model
    self.model.args = self.args # attach hyperparameters to model
def get_model(self, cfg=None, weights=None, verbose=True):
    if fg_mask.sum():
        loss[0], loss[2] = self.bbox_loss(pred_distri, pred_bboxes,
                                         anchor_points, target_bboxes, target_scores,
                                         target_scores_sum, fg_mask)

        loss[0] *= self.hyp.box # box gain
        loss[1] *= self.hyp.cls # cls gain
        loss[2] *= self.hyp.dfl # dfl gain

    return loss.sum() * batch_size, loss.detach() # loss(box, cls, dfl)

```

```

# Ultralytics YOLO , GPL-3.0 license

import torch
import torch.nn as nn
import torch.nn.functional as F

# from .metrics import bbox_iou
# from .tal import bbox2dist
class VarifocalLoss(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, pred_score, gt_score, label, alpha=0.75, gamma=2.0):
        weight = alpha * pred_score.sigmoid().pow(gamma) * (1 - label) +
        gt_score * label
        with torch.cuda.amp.autocast(enabled=False):
            loss = (F.binary_cross_entropy_with_logits(pred_score.float(),
            gt_score.float(), reduction="none") *
                    weight).sum()
        return loss
class BboxLoss(nn.Module):

    def __init__(self, reg_max, use_dfl=False):
        super().__init__()
        self.reg_max = reg_max
        self.use_dfl = use_dfl

    def forward(self, pred_dist, pred_bboxes, anchor_points, target_bboxes,
target_scores, target_scores_sum, fg_mask):

```

```

        # IoU loss
        weight = torch.masked_select(target_scores.sum(-1),
fg_mask).unsqueeze(-1)
        iou = bbox_iou(pred_bboxes[fg_mask], target_bboxes[fg_mask],
xywh=False, CIoU=True)
        loss_iou = ((1.0 - iou) * weight).sum() / target_scores_sum

        # DFL loss
        if self.use_dfl:
            target_ltrb = bbox2dist(anchor_points, target_bboxes,
self.reg_max)
            loss_dfl = self._df_loss(pred_dist[fg_mask].view(-1,
self.reg_max + 1), target_ltrb[fg_mask]) * weight
            loss_dfl = loss_dfl.sum() / target_scores_sum
        else:
            loss_dfl = torch.tensor(0.0).to(pred_dist.device)

        return loss_iou, loss_dfl

    @staticmethod
    def _df_loss(pred_dist, target):
        # Return sum of left and right DFL losses
        tl = target.long() # target left
        tr = tl + 1 # target right
        wl = tr - target # weight left
        wr = 1 - wl # weight right
        return (F.cross_entropy(pred_dist, tl.view(-1),
reduction="none").view(tl.shape) * wl +
                F.cross_entropy(pred_dist, tr.view(-1),
reduction="none").view(tl.shape) * wr).mean(-1, keepdim=True)

# Ultralytics YOLO 🚀, GPL-3.0 license
"""
Model validation metrics
"""
import math
import warnings
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn

# from ultralytics.yolo.utils import TryExcept

```

```

# boxes
def box_area(box):
    # box = xyxy(4,n)
    return (box[2] - box[0]) * (box[3] - box[1])

def bbox_ioa(box1, box2, eps=1e-7):
    """Returns the intersection over box2 area given box1, box2. Boxes are
    x1y1x2y2
    box1:      np.array of shape(nx4)
    box2:      np.array of shape(mx4)
    returns:   np.array of shape(nxm)
    """

    # Get the coordinates of bounding boxes
    b1_x1, b1_y1, b1_x2, b1_y2 = box1.T
    b2_x1, b2_y1, b2_x2, b2_y2 = box2.T

    # Intersection area
    inter_area = (np.minimum(b1_x2[:, None], b2_x2) - np.maximum(b1_x1[:,
None], b2_x1)).clip(0) * \
        (np.minimum(b1_y2[:, None], b2_y2) - np.maximum(b1_y1[:,
None], b2_y1)).clip(0)

    # box2 area
    box2_area = (b2_x2 - b2_x1) * (b2_y2 - b2_y1) + eps

    # Intersection over box2 area
    return inter_area / box2_area

def box_iou(box1, box2, eps=1e-7):
    # https://github.com/pytorch/vision/blob/master/torchvision/ops/boxes.py
    """
    Return intersection-over-union (Jaccard index) of boxes.
    Both sets of boxes are expected to be in (x1, y1, x2, y2) format.
    Arguments:
        box1 (Tensor[N, 4])
        box2 (Tensor[M, 4])
    Returns:
        iou (Tensor[N, M]): the NxM matrix containing the pairwise
        IoU values for every element in boxes1 and boxes2
    """

    # inter(N,M) = (rb(N,M,2) - lt(N,M,2)).clamp(0).prod(2)
    (a1, a2), (b1, b2) = box1.unsqueeze(1).chunk(2, 2),
box2.unsqueeze(0).chunk(2, 2)

```

```

inter = (torch.min(a2, b2) - torch.max(a1, b1)).clamp(0).prod(2)

# IoU = inter / (area1 + area2 - inter)
return inter / ((a2 - a1).prod(2) + (b2 - b1).prod(2) - inter + eps)

def bbox_iou(box1, box2, xywh=True, GIoU=False, DIOU=False, CIOU=False,
eps=1e-7):
    # Returns Intersection over Union (IoU) of box1(1,4) to box2(n,4)

    # Get the coordinates of bounding boxes
    if xywh: # transform from xywh to xyxy
        (x1, y1, w1, h1), (x2, y2, w2, h2) = box1.chunk(4, -1),
box2.chunk(4, -1)
        w1_, h1_, w2_, h2_ = w1 / 2, h1 / 2, w2 / 2, h2 / 2
        b1_x1, b1_x2, b1_y1, b1_y2 = x1 - w1_, x1 + w1_, y1 - h1_, y1 + h1_
        b2_x1, b2_x2, b2_y1, b2_y2 = x2 - w2_, x2 + w2_, y2 - h2_, y2 + h2_
    else: # x1, y1, x2, y2 = box1
        b1_x1, b1_y1, b1_x2, b1_y2 = box1.chunk(4, -1)
        b2_x1, b2_y1, b2_x2, b2_y2 = box2.chunk(4, -1)
        w1, h1 = b1_x2 - b1_x1, b1_y2 - b1_y1 + eps
        w2, h2 = b2_x2 - b2_x1, b2_y2 - b2_y1 + eps

    # Intersection area
    inter = (b1_x2.minimum(b2_x2) - b1_x1.maximum(b2_x1)).clamp(0) * \
        (b1_y2.minimum(b2_y2) - b1_y1.maximum(b2_y1)).clamp(0)

    # Union Area
    union = w1 * h1 + w2 * h2 - inter + eps

    # IoU
    iou = inter / union
    if CIOU or DIOU or GIoU:
        cw = b1_x2.maximum(b2_x2) - b1_x1.minimum(b2_x1) # convex (smallest
enclosing box) width
        ch = b1_y2.maximum(b2_y2) - b1_y1.minimum(b2_y1) # convex height
        if CIOU or DIOU: # Distance or Complete IoU
https://arxiv.org/abs/1911.08287v1
            c2 = cw ** 2 + ch ** 2 + eps # convex diagonal squared
            rho2 = ((b2_x1 + b2_x2 - b1_x1 - b1_x2) ** 2 + (b2_y1 + b2_y2 -
b1_y1 - b1_y2) ** 2) / 4 # center dist ** 2
            if CIOU: # https://github.com/Zzh-tju/DIOU-SSD-
pytorch/blob/master/utils/box/box\_utils.py#L47
                v = (4 / math.pi ** 2) * (torch.atan(w2 / h2) -
torch.atan(w1 / h1)).pow(2)
                with torch.no_grad():
                    alpha = v / (v - iou + (1 + eps))

```

```

        return iou - (rho2 / c2 + v * alpha) # CIoU
    return iou - rho2 / c2 # DIoU
    c_area = cw * ch + eps # convex area
    return iou - (c_area - union) / c_area # GIoU
https://arxiv.org/pdf/1902.09630.pdf
    return iou # IoU

def mask_iou(mask1, mask2, eps=1e-7):
    """
    mask1: [N, n] m1 means number of predicted objects
    mask2: [M, n] m2 means number of gt objects
    Note: n means image_w x image_h
    return: masks iou, [N, M]
class ConfusionMatrix:
    # Updated version of
https://github.com/kaanakan/object_detection_confusion_matrix
    def __init__(self, nc, conf=0.25, iou_thres=0.45):
        self.matrix = np.zeros((nc + 1, nc + 1))
        self.nc = nc # number of classes
        self.conf = conf
        self.iou_thres = iou_thres

    def process_batch(self, detections, labels):
        """
        Return intersection-over-union (Jaccard index) of boxes.
        Both sets of boxes are expected to be in (x1, y1, x2, y2) format.
        Arguments:
            detections (Array[N, 6]), x1, y1, x2, y2, conf, class
            labels (Array[M, 5]), class, x1, y1, x2, y2
        Returns:
            None, updates confusion matrix accordingly
        """
        if detections is None:
            target_cls: True object classes (nparray).
            plot: Plot precision-recall curve at mAP@0.5
            save_dir: Plot save directory
        # Returns
        f1 = 2 * p * r / (p + r + eps)
        names = [v for k, v in names.items() if k in unique_classes] # list:
only classes that have data
        names = dict(enumerate(names)) # to dict
        if plot:
            plot_pr_curve(px, py, ap, save_dir / f'{prefix}PR_curve.png', names)
            plot_mc_curve(px, f1, save_dir / f'{prefix}F1_curve.png', names,
ylabel='F1')

```

```

        plot_mc_curve(px, p, save_dir / f'{prefix}P_curve.png', names,
ylabel='Precision')
        plot_mc_curve(px, r, save_dir / f'{prefix}R_curve.png', names,
ylabel='Recall')

i = smooth(f1.mean(0), 0.1).argmax() # max F1 index
p, r, f1 = p[:, i], r[:, i], f1[:, i]
tp = (r * nt).round() # true positives
fp = (tp / (p + eps) - tp).round() # false positives
return tp, fp, p, r, f1, ap, unique_classes.astype(int)
class Metric:

    def __init__(self) -> None:
        self.p = [] # (nc, )
        self.r = [] # (nc, )
        self.f1 = [] # (nc, )
        self.all_ap = [] # (nc, 10)
        self.ap_class_index = [] # (nc, )

    @property
    def ap50(self):
        """AP@0.5 of all classes.
        Return:
            (nc, ) or [].
        """
        return self.all_ap[:, 0] if len(self.all_ap) else []

    @property
    def ap(self):
        """AP@0.5:0.95
        Return:
            (nc, ) or [].
        """
        return self.all_ap.mean(1) if len(self.all_ap) else []

    @property
    def mp(self):
        """mean precision of all classes.
        Return:
            float.
        """
        return self.p.mean() if len(self.p) else 0.0

    @property
    def mr(self):
        """mean recall of all classes.
        Return:

```

```

        float.
    """
    return self.r.mean() if len(self.r) else 0.0

@property
def map50(self):
    """Mean AP@0.5 of all classes.
    Return:
        float.
    """
    return self.all_ap[:, 0].mean() if len(self.all_ap) else 0.0

@property
def map(self):
    """Mean AP@0.5:0.95 of all classes.
    Return:
        float.
    """
    return self.all_ap.mean() if len(self.all_ap) else 0.0

def mean_results(self):
    """Mean of results, return mp, mr, map50, map"""
    return [self.mp, self.mr, self.map50, self.map]

def class_result(self, i):
    """class-aware result, return p[i], r[i], ap50[i], ap[i]"""
    return self.p[i], self.r[i], self.ap50[i], self.ap[i]

def get_maps(self, nc):
    maps = np.zeros(nc) + self.map
    for i, c in enumerate(self.ap_class_index):
        maps[c] = self.ap[i]
    return maps

def fitness(self):
    # Model fitness as a weighted combination of metrics
    w = [0.0, 0.0, 0.1, 0.9] # weights for [P, R, mAP@0.5,
mAP@0.5:0.95]
    return (np.array(self.mean_results()) * w).sum()

def update(self, results):
    """
    Args:
        results: tuple(p, r, ap, f1, ap_class)
    """
    self.p, self.r, self.f1, self.all_ap, self.ap_class_index = results

```

```

class DetMetrics:

    def __init__(self, save_dir=Path("."), plot=False, names=()) -> None:
        self.save_dir = save_dir
        self.plot = plot
        self.names = names
        self.metric = Metric()

    def process(self, tp, conf, pred_cls, target_cls):
        results = ap_per_class(tp, conf, pred_cls, target_cls,
                                plot=self.plot, save_dir=self.save_dir,
                                names=self.names)[2:]
        self.metric.update(results)

    @property
    def keys(self):
        return ["metrics/precision(B)", "metrics/recall(B)",
                "metrics/mAP50(B)", "metrics/mAP50-95(B)"]

    def mean_results(self):
        return self.metric.mean_results()

HOME = "/content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-
main/ultralytics/yolo/v8/detect"
%cd {HOME}
%cd {dataset.location}
%cd {HOME}

!python train.py model=yolov8l.pt data={dataset.location}/data.yaml batch=3
epochs=80 imgsz=640
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-
main/runs/detect/train3/confusion_matrix.png")
imshow(testim)
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-
main/runs/detect/train3/F1_curve.png")
imshow(testim)

```

```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-
main/runs/detect/train3/PR_curve.png")
imshow(testim)
```

```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-
main/runs/detect/train3/P_curve.png")
imshow(testim)
```

```
import cv2
from google.colab.patches import cv2_imshow
!git clone https://github.com/ifzhang/ByteTrack.git
%cd {HOME}/ByteTrack
!sed -i 's/onnx==1.8.1/onnx==1.9.0/g' requirements.txt

!pip3 install -q -r requirements.txt
!python3 setup.py -q develop
!pip install -q cython_bbox
!pip install -q onemetric

from IPython import display
display.clear_output()

import sys
sys.path.append(f"{HOME}/ByteTrack")

import yolox
print("yolox. version :", yolox. version )
```

```
from yolox.tracker.byte_tracker import BYTETracker, STrack
from onemetric.cv.utils.iou import box_iou_batch
from dataclasses import dataclass
```

```

@dataclass(frozen=True)
class BYTETrackerArgs:
    track_thresh: float = 0.25
    track_buffer: int = 30
    match_thresh: float = 0.8
    aspect_ratio_thresh: float = 3.0
    min_box_area: float = 1.0
    mot20: bool = False

```

```

!pip install supervision==0.1.0

```

```

from IPython import display
display.clear_output()

```

```

import supervision
print("supervision.__version__:", supervision.__version__)
from supervision.draw.color import ColorPalette
from supervision.geometry.dataclasses import Point
from supervision.video.dataclasses import VideoInfo
from supervision.video.source import get_video_frames_generator
from supervision.video.sink import VideoSink
from supervision.notebook.utils import show_frame_in_notebook
from supervision.tools.detections import Detections, BoxAnnotator
from supervision.tools.line_counter import LineCounter, LineCounterAnnotator

import numpy as np
import scipy.linalg

chi2inv95 = {
    1: 3.8415,
    2: 5.9915,
    3: 7.8147,
    4: 9.4877,
    5: 11.070,
    6: 12.592,
    7: 14.067,
    8: 15.507,
    9: 16.919}

class KalmanFilter(object):
    def __init__(self):
        ndim, dt = 4, 1.
        self._motion_mat = np.eye(2 * ndim, 2 * ndim)
        for i in range(ndim):

```

```

        self._motion_mat[i, ndim + i] = dt
    self._update_mat = np.eye(ndim, 2 * ndim)
    self._std_weight_position = 1. / 20
    self._std_weight_velocity = 1. / 160
def initiate(self, measurement):
    mean_pos = measurement
    mean_vel = np.zeros_like(mean_pos)
    mean = np.r_[mean_pos, mean_vel]
    std = [
        2 * self._std_weight_position * measurement[3],
        2 * self._std_weight_position * measurement[3],
        1e-2,
        2 * self._std_weight_position * measurement[3],
        10 * self._std_weight_velocity * measurement[3],
        10 * self._std_weight_velocity * measurement[3],
        1e-5,
        10 * self._std_weight_velocity * measurement[3]]
    covariance = np.diag(np.square(std))
    return mean, covariance
def predict(self, mean, covariance):
    std_pos = [
        self._std_weight_position * mean[3],
        self._std_weight_position * mean[3],
        1e-2,
        self._std_weight_position * mean[3]]
    std_vel = [
        self._std_weight_velocity * mean[3],
        self._std_weight_velocity * mean[3],
        1e-5,
        self._std_weight_velocity * mean[3]]
    motion_cov = np.diag(np.square(np.r_[std_pos, std_vel]))

    mean = np.dot(self._motion_mat, mean)
    covariance = np.linalg.multi_dot((
        self._motion_mat, covariance, self._motion_mat.T)) + motion_cov

    return mean, covariance
def project(self, mean, covariance):
    std = [
        self._std_weight_position * mean[3],
        self._std_weight_position * mean[3],
        1e-1,
        self._std_weight_position * mean[3]]
    innovation_cov = np.diag(np.square(std))
    mean = np.dot(self._update_mat, mean)
    covariance = np.linalg.multi_dot((
        self._update_mat, covariance, self._update_mat.T))

```

```

        return mean, covariance + innovation_cov
    def update(self, mean, covariance, measurement):
        projected_mean, projected_cov = self.project(mean, covariance
    )

        chol_factor, lower = scipy.linalg.cho_factor(
            projected_cov, lower=True, check_finite=False)
        kalman_gain = scipy.linalg.cho_solve(
            (chol_factor, lower), np.dot(covariance, self._update_mat
    .T).T,
            check_finite=False).T
        innovation = measurement - projected_mean
        new_mean = mean + np.dot(innovation, kalman_gain.T)
        new_covariance = covariance - np.linalg.multi_dot((
            kalman_gain, projected_cov, kalman_gain.T))
        return new_mean, new_covariance
    def gating_distance(self, mean, covariance, measurements,
                        only_position=False):
        mean, covariance = self.project(mean, covariance)
        if only_position:
            mean, covariance = mean[:2], covariance[:2, :2]
            measurements = measurements[:, :2]
        cholesky_factor = np.linalg.cholesky(covariance)
        d = measurements - mean
        z = scipy.linalg.solve_triangular(
            cholesky_factor, d.T, lower=True, check_finite=False,
            overwrite_b=True)
        squared_maha = np.sum(z * z, axis=0)
        return squared_maha
import cv2
import numpy as np
import lap
from scipy.spatial.distance import cdist
from cython_bbox import bbox_overlaps as bbox_ious
from yolox.motdt_tracker import kalman_filter
def _indices_to_matches(cost_matrix, indices, thresh):

import cv2
import numpy as np
def generate_2d_mask(polygon: np.ndarray, resolution_wh: Tuple[int, i
nt]) -> np.ndarray:
    width, height = resolution_wh
    mask = np.zeros((height, width), dtype=np.uint8)
    cv2.fillPoly(mask, [polygon], color=1)
    return mask

```

```

def box_iou_batch(boxes_true: np.ndarray, boxes_detection: np.ndarray
) -> np.ndarray:
    def box_area(box):
        return (box[2] - box[0]) * (box[3] - box[1])

    area_true = box_area(boxes_true.T)
    area_detection = box_area(boxes_detection.T)

    top_left = np.maximum(boxes_true[:, None, :2], boxes_detection[:,
:2])
    bottom_right = np.minimum(boxes_true[:, None, 2:], boxes_detectio
n[:, 2:])

    area_inter = np.prod(np.clip(bottom_right - top_left, a_min=0, a_
max=None), 2)
    return area_inter / (area_true[:, None] + area_detection - area_i
nter)
def non_max_suppression(
    predictions: np.ndarray, iou_threshold: float = 0.5
) -> np.ndarray:
    assert 0 <= iou_threshold <= 1, (
        f"Value of `iou_threshold` must be in the closed range from 0
to 1, "
        f"{iou_threshold} given."
    )
    rows, columns = predictions.shape
    if columns == 5:
        predictions = np.c_[predictions, np.zeros(rows)]
    sort_index = np.flip(predictions[:, 4].argsort())
    predictions = predictions[sort_index]

```

B. SCREENSHOTS:

```

from google.colab import drive
drive.mount('/content/drive/')

Mounted at /content/drive/

[ ] !git clone https://github.com/kalvaakhil/akhil-yolov8.git

Cloning into 'akhil-yolov8'...
remote: Enumerating objects: 184, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (36/36), done.
remote: Total 184 (delta 23), reused 51 (delta 19), pack-reused 127
Receiving objects: 100% (184/184), 139.18 MiB | 47.57 MiB/s, done.
Resolving deltas: 100% (48/48), done.

[ ] %cd /content/akhil-yolov8/YOLOv8 Segmentation DeepSORT Object Tracking-main

/content/akhil-yolov8/YOLOv8 Segmentation DeepSORT Object Tracking-main

[ ] !pip install -e \".[dev]\"

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Obtaining file:///content/akhil-yolov8/YOLOv8 Segmentation DeepSORT Object Tracking-main
Preparing metadata (setup.py) ... done
Collecting hydra-core==1.2.0
  Downloading hydra_core-1.3.2-py3-none-any.whl (154 kB)
    Requirement already satisfied: matplotlib==3.2.2 in /usr/local/lib/python3.9/dist-packages (from ultralytics==8.0.3) (3.7.1)
    Requirement already satisfied: numpy==1.18.5 in /usr/local/lib/python3.9/dist-packages (from ultralytics==8.0.3) (1.22.4)

```

```
[ ] %cd /content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-main/ultralytics/yolo/v8/detect
/content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-main/ultralytics/yolo/v8/detect

!pip install roboflow

from roboflow import Roboflow
rf = Roboflow(api_key="ncEaFKHsT7P97336cNmb")
project = rf.workspace("yolo-trials").project("vehicle-detection-eckrb")
dataset = project.version(3).download("yolov8")

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting roboflow
  Downloading roboflow-1.0.1-py3-none-any.whl (55 kB)
    Downloading roboflow-1.0.1-py3-none-any.whl (55 kB)
Collecting idna==2.10
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
    Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
Collecting cyclizer==0.10.0
  Downloading cyclizer-0.10.0-py2.py3-none-any.whl (6.5 kB)
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packages (from roboflow) (2.27.1)
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.9/dist-packages (from roboflow) (8.4.0)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.9/dist-packages (from roboflow) (6.0)
Requirement already satisfied: chardet==4.0.0 in /usr/local/lib/python3.9/dist-packages (from roboflow) (4.0.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/dist-packages (from roboflow) (2.8.2)
Collecting python-dotenv
  Downloading python_dotenv-1.0.0-py3-none-any.whl (19 kB)
Collecting wget
  Downloading wget-3.2.zip (10 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.9/dist-packages (from roboflow) (4.65.0)
Collecting pyparsing==2.4.7
  Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
    Downloading pyparsing-2.4.7-py2.py3-none-any.whl (67 kB)
Requirement already satisfied: opencv-python>=4.1.2 in /usr/local/lib/python3.9/dist-packages (from roboflow) (4.7.0.72)
Requirement already satisfied: urllib3>=1.20.0 in /usr/local/lib/python3.9/dist-packages (from roboflow) (1.26.15)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.9/dist-packages (from roboflow) (1.4.4)
Requirement already satisfied: six in /usr/local/lib/python3.9/dist-packages (from roboflow) (1.16.0)
Requirement already satisfied: certifi==2022.12.7 in /usr/local/lib/python3.9/dist-packages (from roboflow) (2022.12.7)
Collecting requests-toolbelt
```

```
import torch
class Loss:
    def __init__(self, model): # model must be de-parallelized
        device = next(model.parameters()).device # get model device
        h = model.args # hyperparameters

        m = model.model[-1] # Detect() module
        self.bce = nn.BCEWithLogitsLoss(reduction='none')
        self.hyp = h
        self.stride = m.stride # model strides
        self.nc = m.nc # number of classes
        self.no = m.no
        self.reg_max = m.reg_max
        self.device = device

        self.use_dfl = m.reg_max > 1
        self.assigner = TaskAlignedAssigner(topk=10, num_classes=self.nc, alpha=0.5, beta=6.0)
        self.bbox_loss = BboxLoss(m.reg_max - 1, use_dfl=self.use_dfl).to(device)
        self.proj = torch.arange(m.reg_max, dtype=torch.float, device=device)

    def preprocess(self, targets, batch_size, scale_tensor):
        if targets.shape[0] == 0:
            out = torch.zeros(batch_size, 0, 5, device=self.device)
        else:
            i = targets[:, 0] # image index
            counts = i.unique(return_counts=True)
            out = torch.zeros(batch_size, counts.max(), 5, device=self.device)
            for j in range(batch_size):
                matches = i == j
                n = matches.sum()
                if n:
                    out[j, :n] = targets[matches, 1:]
            out[... ,1:5] = xywh2xyxy(out[... ,1:5].mul(scale_tensor))
        return out

    def bbox_decode(self, anchor_points, pred_dist):
        if self.use_dfl:
            b, a, c = pred_dist.shape # batch, anchors, channels
            pred_dist = pred_dist.view(b, a, 4, c // 4).softmax(3).matmul(self.proj.type(pred_dist.dtype))
            # pred_dist = pred_dist.view(b, a, c // 4, 4).transpose(2,3).softmax(3).matmul(self.proj.type(pred_dist.dtype))
            # pred_dist = (pred_dist.view(b, a, c // 4, 4).softmax(2) * self.proj.type(pred_dist.dtype).view(1, 1, -1, 1)).sum(2)
        return dist2bbox(pred_dist, anchor_points, xywh=False)
```

```
[4] %cd /content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-main/ultralytics/yolo/v8/detect
/content/akhil-yolov8/YOLOv8_Segmentation_DeepSORT_Object_Tracking-main/ultralytics/yolo/v8/detect

from copy import copy

import numpy as np
import torch
import torch.nn as nn

# BaseTrainer python usage
class DetectionTrainer:
    def get_dataloader(self, dataset_path, batch_size, mode='train', rank=0):
        # TODO: Manage splits differently
        # calculate stride - check if model is initialized
        gs = max(int(de_parallel(self.model).stride.max() if self.model else 0), 32)
        return create_dataloader(path=dataset_path,
                                imgs=self.args.imgsz,
                                batch_size=batch_size,
                                stride=gs,
                                hyp=vars(self.args),
                                augment=mode == 'train',
                                cache=self.args.cache,
                                pad=0 if mode == 'train' else 0.5,
                                rect=self.args.rect or mode == 'val',
                                rank=rank,
                                workers=self.args.workers,
                                close_mosaic=self.args.close_mosaic != 0,
                                prefixcolorstr=('mode: '),
                                shuffle=mode == 'train',
                                seed=self.args.seed)[0] if self.args.v5loader else \
            build_dataloader(self.args, batch_size, img_path=dataset_path, stride=gs, rank=rank, mode=mode,
                             rect=mode == 'val', names=self.data['names'])[0]

    def preprocess_batch(self, batch):
        batch['img'] = batch['img'].to(self.device, non_blocking=True).float() / 255
        return batch
```

```

def preprocess_batch(self, batch):
    batch['img'] = batch['img'] / (self.device, non_blocking=True).float() / 255
    return batch

def set_model_attributes(self):
    self.model.nc = self.data['nc'] # attach number of classes to model
    self.model.names = self.data['names'] # attach class names to model
    self.model.args = self.args # attach hyperparameters to model
    def get_model(self, cfg=None, weights=None, verbose=True):
        model = DetectionModel(cfg, ch3, nc=self.data['nc'], verbose=verbose and RANK == -1)
        if weights:
            model.load(weights)
    return model

def get_validator(self):
    self.loss_names = 'box_loss', 'cls_loss', 'dfl_loss'
    return v0.detect.DetectionValidator(self.loss_loader, save_dir=self.save_dir, args=copy(self.args))

def criterion(self, preds, batch):
    if not hasattr(self, 'compute_loss'):
        self.compute_loss = loss(Dc_parallel(self.model))
    return self.compute_loss(preds, batch)
def label_loss_items(self, loss_items=None, prefix='train'):
    keys = ['f'*(preis)(x) for x in self.loss_names]
    if loss_items is not None:
        loss_items = [round(float(x), 5) for x in loss_items]
    return dict(zip(keys, loss_items))
    else:
        return keys

def progress_string(self):
    return '%10s' % '%10s' *
        (4 + len(self.loss_names)) % ('Epoch', 'GPU mem', '%self.loss_names', 'Instances', 'Size')
def plot_training_samples(self, batch, ni):
    plot_images(images=batch['img'],
                batch_idx=batch['batch_idx'],
                cls=batch['cls'].squeeze(-1),
                bboxes=batch['bboxes'],
                paths=batch['im_file'],
                frames=self.save_dir / f'train_batch{ni}.jpg')

def plot_metrics(self):
    plot_results(files=self.csv)

def plot_training_labels(self):
    boxes = np.concatenate([b['bboxes'] for lb in self.train_loader.dataset.labels], 0)
    cls = np.concatenate([l['cids'] for lb in self.train_loader.dataset.labels], 0)
    plot_labels(boxes=boxes, cls=cls, names=self.data['names'], save_dir=self.save_dir)

```

```

class Loss:
    def __init__(self, model): # model must be de-parallelized
        device = next(model.parameters()).device # get model device
        h = model.args # hyperparameters

        m = model.model[-1] # Detect() module
        self.bce = nn.BCEWithLogitsLoss(reduction='none')
        self.hyp = h
        self.stride = m.stride # model strides
        self.nc = m.nc # number of classes
        self.no = m.no
        self.reg_max = m.reg_max
        self.device = device

        self.use_dfl = m.reg_max > 1
        roll_out_thr = h.min_memory if h.min_memory > 1 else 64 if h.min_memory else 0 # 64 is default

        self.assigner = TaskAlignedAssigner(topk=10,
            num_classes=self.nc,
            alpha=0.5,
            beta=6.0,
            roll_out_thr=roll_out_thr)

        self.bbox_loss = BboxLoss(m.reg_max - 1, use_dfl=self.use_dfl).to(device)
        self.proj = torch.arange(m.reg_max, dtype=torch.float, device=device)

    def preprocess(self, targets, batch_size, scale_tensor):
        if targets.shape[0] == 0:
            out = torch.zeros(batch_size, 0, 5, device=self.device)
        else:
            i = targets[:, 0] # image index
            _, counts = i.unique(return_counts=True)
            out = torch.zeros(batch_size, counts.max(), 5, device=self.device)
            for j in range(batch_size):
                matches = i == j
                n = matches.sum()
                if n:
                    out[j, :n] = targets[matches, 1:]
            out[:, 1:5] = xywh2xyxy(out[:, 1:5].mul_(scale_tensor))
        return out

```

```

    return loss

class BboxLoss(nn.Module):

    def __init__(self, reg_max, use_dfl=False):
        super().__init__()
        self.reg_max = reg_max
        self.use_dfl = use_dfl

    def forward(self, pred_dist, pred_bboxes, anchor_points, target_bboxes, target_scores, target_scores_sum, fg_mask):
        # IOU loss
        weight = torch.masked_select(target_scores.sum(-1), fg_mask).unsqueeze(-1)
        iou = bbox_iou(pred_bboxes[fg_mask], target_bboxes[fg_mask], xywh=False, CIoU=True)
        loss_iou = ((1.0 - iou) * weight).sum() / target_scores_sum

        # DFL loss
        if self.use_dfl:
            target_ltrb = bbox2dist(anchor_points, target_bboxes, self.reg_max)
            loss_dfl = self._df_loss(pred_dist[fg_mask].view(-1, self.reg_max + 1), target_ltrb[fg_mask]) * weight
            loss_dfl = loss_dfl.sum() / target_scores_sum
        else:
            loss_dfl = torch.tensor(0.0).to(pred_dist.device)

        return loss_iou, loss_dfl

    @staticmethod
    def _df_loss(pred_dist, target):
        # Return sum of left and right DFL losses
        tl = target.long() # target left
        tr = tl + 1 # target right
        wl = tr - target # weight left
        wr = 1 - wl # weight right
        return (F.cross_entropy(pred_dist, tl.view(-1), reduction="none").view(tl.shape) * wl +
                F.cross_entropy(pred_dist, tr.view(-1), reduction="none").view(tr.shape) * wr).mean(-1, keepdim=True)

```

```

# Model validation metrics
"""
import math
import warnings
from pathlib import Path

import matplotlib.pyplot as plt
import numpy as np
import torch
import torch.nn as nn

# from ultralytics.yolo.utils import TryExcept

# boxes
def box_area(box):
    # box = xyxy(4,n)
    return (box[2] - box[0]) * (box[3] - box[1])

def bbox_iou(box1, box2, eps=1e-7):
    """Returns the intersection over box2 area given box1, box2. Boxes are x1y1x2y2
    box1: np.array of shape (nx4)
    box2: np.array of shape (mx4)
    returns: np.array of shape (nm)
    """
    # Get the coordinates of bounding boxes
    b1_x1, b1_y1, b1_x2, b1_y2 = box1.T
    b2_x1, b2_y1, b2_x2, b2_y2 = box2.T

    # Intersection area
    inter_area = (np.minimum(b1_x2[:, None], b2_x2) - np.maximum(b1_x1[:, None], b2_x1)).clamp(0) * \
        (np.minimum(b1_y2[:, None], b2_y2) - np.maximum(b1_y1[:, None], b2_y1)).clamp(0)

    # box2 area
    box2_area = (b2_x2 - b2_x1) * (b2_y2 - b2_y1) + eps

    # Intersection over box2 area
    return inter_area / box2_area

```

```

def bbox_iou(box1, box2, xywh=True, GIoU=False, DIoU=False, CIoU=False, eps=1e-7):
    # Returns Intersection over Union (IoU) of box1(1,4) to box2(n,4)

    # Get the coordinates of bounding boxes
    if xywh: # transform from xywh to xyxy
        (x1, y1, w1, h1), (x2, y2, w2, h2) = box1.chunk(4, -1), box2.chunk(4, -1)
        w1_, h1_, w2_, h2_ = w1 / 2, h1 / 2, w2 / 2, h2 / 2
        b1_x1, b1_x2, b1_y1, b1_y2 = x1 - w1_, x1 + w1_, y1 - h1_, y1 + h1_
        b2_x1, b2_x2, b2_y1, b2_y2 = x2 - w2_, x2 + w2_, y2 - h2_, y2 + h2_
    else: # x1, y1, x2, y2 = box1
        b1_x1, b1_y1, b1_x2, b1_y2 = box1.chunk(4, -1)
        b2_x1, b2_y1, b2_x2, b2_y2 = box2.chunk(4, -1)
        w1, h1 = b1_x2 - b1_x1, b1_y2 - b1_y1 + eps
        w2, h2 = b2_x2 - b2_x1, b2_y2 - b2_y1 + eps

    # Intersection area
    inter = (b1_x2.minimum(b2_x2) - b1_x1.maximum(b2_x1)).clamp(0) * \
        (b1_y2.minimum(b2_y2) - b1_y1.maximum(b2_y1)).clamp(0)

    # Union Area
    union = w1 * h1 + w2 * h2 - inter + eps

    # IoU
    iou = inter / union
    if CIoU or DIoU or GIoU:
        cw = b1_x2.maximum(b2_x2) - b1_x1.minimum(b2_x1) # convex (smallest enclosing box) width
        ch = b1_y2.maximum(b2_y2) - b1_y1.minimum(b2_y1) # convex height
        if CIoU or DIoU: # Distance or Complete IoU https://arxiv.org/abs/1911.08287v1
            c2 = cw ** 2 + ch ** 2 + eps # convex diagonal squared
            rho2 = ((b2_x1 - b1_x2) ** 2 + (b2_y1 - b1_y2) ** 2) / 4 # center dist ** 2
            if CIoU: # https://github.com/Zzh-tju/DIoU-SSD-pytorch/blob/master/utils/box/box_utils.py#L47
                v = (4 / math.pi ** 2) * (torch.atan(w2 / h2) - torch.atan(w1 / h1)).pow(2)
                with torch.no_grad():
                    alpha = v / (v - iou + (1 + eps))
            return iou - (rho2 / c2 + v * alpha) # CIoU
        return iou - rho2 / c2 # DIoU
    return iou - (c_area - union) / c_area # GIoU https://arxiv.org/pdf/1902.09630.pdf

```

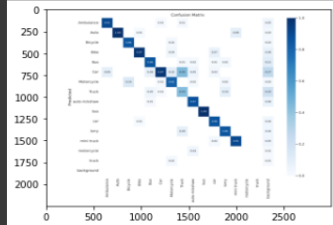
[illegible]

Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
12/50	120	0.1569	0.7745	0.9275	41	640: 100% 273/273 [03:42:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.371/s]	
all	1233	1233	0.769	0.693	0.762	0.737
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
13/50	120	0.1464	0.7802	0.9251	44	640: 100% 273/273 [03:43:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.361/s]	
all	1233	1233	0.758	0.753	0.758	0.753
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
14/50	120	0.1395	0.7893	0.9224	46	640: 100% 273/273 [03:42:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.381/s]	
all	1233	1233	0.764	0.699	0.768	0.759
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
15/50	120	0.1335	0.6837	0.9214	41	640: 100% 273/273 [03:43:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.371/s]	
all	1233	1233	0.765	0.671	0.758	0.753
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
16/50	120	0.1316	0.6831	0.9195	46	640: 100% 273/273 [03:41:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.391/s]	
all	1233	1233	0.768	0.719	0.813	0.813
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
17/50	120	0.1276	0.6442	0.9267	41	640: 100% 273/273 [03:43:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.391/s]	
all	1233	1233	0.889	0.675	0.767	0.76
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
18/50	120	0.1248	0.6383	0.9233	35	640: 100% 273/273 [03:43:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:27:00.000, 1.401/s]	
all	1233	1233	0.826	0.77	0.909	0.887
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
19/50	120	0.1194	0.6189	0.9385	44	640: 100% 273/273 [03:42:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:28:00.000, 1.391/s]	
all	1233	1233	0.76	0.792	0.833	0.819
Epoch	GPU mem	box loss	cls loss	dfl loss	Instances	Size
20/50	120	0.1316	0.5917	0.9333	43	640: 100% 273/273 [03:42:00.000, 1.231/s]
Class	Images	Instances	Box(P)	0	mAP50 mAP50-95: 100% 39/39 [00:27:00.000, 1.401/s]	
all	1233	1233	0.716	0.781	0.822	0.822

CONFUSION MATRIX

```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-main/runs/detect/train3/confusion_matrix.png")
imshow(testim)
```

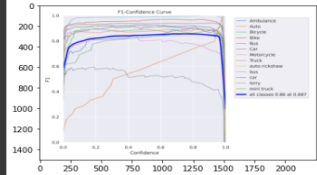
<matplotlib.image.AxesImage at 0x7f8c410b8a60>



F1 SCORE

```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-main/runs/detect/train3/F1_curve.png")
imshow(testim)
```

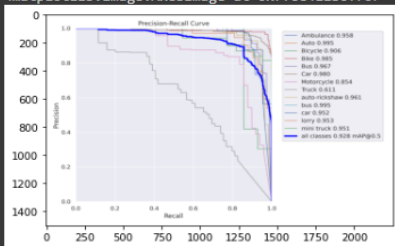
<matplotlib.image.AxesImage at 0x7f8c41a0fbb0>



PR CURVE

```
import cv2
from google.colab.patches import cv2_imshow
import matplotlib.image as mpimg
from matplotlib.pyplot import imshow
%matplotlib inline
testim = mpimg.imread("/content/YOLOv8-DeepSORT-Object-Tracking-main/runs/detect/train3/PR_curve.png")
imshow(testim)
```

<matplotlib.image.AxesImage at 0x7f8c411b6f70>



```

(x) 1 from yolox.tracker.byte_tracker import BYTETracker, STrack
    from yolo.metrics import iou, iou_batch
    from dataclasses import dataclass

    @dataclass(frozen=True)
    class BYTETrackerArgs:
        track_thresh: float = 0.25
        track_buffer: int = 30
        match_thresh: float = 0.8
        aspect_ratio_thresh: float = 3.0
        min_box_area: float = 1.0
        mot20: bool = False

    # Install Roboflow Supervision

    [ ] !pip install supervision==0.1.0

    from IPython import display
    display.clear_output()

    import supervision
    print(f"supervision.__version__: {supervision.__version__}")
    supervision.__version__ = 0.1.0

    [ ] from supervision.draw.color import ColorPalette
    from supervision.geometry.dataclasses import Point
    from supervision.video.dataclasses import VideoInfo
    from supervision.video.sources import get_video_frames_generator
    from supervision.video.sinks import VideoSink
    from supervision.notebook_utils import show_frame_in_notebook
    from supervision.tools import Detections, BoxAnnotator
    from supervision.tools.line_counter import LineCounter, LineCountAnnotation

```

```

1 def iou(bbox: tuple, candidates: tuple):
    iou = np.zeros((len(atlbrs), len(btbrs)), dtype=np.float)
    if iou.size == 0:
        return iou
    iou = np.ascontiguousarray(atlbrs, dtype=np.float),
    np.ascontiguousarray(btbrs, dtype=np.float)
    )
    return iou

def iou_distance(tracks, btrack):
    atlbrs = [track.tlbr for track in tracks]
    btbrs = [track.tlbr for track in btrack]
    iou = iou_batch(atlbrs, btbrs)
    cost_matrix = 1 - iou

def nearest_neighbour_distance(tracks, detections, metric='cosine'):
    cost_matrix = np.zeros((len(tracks), len(detections)), dtype=np.float)
    if cost_matrix.size == 0:
        return cost_matrix
    det_features = np.asarray([track.curr_feature for track in detections], dtype=np.float32)
    for i, track in enumerate(tracks):
        cost_matrix[i, :] = np.maximum(0.0, cdist(track.features, det_features, metric='cosine'))
    return cost_matrix

def mean_neighbour_distance(tracks, detections, metric='cosine'):
    if cost_matrix.size == 0:
        return cost_matrix
    track_features = np.asarray([track.curr_feature for track in tracks], dtype=np.float32)
    det_features = np.asarray([track.curr_feature for track in detections], dtype=np.float32)
    cost_matrix = cdist(track_features, det_features, metric)
    return cost_matrix

def gate_cost_matrix(cost_matrix, tracks, detections, only_position=False):
    if cost_matrix.size == 0:
        return cost_matrix
    gating_dim = 2 if only_position else 4
    gating_threshold = kalman_filter.chi2invoc[gating_dim]
    measurements = np.asarray([det.to_xyah() for det in detections])
    for row, track in enumerate(tracks):
        gating_distance = kf.gating_distance(
            cost_matrix[track_id, :], measurements, only_position)
        cost_matrix[row, gating_distance > gating_threshold] = np.inf
    return cost_matrix

```

```

1 # vim: expandtab:ts=4:sw=4
from future import absolute_import
import numpy as np
from yolox.deepsort_tracker import linear_assignment

def iou(bbox, candidates):
    bboxes_tl, bboxes_br = bbox[:2], bbox[2:] + bbox[2:]
    candidates_tl = candidates[:, :2]
    candidates_br = candidates[:, 2:] + candidates[:, 2:]

    tl = np.c_[np.maximum(bboxes_tl[0], candidates_tl[:, 0]), np.newaxis],
    br = np.c_[np.minimum(bboxes_br[0], candidates_br[:, 0]), np.newaxis],
    wh = np.maximum(0., br - tl)

    area_intersection = wh.prod(axis=1)
    area_bbox = bboxes[2:].prod()
    area_candidates = candidates[:, 2:].prod(axis=1)
    return area_intersection / (area_bbox + area_candidates - area_intersection)

def iou_cost(tracks, detections, track_indices=None,
            detection_indices=None):
    if track_indices is None:
        track_indices = np.arange(len(tracks))
    if detection_indices is None:
        detection_indices = np.arange(len(detections))

    cost_matrix = np.zeros((len(track_indices), len(detection_indices)))
    for row, track_id in enumerate(track_indices):
        if tracks[track_id].time_since_update > 1:
            cost_matrix[row, :] = linear_assignment.INFTY_COST
            continue

        bbox = tracks[track_id].to_tlwh()
        candidates = np.asarray(
            [detections[i].tlwh for i in detection_indices])
        cost_matrix[row, :] = 1 - iou(bbox, candidates)

    return cost_matrix

```

```

1 # settings
Run cell (Ctrl+Enter)
cell has not been executed in this session
TARGET_VIDEO_PATH = "/HOME/counting-result.mp4"

[ ] VideoInfo.from_video_path(SOURCE_VIDEO_PATH)
VideoInfo(width=3840, height=2160, fps=25, total_frames=538)

[ ] from tqdm.notebook import tqdm
byte_tracker = BYTETracker(BYTETrackerArgs())
# create VideoInfo instance
video_info = VideoInfo.from_video_path(SOURCE_VIDEO_PATH)
# create frame generator
generator = get_video_frames_generator(SOURCE_VIDEO_PATH)
line_counter = LineCounter(start=LINE_START, end=LINE_END)
box_annotator = BoxAnnotator(color=ColorPalette(), thickness=4, text_thickness=4, text_scale=2)
line_annotator = LineCounterAnnotator(thickness=4, text_thickness=4, text_scale=2)
with VideoSink(TARGET_VIDEO_PATH, video_info) as sink:
    # loop over video frames
    for frame in tqdm(generator, total=video_info.total_frames):
        # model prediction on single frame and conversion to supervision Detections
        results = model(frame)
        detections = Detections(
            xyxy=results[0].boxes.xyxy.cpu().numpy(),
            confidence=results[0].boxes.conf.cpu().numpy(),
            class_id=results[0].boxes.cls.cpu().numpy().astype(int)
        )
        # filtering out detections with unwanted classes
        mask = np.array([class_id in CLASS_ID for class_id in detections.class_id], dtype=bool)
        detections.filter(mask=mask, inplace=True)
        # tracking detections
        tracks = byte_tracker.update(
            output_results=detections2boxes(detections=detections),
            img_info=frame.shape,
            img_size=frame.shape
        )

```

```

    )
    # filtering out detections with unwanted classes
    mask = np.array([class_id in CLASS_ID for class_id in detections.class_id], dtype=bool)
    detections.filter(mask=mask, inplace=True)
    # tracking detections
    tracks = byte_tracker.update(
        output_results=detections2boxes(detections=detections),
        img_info=frame.shape,
        img_size=frame.shape
    )
    tracker_id = match_detections_with_tracks(detections=detections, tracks=tracks)
    detections.tracker_id = np.array(tracker_id)
    # filtering out detections without trackers
    mask = np.array([tracker_id is not None for tracker_id in detections.tracker_id], dtype=bool)
    detections.filter(mask=mask, inplace=True)
    labels = [
        f"{CLASS_NAMES_DICT[class_id]} {confidence:0.2f}"
        for _, confidence, class_id, tracker_id
        in detections
    ]
    # updating line counter
    line_counter.update(detections=detections)
    # annotate and display frame
    frame = box_annotator.annotate(frame=frame, detections=detections, labels=labels)
    line_annotator.annotate(frame=frame, line_counter=line_counter)
    sink.write_frame(frame)

```

100% 538/538 [01:48<00:00, 7.32it/s]

```

0: 384x640 3 cars, 1 truck, 65.4ms
Speed: 0.6ms preprocess, 65.4ms inference, 9.5ms postprocess per image at shape (1, 3, 640, 640)

0: 384x640 3 cars, 1 truck, 64.0ms
Speed: 0.9ms preprocess, 64.0ms inference, 2.2ms postprocess per image at shape (1, 3, 640, 640)

0: 384x640 4 cars, 2 trucks, 68.8ms
Speed: 0.6ms preprocess, 68.8ms inference, 2.2ms postprocess per image at shape (1, 3, 640, 640)

0: 384x640 5 cars, 1 truck, 84.5ms
Speed: 0.6ms preprocess, 84.5ms inference, 2.1ms postprocess per image at shape (1, 3, 640, 640)

0: 384x640 4 cars, 1 truck, 66.2ms
Speed: 0.6ms preprocess, 66.2ms inference, 1.9ms postprocess per image at shape (1, 3, 640, 640)

0: 384x640 4 cars, 1 truck, 64.0ms
Speed: 0.6ms preprocess, 64.0ms inference, 10.1ms postprocess per image at shape (1, 3, 640, 640)

```

SMART TRAFFIC MONITORING SYSTEM USING YOLO

¹Cheruku Nithesh Reddy, ²Nalla Uday Kiran, ³Dr.D. Adhimuga Sivasakthi

^{1,2}UG Student, Department of Information Technology Sathyabama Institute of Science and Technology, Chennai, India

³ Assistant Professor, Department of Information Technology Sathyabama Institute of Science and Technology, Chennai, India

¹ nitheshs938@gmail.com, ² Udaynalla6296@gmail.com, ³ adhimugasivasakthi.d.it@sathyabama.ac.in

Abstract: Traffic congestion arises from the daily addition of more cars to the road due to global population growth. By keeping an eye on traffic, accidents can be prevented. A model which can track, pick out, categorize cars is required to find careless drivers and other traffic infractions. Authorities need to calculate the wide variety of automobiles in a site visitors scenario because it allows them to keep away from accidents and traffic congestions, brought on due to traffic. Congestion. The method described in the article uses OpenCV and YOLO image processing tools to count, classify, and detect automobiles. The OpenCV software package is used to analyze images from incoming video to identify and recognize objects. In terms of speed and accuracy, the real-world vehicle discovery technique you only look once performs better than the existing object detection methods. CNN and other M.L techniques have significantly improved the performance and fidelity of automobile recognition and classification, allowing for the real-time study of massive volumes of data. This technology opens the door to autonomous driving, better traffic flow, and increased driving safety. Accurate car counting and identification are essential for improving highway management systems in the dynamic traffic environment of today. This research offers a YOLOv8 algorithm-based real-time traffic intelligence monitoring system for vehicle recognition and classification. Through the use of deep learning and computer vision techniques, the system seeks to enhance traffic control. Using annotated datasets, the YOLOv8 model is trained as part of the system's methodology to learn characteristics of vehicles such size, shape, texture, and color. Following training, the model is applied to infer boundaries of boxes and class probabilities in order to identify automobiles in real-time photos and videos. To improve detections and lower false positives, post-processing methods like confidence thresholding & non-maximum suppression are used. A module for classifying vehicles is also included in the system; it divides vehicles into many groups, including bikes, tempos, jeeps, and trucks. To effectively prepare photos for classification, this module employs data preprocessing techniques such as scaling and normalization. Using a varied dataset of car photos, the system's efficacy is assessed during the testing, validation, and training phases. In addition to improving traffic flow evaluation and management, the results show how well the system can identify and categorize vehicles in a variety of traffic scenarios.

Keywords: Data, Machine Learning, Counting, Classification, and Vehicle Detection.

I. INTRODUCTION

Compared to a few years ago, there are a wide variety of automobiles on the road nowadays vehicle identification and counting are becoming more and more crucial to improved highway management systems. The principal goals of the purpose of automobile identification technology's applications, or the technology itself, is for monitoring the vehicle. the usage of its coordinates in snap shots or motion pictures. Automobile detection performance and accuracy are essential components of traffic control systems. Video monitoring systems have been watching over safe locations for a very long time. A significant amount of film can be collected for investigation when traffic surveillance cameras are installed appropriately. The discovery and monitoring of automobiles by means of traffic monitoring systems is another useful method for managing and controlling traffic on the roads. Because cars come in different sizes, the accuracy of the vehicle count is still directly impacted by detection issues. In computer vision, vehicle detection is a major and ongoing challenge. It is essential to research intelligent transportation systems to keep highways safe and effective. These days, object detection is used in a wide range of industries. Video surveillance, sports broadcasting, and robot navigation are a few use cases. Vehicle classification is the process of grouping cars based on their attribute. Detecting and classifying vehicles using the conventional approaches would not yield the desired level of performance. Technological advancements have enabled the utility of system imaginative and prescient and different system gaining knowledge of and deep learning approach. These techniques increase the model's detection accuracy while effectively identifying items. Vehicle identification, vehicle counting, vehicle respective length dedication, and vehicle respective distance dedication are a few of the lesser tasks that this system needs to complete. Some people find it difficult to get close-up photos of every car in

their neighborhood because they move past so quickly. This software can be automated to overcome the limitation that it is not possible to watch a movie while pausing to snap a picture, and it is incredibly well-trained to achieve time-saving quality. Vehicle recognition technologies partially address this issue by tracking individual cars. These systems can identify various automobiles in photos captured from a moving platform. Within the first level, the video series of frames containing moving vehicles are recognized through combining object reputation and deep learning strategies. One crucial component of traffic management systems is the classification of automobile procedures. Numerous uses, such as statistical analysis and reporting, are possible for vehicle classification results. Car counts are essential to traffic flow analysis because they enable the traffic department to pinpoint problem areas and implement the appropriate corrective measures to lower accident and traffic rates. Traffic flow analysis might potentially save lives by reducing the overall number of accidents by identifying and penalizing rule breakers.

II. RELATED WORK

The corpus of current research on automobile categorization highlights the necessity for accurate and productive categorization techniques for a wide range of delivery applications, while also offering insightful records at the creation of unique algorithms and methodology for automobile categorization. Below is a list of literary analyses by various reporters.

Table 1: ANALYSIS OF RESEARCH

Researchers	Current Technology	Technology That Is Provided.	Limitation
Ross GJeff Donahue, Trevor Darrell, and Jitendra Malik	Rich feature hierarchies for accurate object detection and semantic segmentation.	The aggregation of You only looks once and Deepsort improves categorization accurate while improving real-time vehicles discovery.	No information on speed estimation or cars operating at night is included. The resolution and frame rate are restricted to 960*540 and 29 images per second, respectively.
Scott Reed	The default anchor box sizes might not be suitable for very small objects.	DL algorithms like You only look once are used to improve the model's efficiency and correctness.	License plate recognition is a tiny target recognition problem since license plates often make up a small percentage of the entire image. Noise and poor resolution make it difficult to distinguish little items.
Cryel ontimare manlises	YOLO-V4 is used to aid the process of automotive recognition and categorization.	Precision is increased by adding additional improvements when using YOLO V8.	Cars may be reliably identified and tracked using the MHDT technique even under challenging situations such as scale fluctuation, occlusion, rotation, stopped cars, and changing lighting.

P. Adarsh, M. Kumar	For automobile recognition, the somewhat sluggish YOLO-V3 inadequate edition is employed.	YOLO-V8 helps to make the system more rapid and precise.	Both the Kernelized Correlation Filter-based and the Kalman Filter-based tracking strategies were used. Vehicle movement was monitored with the use of these techniques, which increased tracking accuracy.
P. Kumar and S. Sharma	Sensors, which might be every now and then highly priced, are used to track automobiles.	DeepSort enable real-time object detection. YOLO used to break video into frames.	It offers no in-depth comparison analysis or explanation of how it stacks up against other methods.
Zarema Balgabekova	Object recognition is a machine learning problem that involves the correct classification and localization of objects in an image	Object recognition is improved with the help of you only look once Byte track algorithms.	

III. METHODOLOGY

A real-time automobile identification, including, and classification system using you only look once is proposed in this work and can be used for traffic recorder vehicle categorization. Yolo is among the greatest algorithms for recognizing items and classifying them appropriately. Compared to the faster RCNN, it performs better [13].

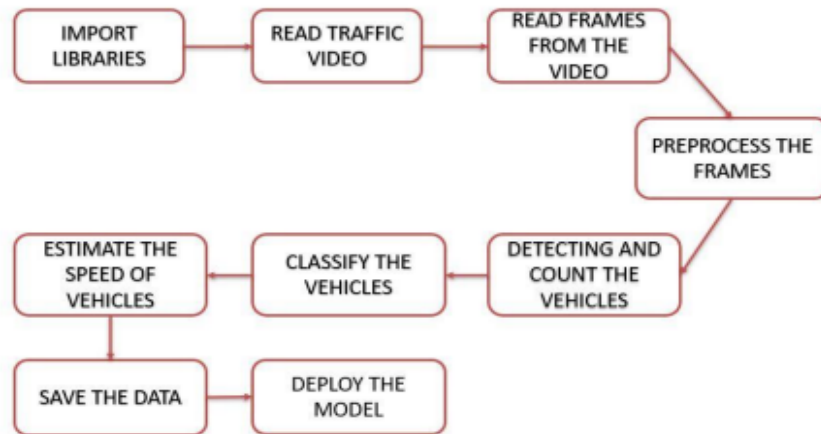


FIGURE. 3.1. Overall Design of Smart Traffic Monitoring System.

Vehicle Detection

A. Training:

The annotated dataset has to be used to train the YOLOv8 model. The model learns features of cars, such as form, texture, color, and size, during training. The model's training process can take days or weeks to finish and demands a large amount of computing capabilities. To reduce the difference between the expected and marked bounding rectangles, the model's weights are adjusted during training.

B. Inference:

The algorithm may be trained to recognize cars in recent snapshots and films. throughout conclusion, the you only look once model recognizes objects in the input image using a variety of convolutional filters. By using the framework, a collection of containers but associated category chances per every vehicle that is discovered.

C. Post-Processing:

Following you only look once algorithm is run, the result is put up refined to improve the bounding boxes that are discovered and eliminate any unnecessary or false-positive detections. Low-confidence detections are minimized by selecting a confidence threshold, removing overlapping detections with non-maximum suppression (NMS), and adjusting bounding boxes to good suit the recognized automobiles.

Classification of vehicles

It is a massive task to categorize all of the various types of automobiles that exist. Finally, records are categorized is divided into different automobiles types: Tempos, jeep, lorries, bikes. Following things are done in developing the vehicle categorization model are as follows:

- 1) Downloading and installing the requirements for you only look once algorithm from GitHub. Replicate the GitHub repository and install all of the libraries needed to use the YOLO model, including pandas, sklearn, seaborn and others.
- 2) Data accumulation and classification according to vehicle type (Tempos, jeep, lorries, bikes). To train a YOLO model for vehicle categorization, a large amount of data and computing power are required, which is a demanding thing. information collection, annotation, training, and assessment are crucial for model success since data affects model efficiency.



FIGURE. 3.2.1 Categorization of automobiles.

The above image depicts vehicle confidence levels as well as class-based classification.

The confidence value expresses the automobiles find out degree of fact that an object is within a defined bounding box. Among the you only look once model's result, the confidence value, required to reject low-confidence discovery which may be F.P or have wrong localization [8].

Every container is allocated a confidence result via YOLO, and simplest discovery with excessive required confidence output are saved for additional transformation. This ensures that false positive detections have no influence on the model and that it only predicts things that it is certain about [16]. Along with the confidence level, YOLO produces containers position, the item name, the containers size. The bounding boxes around the objects in a picture are drawn using the results that are utilized for automobile detection.

3.2.1 Data Set Preparation:

Free snapshots, an opensource cv dataset, is the source of the data. The Yolo paradigm [10] anticipates txt files with comments attached. Using "Roboflow" software, information has been marked inside containers. Using this dataset, the model's efficiency is evaluated and expertise. 3 folders, "training," "Validation," and "testing," have been created from the dataset. Snapshots with names and containers information are contained in each folder. The collection includes 627 photos of various vehicles, including Tempos, jeep, lorries, bikes. The photographs in the collection were gathered in a way that allowed for a variety of backdrop circumstances, scenarios, and backgrounds. This facilitates more effective training of the model.

3.2.2 Data Preprocessing:

1)Resizing:

The images are resized using the following procedures after the data has been loaded into the notebook:

- 1) The input image is cropped or padded into a square shape, maintaining the aspect ratio. The most common way to get the smallest aspects of the snapshots to equal the length of the larger side is to add zeros to it.
- 2) To facilitate the cognitive network's effective processing of the inflated or trimmed picture, a snapshot is converted to a set size, usually a pow(2). 640 * 640 pixels is the most common YOLO size.
- 3) Dividing the shrunk image's pixel values by 255 is the usual method for normalization. This stage ascertains whether the input values fall within the permissible range of the neural network, which is normally limited to values between 0 and 1.
- 4) The normalized image is removed from the pixel mean value. The purpose of this phase is to remove any lightness or dullness changes based on the snapshot that could interfere with object detection.
- 5) After normalization and the finished image is fed through the YOLO neural system for recognizing things and mean-subtraction categorization.
- 6) To detect objects, YOLO splits given picture onto an area of the pattern. The job for every part is to predict a collection of containers as well as associated class probabilities for its items. The YOLO method generates a collection of containers and related class likelihoods for everything identified in the snapshot. These border boxes make it possible to identify and classify the many objects in the image, like the vehicles on a highway scenario.

2)Normalization:

The snapshots pixel values must be normalized to enable the simulation to allow one to pick things up faster and accurately. This is essential for object detection [4]. Since the weights of the model are initialized with a small random number, Gradients are enabled when the input is normalized. to flow through the network efficiently. The model can pick up knowledge faster and produce an improved response in this way. The input image scales are set to 0 or 1 throughout the normalizing process. The model learned more effectively because both the images and the bounding boxes were standardized. Despite the aspects and place of the items, containers regularization is crucial since it aids the model in understanding the connections between the objects that are included and the containers.

3) Train, Validate, and Test the model:

The first two develop and verify play a vital function in determining the system's overall effectiveness of the model. From the accumulated information seventy percent is used for training one fifth of 100 for the system's verification, but they left out info in assessing the functionality of algorithm overall efficiency. The developed system put to the test and its overall effectiveness is assessed. using 130 photos of confirmation. The purpose of

the validation is to confirm the cars' category. The annotated dataset is used to train the model through the use of the "PyTorch" deep learning framework. YOLO is based on a (CNN) [10] that is developed to comprehend and categorize objects in a film. Because CNN design is complex and developed a lot of computing power to train, powerful GPUs have been employed [9].

Variable adjustment is necessary for improved performance once the model has been developed. Thus, factors and hyperparameters are employed to optimize overall efficiency. In addition, improve the overall performance of algorithms and keep away from overfitting issues. know-how the features affect the algorithm's overall efficiency became made less complicated using the technique. adjusting system's variables has been accomplished the usage of the trial-and-errors technique. traffic motion pictures have been gathered from visitors' surveillance cameras and used in this have a look at for automobile discovery and categorization. To decide the algorithm's overall efficiency, a unique test dataset was employed. To do that, metrics such as accuracy, recall, are calculated, the procedure of adjusting features has been frequently done the overall efficiency of the system touches the satisfactory level.

Counting of vehicles

The usage of you only look once to remember automobiles can provide beneficial information that are employable to reinforce productiveness, protection, along with security in an expansion of scenarios. The real-time automobile monitoring functionality permits organizations to produce information-driven judgments that may optimize processes and eventually lead to cost reduction [12]. In measuring the automobiles in a film placing a straight line is essential that depend features is asserted having the result 0.

Every time an automobile passes the line, it is measured. The number in the parameter is increased, and at the conclusion, the number of automobiles is provided. YOLO employs a method to guarantee that an automobile is only captured once, even if it's visible in many frames [13]. When a automobiles container differs from any other discovered enclosing box, YOLO labels it as new. It does this through analyzing the containers of the identified automobiles across frames.



FIGURE. 3.3.1 Hypothetical path used to tally each automobile.

The above diagram shows a trajectory which measures the total quantity of automobiles in the window; each time an automobile passes the path, the number in the count is increased. Listing every kind of automobile is thus accomplished.

The DeepSORT algorithm [4] monitors each vehicle. gradually once noticing it the use of an aggregate of look capabilities and movement forecast. this is executed through calculating the resemblance rating among the entry of each automobile within its current window as well as how it appeared in previous window. Every automobile in the previous shot is compared to the other using a comparison score. Following the tracking of every vehicle, the DeepSORT algorithm may be used to determine the range of automobiles in the clip. Set of rules [1]. that is carried out through measuring the determining code which are It is attributed to every single automobile at some point of Recognition of automobiles. modifications in the surroundings, lights, or other elements through

DeepSORT do no longer influence the accuracy of the automobile calculating. DeepSORT integrates recognition of items, object tracking, statistics affiliation as well as path assessment to measure number the quantity of automobiles in film movement [17]. The diffusion of programs, which includes the monitoring of traffic, is a novel and entirely environmentally friendly method of measuring automobiles in the moment.

Byte Track uses object recognition and monitoring in conjunction with monitoring automobiles as its items of attention. This is achieved by predicting each automobile location in the next panel based on its movements in the preceding shots. Next, a comparison score between the actual spot in the next shot and what was predicted is calculated. The similarity score is utilized to assess if an automobile in the subsequent picture is identical as an automobile in the previous frame. Byte track analyzes the characteristics of an object across multiple images to determine its validity. they're the identical asset. It acts in such a manner through the usage of a metric referred to as the Measurement of Cosine Distance. The Cosine Distance Measure compares 2 distinctive vectors—which, in the instance of Byte Track, depict an item's appearance in two different frames to evaluate exactly how comparable they are. The Cosine Distance Metric calculates the provided distance and delivers a value between -1 and 1. Perspective among the two defining variables. A cost of one indicates whole identification among 2 characteristic carriers, with a cost of -1 signifying total differentiation. In order to monitor an item over time, Byte Track first estimates where it will be in the subsequent picture based on how it moved in the preceding pixels. Next, by comparing the anticipated position with the precise positions of the items observed in the succeeding framework, the Cosine Distance Metric can be used for evaluating how similar the visual characteristics of the object inside each image are. The Byte Track is used to figure out whether the item in the subsequent screen is the same as the one in the previous frame. If the degree of similarity is greater than an established limit, that item is assigned the same identity code.

IV. RESULTS

In this study on YOLO-based finished automobile identification via categorization, a collection of images automobiles utilized regarding development of a system and assessment. Various vehicle kinds and positions have been captured in the collection of data. With a mAP (50-ninety-five) of 92.7% and an acceptable precision of 92.8%, the model's abilities to recognize and classify autos are demonstrated. Despite this, the YOLO version may system images in the present moment. With an extraordinarily substantial amount of algorithmic intricacy. real-time. Even though the YOLO machine learning model's processing expenses increased substantially, it was still able to analyze snapshots with actual moment.

Model summary: 268 layers, 43618173 parameters, 0 gradients, 158.0 GiB/s

Class	Images	Instances	BoxP	P	AP50	AP50-95
all	1231	1231	0.88	0.86	0.928	0.827
ambulance	1231	22	0.824	0.955	0.958	0.958
Auto	1231	2	0.531	1	0.995	0.995
Bicycle	1231	14	0.845	0.857	0.906	0.906
bike	1231	280	0.905	0.904	0.905	0.902
bus	1231	79	0.905	0.836	0.967	0.967
Car	1231	554	0.919	0.955	0.98	0.979
motorcycle	1231	49	0.857	0.732	0.854	0.854
truck	1231	69	0.819	0.777	0.811	0.811
auto-rickshaw	1231	56	0.96	0.867	0.961	0.961
bus	1231	30	0.991	1	0.995	0.995
car	1231	54	0.979	0.862	0.952	0.952
lorry	1231	42	0.902	0.672	0.953	0.947
mini truck	1231	22	0.867	0.891	0.951	0.951

Speed: 0.4ms pre-process, 16.7ms inference, 0.0ms loss, 1.4ms post-process per image
 Saving /content/YOLOv8-DeepSORT-Object-tracking-main/runs/detect/train/predictions.json...
 Results saved to /content/YOLOv8-DeepSORT-Object-tracking-main/runs/detect/train

FIGURE 4. Model's Efficiency.

All automobile categories had excellent outcomes from the model; the Tempo category had the best correctness (99.5%), while the lorry category had the lowest (61.1%). Further investigations may enhance the design, but for now it was impossible to accurately identify tiny automobiles, such as bikes. The trained model fared better in terms of precision and recall than previous approaches. These results demonstrate the way well YOLO operates as an algorithm for automobile identification and categorization duties, and they raise the possibility of a wide range of practical applications, such as self-driving automobiles and surveillance of traffic. In further study, the algorithm

will be evaluated on a bigger and more diverse dataset, and methods to improve the effectiveness of the model on little automobiles are being looked at.

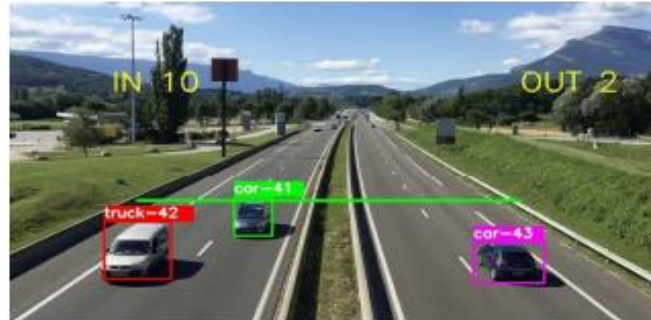


FIGURE.5. Result.

The aforementioned graphic displays the classes of each group of cars, the total amount of arriving and exiting automobiles, and a hypothetical line that was constructed to determine the number of automobiles passing the designated line.

V. DISCUSSIONS

Yolo is the most efficient for vehicle recognition and categorization due to its instantaneous image processing capabilities. This makes it an appropriate framework to use in practical applications where fast computations are essential, such as autonomous vehicle operation and traffic surveillance. YOLO's synchronous recognition of items and categorization skills can reduce processing difficulty and increase the productivity of boom systems.

Nevertheless, a number of several hazards associated with using YOLO for vehicles categorization and identification. Its enormous computing material consumption would exclude its employment in contexts with scarce assets among other drawbacks. A further shortcoming of the simulation may be its inability to distinguish between small automobiles; this is an issue that will be fixed in future research. The version needs to be adjusted as well for the particular tasks and information that might require the journey in order to produce the extraordinary results. On a typical basis, YOLO is a strong model framework for recognizing and categorizing cars, Nevertheless, they have certain limitations. Future study may concentrate on improving the version's efficiency on compact cars and creating.

VI. CONCLUSION

In summary, tasks involving cv such as automobile recognition and categorization are critical since they are utilized in a wide range of programs, such as autonomous vehicles, self-driving automobiles, and surveillance of traffic. Many artificial intelligence (AI) approaches, which includes as CNN and yolo v8 algorithms, templates, and feature-based approaches approaches, are currently developed for the categorization and recognition of automobiles.

CNN simulations is shown to exist extremely successfully through problems requiring a categorization for identification of automobiles because of being able to obtain exact data from photographs and tolerating differences in illumination and perspective. CNN is used in the "YOLO" real-time detection of objects method [12] to identify items in a film or photograph. Yolo enables for instantaneous tracking and choice-making, which makes it possible to deliver helpful knowledge that can be utilized in a number of circumstances to increase work efficiency and protection.

Ultimately, the accuracy of automobile categorization and recognition has risen considerably during the previous several decades as a consequence of the widespread implementation of artificial intelligence algorithms.

Particularly when it comes to managing obstacles and identifying automobiles in poor brightness, there may be still opportunity for development. YOLO's inability to recognize a lot simultaneously whenever they're located near each other constitutes one of its issues. Subsequent research endeavors in this domain will focus on devising resolutions for these issues and augmenting the efficacy of automobile categorization and recognition tools.

VII. ACKNOWLEDGMENTS

Firstly, we would like to thank our Professor, Adimuga Sivashakthi. She motivated us to work on this project without second thoughts and encouraged us to think creatively. This assignment was completed with her full support and teaching aids. Despite our doubts, she believed in us. In addition, we are grateful to each and every member of this group. This assignment was a success because of each and every individual's contribution.

VIII. REFERENCES

1. A. Pujara and M. Bhamare, "DeepSORT: Real Time & Multi-Object Detection and Tracking with YOLO and TensorFlow," 2022 International Conference on Augmented Intelligence and Sustainable Systems (ICAISS), Trichy, India, 2022, pp. 456-460, doi: 10.1109/ICAISS55157.2022.10011018.
2. K. V. Arya, S. Tiwari, and S. Behwalc, "Real-time vehicle detection and tracking," 2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications, and Information Technology (ECTI-CON), Chiang Mai, Thailand, 2016, pp. 1-6, doi: 10.1109/ECTICon.2016.7561327.
3. M. A. El-Khoreby and S. Abd Rahman Abu-Bakar, "Vehicle detection and counting for complex weather conditions," 2017 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuching, Malaysia, 2017, pp. 425-428, doi: 10.1109/ICSIPA.2017.8120648.
4. M. Gluhaković, M. Herceg, M. Popovic and J. Kovačević, "Vehicle Detection in the Autonomous Vehicle Environment for Potential Collision Warning," 2020 Zooming Innovation in Consumer Technologies Conference (ZINC), Novi Sad, Serbia, 2020, pp. 178-183, doi: 10.1109/ZINC50678.2020.9161791.
5. M. I. H. Azhar, F. H. K. Zaman, N. M. Tahir, and H. Hashim, "People Tracking System Using DeepSORT," 2020 10th IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Penang, Malaysia, 2020, pp. 137-141, doi: 10.1109/ICCSCE50387.2020.9204956.
6. M. M. Sein, K. S. Htet, K. T. Murata, and S. Phon-Amnuaisuk, "Object Detection, Classification and Counting for Analysis of Visual Events," IEEE 9th Global Conference on Consumer Electronics (GCCE), Kobe, Japan, 2020, pp. 274-275, doi: 10.1109/GCCE50665.2020.9292058.
7. N. S. Chauhan, F. Rahman, R. Sarker and M. M. H. Pious, "Vehicle detection, tracking and counting using linear quadratic estimation technique," 2018 2nd International Conference on Inventive Systems and Control (ICISC), 2018, pp. 603-607, doi: 10.1109/ICISC.2018.8398871.
8. N. Shree Adhithya, C. C. Sunil Kumar, V. Devendra, and C. V. Giriraja, "Frame by Frame Analysis for Counting of Vehicles from Aerial Video," 2018 3rd International Conference on Communication and Electronics Systems (ICCES), Coimbatore, India, 2018, pp. 883-886, doi: 10.1109/CESYS.2018.8724017.
9. P. Adarsh, P. Rathi and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," 2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS), Coimbatore, India, 2020, pp. 687-694, doi: 10.1109/ICACCS48705.2020.9074315.
10. P. Ajitha, J. S. Y. N. Krishna K and A. Sivasangari, "Vehicle Model Classification Using Deep Learning," 2021 5th International Conference on Trends in Electronics and Informatics (ICOEI), Tirunelveli, India, 2021, pp. 1544-1548, doi: 10.1109/ICOEI51242.2021.9452842.
11. P. Kumar and S. Sharma, "A Computer Vision Based on Vehicle Detection and Counting System Using Sensor Security," 2021 4th International Conference on Recent Developments in Control, Automation & Power Engineering (RDCAPE), Noida, India, 2021, pp. 624-629, doi: 10.1109/RDCAPE52977.2021.9633454.
12. Q. Pan, Y. Guo, and Z. Wang, "A scene classification algorithm of visual robot based on Tiny Yolo v2," 2019 Chinese Control Conference (CCC), Guangzhou, China, 2019, pp. 8544-8549, doi: 10.23919/ChiCC.2019.8866632.
13. R. R. Chandrika, N. S. G. Ganesh, A. Mummoorthy and K. M. K. Raghunath, "Vehicle Detection and Classification using Image processing," 2019 International Conference on Emerging Trends in Science and Engineering (ICESE), Hyderabad, India, 2019, pp. 1-6, doi: 10.1109/ICESE46178.2019.9194678.

14. S. Tabassum, M. S. Ullah, N. H. Al-Nur and S. Shatabda, "Native Vehicles Classification on Bangladeshi Roads Using CNN with Transfer Learning," 2020 IEEE Region 10 Symposium (TENSYP), Dhaka, Bangladesh, 2020, pp. 40-43, doi: 10.1109/TENSYP50017.2020.9230991.cnn
15. S. I. C. Imperial, A. L. L. Lucas, and M. V. Caya, "Vehicle Type Classification and Counting Using YOLOv4 Algorithm," 2022 IEEE International Conference on Artificial Intelligence in Engineering and Technology (IICAET), Kota Kinabalu, Malaysia, 2022, pp. 1-6, doi: 10.1109/IICAET55139.2022.9936874.
16. X. Hou, Y. Wang and L. -P. Chau, "Vehicle Tracking Using Deep SORT with Low Confidence Track Filtering," 2019 16th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), Taipei, Taiwan, 2019, pp. 1-6, doi: 10.1109/AVSS.2019.8909903.
17. Y. Lee, C. Lee, H. -J. Lee and J. -S. Kim, "Fast Detection of Objects Using a YOLOv3 Network for a Vending Machine," 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 2019, pp. 132-136, doi: 10.1109/AICAS.2019.8771517.