A Major Project report on

TRAFFIC DENSITY ANALYSIS

Submitted to Anurag University in Partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Artificial Intelligence

Submitted by

Anushna Vallampatla

20EG106244

Under the Guidance of

Dr. A. Mallikarjuna Reddy

Associate Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

School Of Engineering

ANURAG UNIVERSITY

2020-2024

ANURAG UNIVERSITY | SCHOOL OF ENGINEERING

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

DEPARTMENT OF ARTIFICIAL INTELLIGENCE

CERTIFICATE

This is to certify that the project report titled Traffic density analysis is being submitted by Anushna Vallampatla bearing 20EG106244 in IV B. TECH II semester *Artificial Intelligence* is a record bonafide work carried out by them. The results embodied in this report have not been submitted to any other University for the award of any degree.

Student's Name	Student's Signature
Anushna Vallampatla	

Internal Guide	**Head of the Department**
Dr. A Mallikarjuna Reddy	Dr. A. Mallikarjuna Reddy

External Examiner

ACKNOWLEDGEMENT

We owe our gratitude to **Prof. S. Ramchandram** , Vice-Chancellor, Anurag University, for extending the University facilities to the successful pursuit of our project so far and his kind patronage.

We acknowledge our deep sense of gratitude to **Prof. Balaji Utla**, Registrar, Anurag University, for being a constant source of inspiration and motivation.

We wish to record our profound gratitude to **Dr. V. Vijaya Kumar**, Professor, Dean – School of Engineering, for his motivation and encouragement.

We sincerely thank **Dr. A. Mallikarjuna Reddy**, Associate Professor and the Head of the Department of Artificial Intelligence, Anurag University, for all the facilities provided to us in the pursuit of this project.

We owe a great deal to our project coordinator **Ms. M. Madhavi**, Assistant Professor Department of Artificial Intelligence, Anurag University for supporting us throughout the project work.

We are indebted to our project guide **Dr. A. Mallikarjuna Reddy**, Associate Professor, Department of Artificial Intelligence, Anurag University. We feel it’s a pleasure to be indebted to our guide for his valuable support, advice, and encouragement and we thank him for his superb and constant guidance towards this project.

TABLE OF CONTENTS

ABSTRACT

Real-time traffic density estimation is a critical component of modern urban and traffic management systems, providing insights into traffic flow patterns, congestion levels, and peak traffic periods. This focuses on developing a comprehensive toolset utilizing advanced deep learning techniques, specifically leveraging the YOLOv8 (You Only Look Once) model, for accurate vehicle counting and density estimation in urban environments. The primary objective is to create a robust framework capable of efficiently detecting and counting vehicles within designated areas in real-time. The methodology encompasses data collection, model development, training, evaluation, and validation stages. By training the YOLOv8 model on a custom dataset and evaluating its performance using standard metrics, including precision, recall, and F1-score, aims to validate the model's effectiveness in diverse traffic scenarios. The toolset's applicability is demonstrated through the analysis of traffic density in sample images and videos, enabling the identification of congestion hotspots and informing data-driven traffic management strategies. Through this endeavour, seeks to contribute significantly to the enhancement of traffic management practices and the optimization of urban mobility, ultimately fostering more sustainable and efficient urban environments.

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Urbanization trends have led to unprecedented challenges in urban mobility, with traffic congestion being a significant concern for cities worldwide. As urban populations continue to grow, traditional methods of traffic surveillance and management struggle to cope with the increasing demand for efficient transportation solutions. Conventional approaches often rely on manual monitoring and static infrastructure, resulting in limited effectiveness in addressing dynamic traffic patterns and congestion hotspots.

In this context, the integration of digital technologies, particularly artificial intelligence (AI) and computer vision, offers a promising avenue for transforming urban mobility management. By leveraging advanced algorithms and real-time data analysis, cities can gain valuable insights into traffic dynamics, optimize public transportation systems, and mitigate congestion-related issues. Deep learning-based object detection techniques, such as the YOLO (You Only Look Once) model, have emerged as powerful tools for accurately detecting and classifying objects, including vehicles, in complex urban environments.

The motivation behind this problem statement stems from the urgent need to develop innovative solutions for addressing traffic congestion and enhancing urban mobility. By harnessing the capabilities of state-of-the-art technologies, such as YOLOv8, cities can move towards more sustainable and efficient transportation systems, ultimately improving the quality of life for residents and visitors alike.

1.1 Navigating the Modern Urban Gridlock: Challenges and Solutions

Modern urban areas face a pervasive challenge of traffic congestion due to rapid urbanization and increasing vehicle ownership, resulting in crowded roadways and extended commute durations. The rise of ride-sharing services and delivery vehicles further complicates traffic patterns, impacting productivity and air quality. Efforts such as smart city initiatives utilize technology and data analytics to mitigate congestion, yet infrastructure development often lags behind urban growth, exacerbating traffic issues. Momentum is building for sustainable transportation solutions like public transit expansion and promotion of cycling and walking. The integration of autonomous vehicles and intelligent traffic management systems shows promise for congestion reduction and safety enhancement. Collaboration among policymakers, urban planners, and transportation experts is crucial to tackle the complex challenges posed by modern traffic. effectively.

1.2 Utilizing Object Detection in Traffic Management

Object detection technology plays a pivotal role in modern traffic management systems by providing real-time insights into vehicle movement and congestion. By deploying cameras equipped with object detection algorithms in strategic locations, authorities can monitor traffic flow, identify congestion hotspots, and optimize signal timings accordingly. This proactive approach enables more efficient traffic management and reduces congestion-related delays, ultimately enhancing overall road safety and commuter experience.

Fig 1.1 A Snapshot of Traffic

Disadvantages of Satellite View in Traffic Management

While satellite imagery offers a wide-area view of traffic conditions, it has limitations in providing detailed information required for precise traffic management. Factors such as cloud cover, resolution limitations, and latency in data transmission can hinder the accuracy and timeliness of satellite-based traffic monitoring. Additionally, satellite imagery may not capture real-time changes in traffic patterns, limiting its effectiveness in dynamic traffic environments.

Vehicle Detection with OpenCV and YOLOv8

Vehicle detection using OpenCV and YOLOv8 involves preprocessing video frames to identify vehicles based on predefined features and patterns. OpenCV provides a robust framework for image processing tasks, such as edge detection and contour analysis, essential for vehicle detection. YOLOv8, on the other hand, utilizes deep learning techniques to detect and classify vehicles with high accuracy and efficiency. By leveraging pre-trained neural network models, YOLOv8 can recognize vehicles in complex traffic scenarios, making it a powerful tool for traffic management applications.

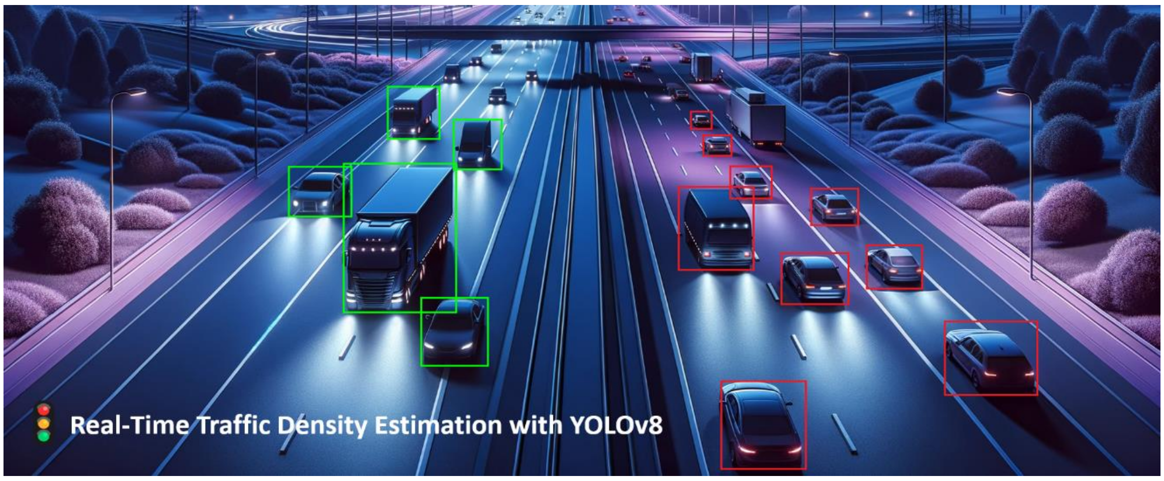


Fig 1.2 Real-Time Traffic Density Estimation With Yolov8

Differences between OpenCV and YOLOv8 for Vehicle Detection

Framework: OpenCV is an open-source computer vision library that provides a wide range of image processing functions, while YOLOv8 is a deep learning-based object detection model specifically designed for real-time detection tasks.

Accuracy: YOLOv8, being a deep learning model, typically offers higher accuracy in vehicle detection compared to traditional image processing techniques employed in OpenCV.

Speed: YOLOv8 is optimized for speed and can perform real-time detection on video streams, whereas OpenCV-based approaches may be computationally intensive and slower.

Flexibility: OpenCV provides flexibility in implementing custom image processing pipelines, while YOLOv8 offers a pre-trained model that can be easily integrated into existing systems with minimal customization.

Complexity: Implementing vehicle detection with YOLOv8 may require familiarity with deep learning concepts and training procedures, whereas OpenCV-based approaches are generally more straightforward to implement for basic detection tasks.

Resource Requirements: YOLOv8 may require more computational resources, particularly during training and inference, compared to OpenCV-based approaches, which can run efficiently on resource-constrained devices.

Adaptability: YOLOv8 can adapt to diverse traffic scenarios and vehicle types with minimal manual intervention, whereas OpenCV-based approaches may require fine-tuning and parameter adjustments for optimal performance in different environments.

2. LITERATURE SURVEY

2.1 Existing Systems:

Traffic Density Monitoring Control System Using Convolution Neural Network oct 2023. [1]

The author proposed a Traffic Density Monitoring System (TDMS) based on Convolutional Neural Networks (CNNs) for real-time traffic density estimation. Unlike traditional methods, this system autonomously learns from traffic photos, improving scalability and efficiency. It also advocates for an intelligent traffic signal control framework using deep reinforcement learning, enhancing adaptability to changing traffic patterns. Validated with real-world data, this approach promises to enhance transportation management, leading to safer and more efficient road networks.

Real-Time Traffic State Measurement Using Autonomous Vehicles Open Data 2023[2]

The author proposed a framework showcasing the potential of autonomous vehicle (AV) technologies to revolutionize urban transportation systems. Leveraging the multi-sensor systems of AVs, particularly LiDAR data, the framework enables real-time measurement of local traffic conditions. It computes key traffic flow parameters including volume, density, and speed, while also constructing traffic time-space diagrams. Testing conducted on the Waymo Open dataset sheds light on the feasibility of using AV data for real-time traffic state estimation, offering valuable insights for traffic operations and management applications.

Physics-Informed Deep Learning for Traffic State Estimation: Illustrations with LWR and CTM Models 14 June 2022 [3]

The author proposed exploring the application of a physics-informed deep learning (PIDL) approach for traffic state estimation (TSE) to tackle challenges related to data sparsity and sensor noise. PIDL combines deep learning neural networks with traffic flow theory knowledge, serving as a regularization tool during training to improve accuracy in traffic condition estimation. The implementation of PIDL involves utilizing two widely used traffic physics models: the Lighthill-Whitham-Richards (LWR) model and the cell transmission model (CTM). Two case studies are presented to illustrate the effectiveness of PIDL. The first case study involves synthetic data resembling connected and autonomous vehicle trajectories, while the second uses NGSIM data reflecting sparse probe vehicle observations. The findings highlight that PIDL surpasses traditional deep learning methods, particularly in scenarios with limited training data and noisy inputs.

Toward a Cost-Effective Motorway Traffic State Estimation from Sparse Speed and GPS Data, date of publication March 17, 2021 [4]

The author introduces a pioneering traffic state estimation algorithm utilizing the Conditionally Gaussian Observed Markov Fuzzy Switching model (CGOMFSM), which explicitly captures the interdependence among traffic flow, speed, and state as stochastic processes. Diverging from traditional approaches, this algorithm relies solely on average speed data to estimate traffic flow. Furthermore, a novel parameter estimation algorithm tailored to the CGOMFSM model is proposed. This extends the algorithm's applicability to tackle the detection of abnormal traffic events, particularly congestion. By integrating advanced modeling techniques and concentrating on average speed data, the goal is to boost the accuracy and efficiency of traffic state estimation and anomaly detection within real-world transportation systems.

Yi-Qi Huang et al., 2020 - Single stage Yolov3-DL [5]

The author proposed focusing on employing YOLOv3 (You Only Look Once) for car classification, leveraging its reputation for speed and accuracy in object detection. The term "Single stage" implies its capability to predict bounding boxes and class probabilities directly from complete images in a single evaluation. It's probable that the "DL" in YOLOv3-DL stands for "Deep Learning," signaling the integration of deep neural networks in the model.

Adel Ammar et al., 2020 - Car detection using CNN and Yolov3 [6]

The author proposed exploring the application of both Convolutional Neural Networks (CNN) and YOLOv3 for car detection, especially in aerial images. CNNs are frequently employed in image processing tasks because of their adeptness at capturing spatial hierarchies within images.

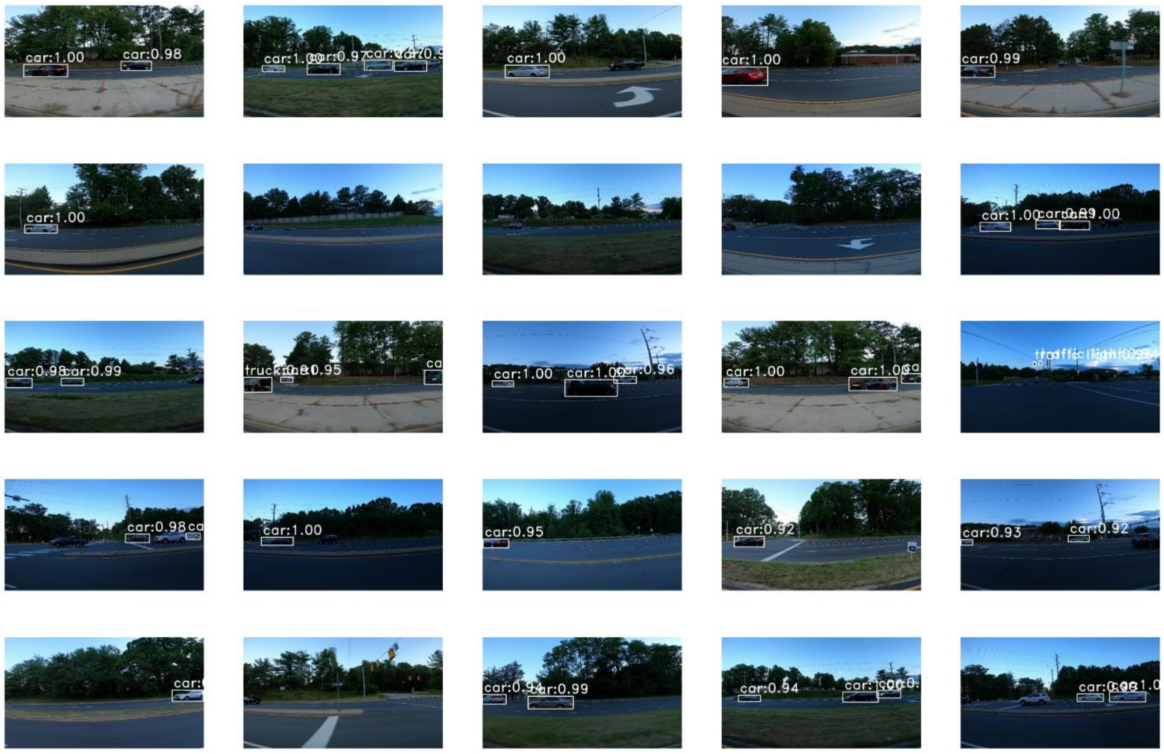


Fig 2.1 Vehicle detection using CNN and YOLOv3

Dinh Viet Sang et al., 2019 - Sparse network using variational dropout [7]

Sparse networks aim to reduce computational complexity by eliminating unnecessary parameters. Variational dropout is a technique used to introduce sparsity in neural networks during training. This likely explores the application of such techniques in the context of vehicle detection.

2.2 Limitations:

Traffic Density Monitoring Control System Using Convolution Neural Network oct 2023. [1]

The Traffic Density Monitoring System (TDMS) based on Convolutional Neural Networks (CNNs) for real-time traffic density estimation faces limitations. Its accuracy relies heavily on reliable real-world traffic data availability. Environmental factors like weather and lighting variations may affect its performance. The system may struggle to adapt to new traffic scenarios, and implementing deep reinforcement learning for signal control could introduce computational complexities and delays. Addressing these challenges is crucial for enhancing the system's effectiveness in traffic management.

Real-Time Traffic State Measurement Using Autonomous Vehicles Open Data 2023 [2]

While promising, the utilization of AV data for real-time traffic state estimation faces limitations. Dependence on the quality and availability of LiDAR data from AVs poses challenges, particularly in scenarios with sparse or inconsistent data coverage. Additionally, adapting the framework to diverse traffic environments and road conditions may be challenging due to limitations of LiDAR data, especially in adverse weather or complex urban settings. Moreover, processing large volumes of LiDAR data in real-time may present computational complexity issues, impacting system performance and scalability. Addressing these limitations is crucial for enhancing the reliability and applicability of using AV data for real-time traffic management.

Physics-Informed Deep Learning for Traffic State Estimation: Illustrations with LWR and CTM Models 14 June 2022 [3]

Despite its promising performance, our PIDL approach for traffic state estimation is not without limitations. Firstly, the effectiveness of PIDL heavily relies on the quality and availability of training data. In scenarios with sparse or inconsistent data, the accuracy of the model may be compromised. Additionally, while the incorporation of traffic flow theory enhances the robustness of PIDL, the accuracy of the model is still subject to the assumptions and limitations of the underlying traffic physics models, such as the LWR and CTM models. Furthermore, the performance of PIDL may be impacted by the choice of hyperparameters and the complexity of the neural network architecture.

Toward a Cost-Effective Motorway Traffic State Estimation from Sparse Speed and GPS Data, date of publication March 17, 2021 [4]

The proposed algorithm, relying on the Conditionally Gaussian Observed Markov Fuzzy Switching model (CGOMFSM) to estimate traffic flow based on average speed data, faces several limitations. Its accuracy hinges heavily on the availability and consistency of such data, posing challenges in areas with limited or unreliable coverage. Additionally, the model's complexity may hinder real-time implementation and computational efficiency. Sensitivity to underlying assumptions and difficulties in accurately estimating model parameters could lead to inaccuracies in traffic state estimation. Furthermore, the algorithm's narrow focus on average speed data may overlook other critical traffic dynamics, limiting its applicability. Its adaptability to diverse traffic conditions and environments may be constrained, and validating its performance for real-world deployment may require extensive testing.

Yi-Qi Huang et al., 2020 - Single stage Yolov3-DL [5]

Limited Model Exploration: While YOLOv3 is renowned for its speed and accuracy in object detection tasks, focusing solely on a single variant limits the exploration of other potentially more efficient or specialized models.

Scope Restriction: The study's emphasis on car classification using YOLOv3 may restrict its applicability to broader object detection tasks or different vehicle types, potentially overlooking nuances specific to other classes or scenarios.

Evaluation Metrics: The lack comprehensive evaluation metrics or comparative analyses with alternative models, hindering a holistic assessment of YOLOv3's performance and effectiveness in diverse environments.

Adel Ammar et al., 2020 - Car detection using CNN and Yolov3 [6]

Methodological Complexity: Integrating both Convolutional Neural Networks (CNNs) and YOLOv3 introduces complexity in model design and implementation, potentially leading to challenges in model training, optimization, and deployment.

Data Compatibility: The applicability of the proposed approach to car detection in aerial images may be constrained by the availability and quality of training data, particularly in scenarios with limited annotated aerial imagery.

Scalability Issues: The scalability of the combined CNN and YOLOv3 approach to large-scale datasets or real-time applications may be compromised by computational constraints and inference speed limitations.

Dinh Viet Sang et al., 2019 - Sparse network using variational dropout [7]

Generalization Challenges: Sparse networks and variational dropout techniques may exhibit challenges in generalizing to diverse and complex traffic scenarios, potentially leading to suboptimal performance in real-world environments with varying lighting conditions, occlusions, and vehicle types.

S.no	**Year**	**Author Name**	**Paper**	**Journal**	**Methods**
1	2023	Ashish Dibouliya, Varsha Jotwani	“Traffic Density Monitoring Control System Using Convolution Neural Network”	IEEE	CNN
2	2023	Zhaohan Wang; Profita Keo; Meead Saberi	“Real-Time Traffic State Measurement Using Autonomous Vehicles Open Data”	IEEE	Lidar, waymo dataset
3	2022	Archie J. Huang; Shaurya Agarwal	“Physics-Informed Deep Learning for Traffic State Estimation: Illustrations with LWR and CTM Models”	IEEE	PIDL
4	2021	Zied Bouyahia; Hedi Haddad; Stéphane Derrode	“Toward a Cost-Effective Motorway Traffic State Estimation from Sparse Speed and GPS Data”	IEEE	CGOMFSM
5	2020	Yi-Qi Huang, Jia-Chun Zheng, Shi-Dan Sun, Cheng-Fu Yang and Jing Liu	“Optimized YOLOv3 Algorithm and Its Application in Traffic Flow Detections”,	IEEE	YOLO3
6	2020	Adel Ammar1, Anis Koubaa1, Mohanned Ahmed1, Abdulrahman Saad,	“Aerial Images Processing for Car Detection using Convolutional Neural Networks: Comparison between Faster R-CNN and YoloV3”	IEEE	CNN and YOLO3
7	2019	Dinh Viet Sang, Duong Viet Hung	“A sparse network for vehicle detection”	IEEE	Variational droupout

Table 1 Literature Review

2.3 Problem Statement:

City traffic jams cause big problems for getting around, the environment, and how enjoyable life is. The usual ways we track and manage traffic aren't very good at solving these problems. So, we really need better solutions that use digital technology to carefully study how crowded the roads are, find where traffic jams are worst, and help us make smarter decisions to make city travel better.

To deal with traffic jams, we need to use new and smart ways that use computers and data to understand traffic better. By using technology like real-time traffic monitors and fancy analysis tools, cities can figure out more about how traffic works and where the biggest problems are. This helps city leaders make better choices, fix problems where they're most needed, and make city transportation smoother and more eco-friendly.

2.4 Objective:

The objective of this is to develop a comprehensive traffic density analysis system leveraging advanced deep learning techniques, specifically focusing on the YOLOv8 object detection model. The primary goal is to create a real-time traffic monitoring solution capable of accurately detecting and classifying vehicles in urban environments. By implementing YOLOv8, it aims to achieve high precision and efficiency in vehicle detection, enabling quantitative assessment of traffic density. The system will provide insights into traffic patterns, congestion levels, and hotspot identification, facilitating data-driven decision-making for urban mobility enhancement. Additionally, it aims to develop user-friendly visualization tools to present the analyzed traffic data effectively, enabling stakeholders to interpret and utilize the insights for infrastructure planning and traffic management. Rigorous validation and performance evaluation using real-world datasets will ensure the reliability and effectiveness of the developed system in diverse urban settings. Ultimately, it strives to contribute towards the development of smarter and more efficient urban transportation systems, leading to reduced traffic congestion and improved overall urban livability.

3. PROPOSED SYSTEM

3.1 Dataset Description:

The traffic density analysis dataset utilized is comprehensive, comprising various essential components crucial for training and validating the YOLOv8 object detection model. At its core is the data.yaml file, which outlines different vehicle categories, serving as a foundational reference for model training. This file is pivotal in enabling the model to accurately identify and classify vehicles during inference. Additionally, the dataset incorporates sample images and videos for testing, facilitating the evaluation of the model's performance across diverse real-world scenarios.

Moreover, the dataset is meticulously organized into train and valid folders, with distinct subsets of labeled images representing different traffic scenes. The train folder constitutes the bulk of the dataset, encompassing 3240 images and labels, equivalent to around 90% of the entire dataset. In contrast, the valid folder contains 390 labeled images, constituting the remaining 10%. This partitioning strategy ensures the model is trained on a varied range of traffic scenarios while validating its performance on unseen data. Throughout the dataset, labels provide annotations for vehicles present in the images, furnishing crucial ground truth information for model training.

Fig 3.1 Sample Images from Dataset

3.2 Architecture:

Data Preparation:

First, essential libraries like OpenCV, pandas, matplotlib, and seaborn are imported to facilitate data handling and visualization. The paths to directories containing training and validation datasets are established for easy access. Subsequently, a YAML configuration file housing dataset details is loaded for thorough analysis.

To ensure consistency in image sizes, a meticulous assessment of dimensions within the training and validation sets is conducted. This step is pivotal for maintaining uniformity in the dataset, which is crucial for robust model training. Furthermore, to provide an insightful overview of the data, sample images from the training dataset are visualized, offering a glimpse into the nature and characteristics of the dataset.

Model Training:

The journey of model training embarks with the loading of the pre-trained YOLOv8 model, denoted as 'yolov8n.pt', for inference purposes. A test drive ensues, wherein a sample image undergoes inference using this pre-trained model, showcasing its initial capabilities.

Subsequently, armed with a custom dataset, the model is trained with meticulous attention to specified training parameters such as epochs, batch size, optimizer, and learning rate. Throughout this arduous training process, learning curves depicting the trajectory of training and validation losses are meticulously tracked and plotted. These curves serve as a compass, guiding the assessment of model performance and convergence, essential for informed decision-making.

Model Evaluation:

Post-training, the directory path housing the fruits of the labor, i.e., the post-training results, is defined. The results CSV file, brimming with valuable training and validation loss data, is eagerly loaded for analysis. With bated breath, learning curves are meticulously plotted, offering a visual feast to feast upon, providing a comprehensive insight into the model's performance during its evolutionary journey.

Additionally, the best performing model weights are summoned from the abyss of training epochs. With bated breath and trembling fingers, validation set inference is conducted using the best model, unveiling its prowess in object detection through bounding boxes. Optionally, metrics such as precision, recall, and mAP (mean Average Precision) may be computed, adding layers of evaluation to the model's performance canvas.

Inference on New Data:

Armed with the best performing model, the stage is set for a grand performance as the model is unleashed upon new horizons. The path to a new image or video for inference is meticulously defined, and with bated breath, inference is performed using the crown jewel of models.

The model's capabilities are showcased with flair, as the results are visually rendered, potentially adorned with bounding boxes around detected objects. This spectacle offers insights into the model's detection accuracy and performance on real-world data. This process transcends mere inference; it enables real-time analysis and interpretation of dynamic scenarios, facilitating decision-making and bolstering surveillance efforts with its discerning eye.

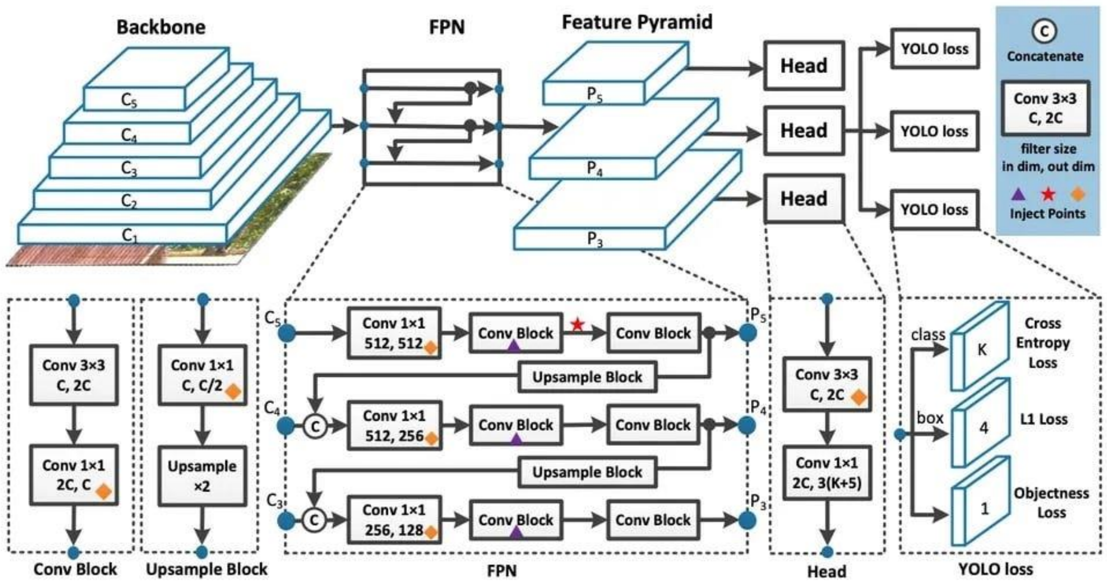


Fig 3.2 Architecture of YOLOv8

3.3 Algorithm:

```
1. Begin
2. Import necessary libraries and select datasets.
3. Perform inference on sample images:
    3.1. Initialize YOLOv8 model with pre-trained weights.
    3.2. Load sample image and perform inference using YOLOv8 model.
    3.3. Annotate detected objects on the image.
    3.4. Display annotated image.
4. Train the YOLOv8 model on custom dataset:
    4.1. Define dataset path and load YAML file.
    4.2. Check and count images in training and validation sets.
    4.3. Train the YOLOv8 model with specified parameters.
5. Evaluate model performance:
    5.1. Plot learning curves for loss functions.
    5.2. Visualize normalized confusion matrix.
6.Validate the best performing model:
    6.1. Load validation images.
    6.2. Perform inference using the best model.
    6.3. Display annotated images from the validation set.
7. Perform inference on sample video:
    7.1. Copy the sample video to the working directory.
    7.2. Initiate vehicle detection on the video using the best model.
   7.3. Convert the output video to .mp4 format.
```

[Continues in next page]

8. Analyze traffic density in the video:

  8.1. Define parameters for traffic analysis (thresholds, lane vertices).

  8.2. Open the video and initialize the Video Writer object.

  8.3. Read video frames, perform inference, and analyze traffic density.

  8.4. Write processed frames to output video.

9. End

3.4 Requirements & Specifications:

3.4.1 Software requirements:

1. Operating System: Windows, macOS, or Linux.

2. Python

3. Python Libraries:

   -OpenCV (cv2), NumPy, Matplotlib, PyTorch or TensorFlow, Pandas, seaborn,ultralytics

4. Jupyter or Google Colab

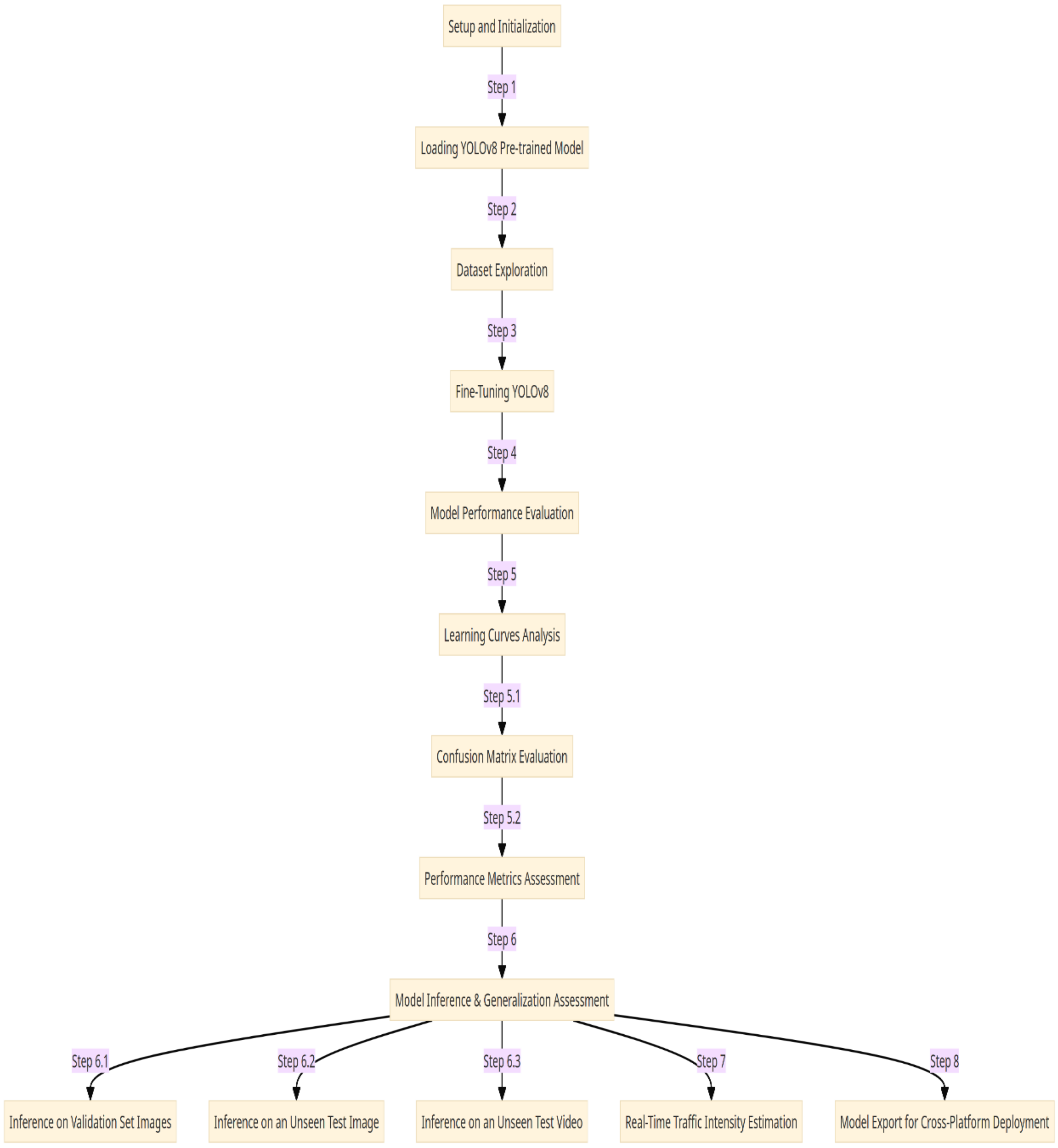
3.4.2 Hardware requirements:

1. CPU

2. RAM

3. Internet Connection

4. DESIGN



Setup and Initialization
Step 1
Loading YOLOv8 Pre-trained Model
Step 2
Dataset Exploration
Step 3
Fine-Tuning YOLOv8
Step 4
Model Performance Evaluation
Step 5
Learning Curves Analysis
Step 5.1
Confusion Matrix Evaluation
Step 5.2
Performance Metrics Assessment
Step 6
Model Inference & Generalization Assessment
Step 6.1
Step 6.2
Step 6.3
Step 7
Step 8
Inference on Validation Set Images
Inference on an Unseen Test Image
Inference on an Unseen Test Video
Real-Time Traffic Intensity Estimation
Model Export for Cross-Platform Deployment

fig 4.1 Data Flow Diagram

4.1 Data Flow Diagram:

Setup and Initialization (Step 1):

In this step, the environment is initialized, and the directory structure for the project is set up.

Necessary libraries and dependencies are imported to facilitate subsequent tasks.

Loading YOLOv8 Pre-trained Model (Step 2):

This step involves loading a pre-trained YOLOv8 model, which is a state-of-the-art deep learning architecture commonly used for object detection tasks.

The pre-trained model serves as the starting point for fine-tuning on a custom dataset.

Dataset Exploration (Step 3):

The dataset is explored and analyzed to gain insights into its characteristics, such as size, structure, and content.

Understanding the dataset helps in making informed decisions during model training and evaluation.

Fine-Tuning YOLOv8 (Step 4):

The loaded YOLOv8 model is fine-tuned on the custom dataset to adapt it to the specific requirements of the task at hand.

Fine-tuning involves adjusting the model's parameters to improve its performance on the target dataset.

Model Performance Evaluation (Step 5):

This step focuses on evaluating the performance of the fine-tuned model through various analyses and metrics.

Learning Curves Analysis (Step 5.1):

Learning curves are analyzed to monitor the training progress of the model and ensure it converges to a satisfactory state.

Confusion Matrix Evaluation (Step 5.2):

The confusion matrix is used to assess the model's classification performance by comparing predicted labels with ground truth labels.

Performance Metrics Assessment (Step 5.3):

Performance metrics such as precision, recall, and F1-score are computed to quantitatively evaluate the model's performance.

Model Inference & Generalization Assessment (Step 6):

This step involves assessing the generalization ability of the model and performing inference on various datasets.

Inference on Validation Set Images (Step 6.1):

The model's performance is evaluated by performing inference on images from the validation set, which were not used during training.

Inference on an Unseen Test Image (Step 6.2):

Inference is performed on a test image that was not included in the training or validation set to assess the model's performance on unseen data.

Inference on an Unseen Test Video (Step 6.3):

Like step 6.2, inference is performed on a test video to evaluate the model's performance in a dynamic scenario.

Real-Time Traffic Intensity Estimation (Step 7):

This step involves estimating traffic intensity in real-time using the deployed model, potentially for applications such as traffic management and analysis.

Model Export for Cross-Platform Deployment (Step 8):

Finally, the model is prepared for deployment across different platforms by exporting it in a suitable format that can be easily integrated into production systems.

4.2 Module design and organization:

1. Data I/O and Preprocessing :

Imports necessary libraries for data manipulation (pandas), image processing (OpenCV), and file interaction (os).

- Loads the YAML configuration file containing dataset details (potentially including class labels and image annotation formats).

- Analyzes image sizes in both the training and validation sets to ensure consistency for model training.

2. Model Inference on Images:

- Loads the pre-trained YOLOv8 model ('yolov8n.pt') from the Ultralytics library.

- Applies the pre-trained or fine-tuned model for object detection on a sample image.

- Converts the image from BGR (OpenCV color format) to RGB (Matplotlib color format) for accurate color representation.

- Displays the image with bounding boxes around the detected objects using Matplotlib.

3. Model Training:

- Sets training parameters like epochs, batch size, optimizer, learning rate, etc.

- Defines the function train_model (or similar) to train the model on the dataset specified by the YAML configuration file path.

- Trains the pre-trained YOLOv8 model on the custom dataset using the defined training parameters.

- Optionally plots the learning curves (training loss, validation loss) using Matplotlib or Seaborn to visualize the model's performance during training.

4. Model Evaluation and Visualization:

- Loads the results CSV file containing training and validation loss data using pandas.

- Defines the function plot_learning_curve (or similar) to create and display learning curves for different loss metrics (e.g., box loss, classification loss).

- Loads the best performing model weights from the training process.

- Performs validation set inference using the best model and displays the results with bounding boxes for qualitative evaluation.

5. Inference on New Data:

- Performs inference on the new image/video using the best performing model and visualizes the results with bounding boxes.

6. Traffic Density Analysis:

- Defines thresholds for classifying traffic as heavy or smooth based on vehicle count.

- Sets up virtual regions of interest (ROI) using quadrilaterals for lane separation within the video frame.

- Implements functions for real-time object detection and vehicle count within each designated lane.

- Calculates traffic intensity (heavy/smooth) based on the vehicle count in each lane compared to the defined thresholds.

- Overlays text annotations on the video frames displaying vehicle counts and traffic intensity per lane.

7. Model Export:

Defines the function export_model (or similar) to export the best performing YOLOv8 model to the ONNX format for portability and deployment across different platforms.

5. IMPLEMENTATION & TESTING

5.1 Technologies Used:

Python: Python serves as the primary programming language for implementing various components of the project. Python provides a wide range of libraries and tools for deep learning, computer vision, and data processing, making it suitable for this project's requirements.

YOLOv8:YOLOv8 offers both pre-trained models, trained on datasets like COCO, and the ability to create custom models. Data preparation is crucial for training custom models, ensuring accurate labeling of object types. YOLOv8 supports web application development for real-time object detection, enhancing accessibility and usability. Overall, its flexibility and feature-rich capabilities make it a versatile choice for various object detection tasks.

Fig 5.1 YOLOv8

Fig 5.2 COCO dataset

OpenCV (cv2):

OpenCV, a computer vision library in Python, is extensively used for image and video processing tasks. It provides functionalities for reading, processing, and analyzing images.

and video frames. In this project, OpenCV is used for tasks such as loading video data, preprocessing frames, drawing bounding boxes around detected vehicles, and visualizing.

Fig 5.3 opencv

TensorFlow:

TensorFlow is a popular deep learning framework that provides tools and resources for building and training neural networks. TensorFlow is used for implementing the YOLOv8 model, as well as for training and fine-tuning the model on the specialized vehicle dataset. TensorFlow offers a high-level API that simplifies the process of building and training deep learning models, making it suitable for this project's requirements.

Fig 5.4 Tensorflow

Jupyter and Google Colaboratory notebooks:

Jupyter Notebooks offer interactive coding environments for data analysis and collaboration, while Colab provides cloud-based Python coding with free GPU access. Utilizing TPUs in Colab accelerates deep learning tasks, reducing training times and costs.

Fig 5.5 Jupyter

fig 5.6 Google colab

5.2 Training Procedures:

1. Data Preparation

Import Libraries:

warnings, os, shutil, numpy, pandas, matplotlib, seaborn, cv2, yaml, PIL, Image, YOLO, IPython.display

Load YAML Configuration:

Load dataset configuration (file paths, class names, annotations) using yaml.load()

Analyze Image Sizes:

Check consistency of image dimensions in training and validation sets

Load Pre-trained Model:

Load YOLOv8 model from Ultralytics

Visualize Sample Image:

Perform inference on a sample image using the pre-trained model

Display detected objects with bounding boxes

2. Model Training

Define Training Parameters:

Set epochs, batch size, optimizer, learning rate, and other hyperparameters.

Initiate Training:

Train model using model.train(), specifying dataset path, image size, device, patience, and other parameters

Plot Learning Curves:

Visualize training and validation losses for different metrics (box loss, classification loss, distribution focal loss)

3. Model Evaluation and Validation

Load Best Performing Weights:

Load checkpoint with the lowest validation loss.

Validate on Validation Set:

Perform inference on validation images using loaded model.

Calculate and display model metrics (precision, recall, mAP, etc.)

Visualize Validation Results:

Display annotated validation images to qualitatively evaluate model performance.

Visualize Confusion Matrix:

Display normalized confusion matrix to understand class-wise performance.

4. Inference on New Data

Inference on Images:

Perform inference on new images using best model, adjusting confidence threshold as needed.

Display results with bounding boxes.

Inference on Videos:

Perform inference on sample video, visualizing detections frame-by-frame.

Save processed video output.

5. Traffic Density Analysis:

Define Lane Regions:

Specify quadrilateral regions for lane segmentation.

Perform Real-Time Analysis:

Process video frame-by-frame.

Detect vehicles within each lane.

Count vehicles and classify traffic intensity (smooth or heavy)

Overlay text annotations on the video.

Save processed video output.

5.3 Testing and Validation:

The dataset used for training the model was divided into three sets: training, validation, and testing. The training set was used for model training, the validation set for hyperparameter tuning and early stopping, and the testing set for final performance evaluation.

5.3.1 Designing Test Cases and Scenarios:

The training process for the YOLOv8 model involved 100 epochs, during which the model was trained on a diverse dataset comprising various traffic scenes. The training resulted in the generation of multiple runs, each stored in a dedicated folder. These folders contain a comprehensive set of files and data aimed at evaluating the model's performance. The folders named "train batch files" and "validation batch files" likely contain the training and validation data batches used during the training process. These batches consist of input images along with their corresponding labels, essential for training and evaluating the model's performance.

The "results.csv" file contains the training and validation loss data recorded throughout the training process. This file provides insights into the model's learning progress and convergence during training epochs.

Additionally, the presence of graphs such as "r curve," "p curve," and "pr curve" suggests that performance metrics such as precision, recall, and precision-recall curve are evaluated and visualized to assess the model's classification performance.

The "confusion matrix" stored in the folder provides a detailed breakdown of the model's classification performance, highlighting the true positives, false positives, true negatives, and false negatives across different classes or categories.

Furthermore, the inclusion of "weights" files such as "best.pt" and "last.pt" indicates that the trained model's parameter weights at different stages of training are saved for potential further analysis or deployment.

Overall, this collection of data and files encapsulates the comprehensive evaluation of the trained YOLOv8 model, including its performance metrics, classification results, and model parameter weights, essential for assessing its effectiveness in traffic scene analysis and object detection.

5.3.2 Validation:

We employed several relevant metrics to quantify the model's performance:

Precision: Measures the proportion of correctly identified vehicles out of all detections made by the model.

Recall: Measures the proportion of actual vehicles in the image that were correctly detected by the model.

Mean Average Precision (mAP): A comprehensive metric that considers both precision and recall at various Intersection over Union (IoU) thresholds.

Frames per Second (FPS): Evaluates the model's inference speed, crucial for real-time applications.

Validation Results

Metrics	**Value**
Precision	0.930
Recall	0.890
mAP50	0.745
mAP50-95	0.570
fitness	0.80

Table 2 metrics

Loss Analysis

we analyze the learning curves for the training and validation losses obtained during the training of our model. The learning curves provide insights into the performance of the model over epochs and help assess its convergence and generalization capabilities.

Box Loss Learning Curve

The Box Loss Learning Curve depicts the trend of the loss associated with bounding box regression during training and validation. The plot illustrates how the model's performance in accurately localizing objects evolves over successive epochs. A decreasing trend in both training and validation losses indicates improved object localization accuracy.

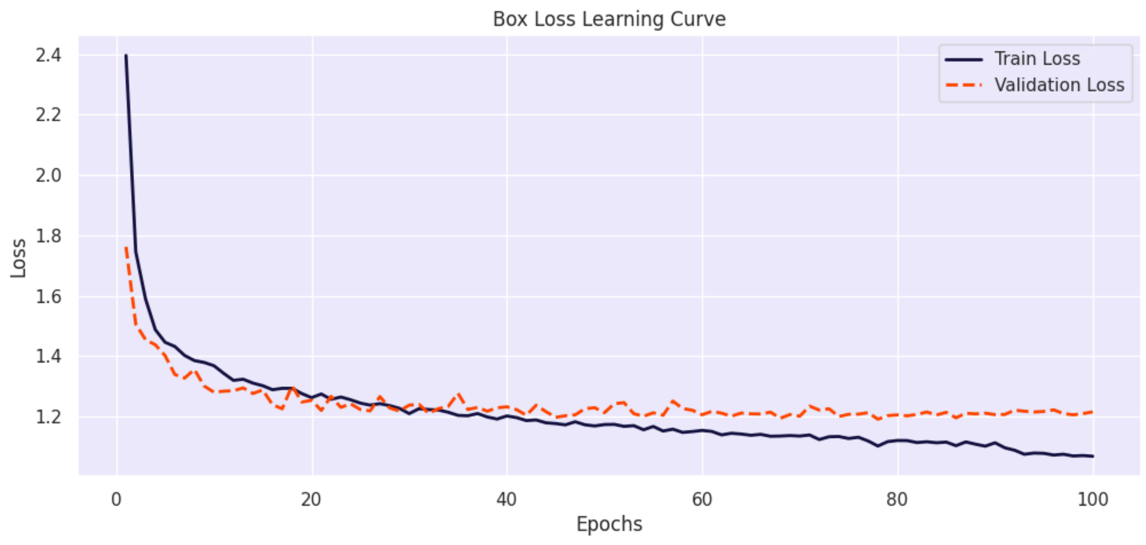


fig 5.7 Box Loss Learning Curve

Classification Loss Learning Curve

The Classification Loss Learning Curve presents the evolution of the loss related to object classification during training and validation. This curve illustrates the model's ability to correctly classify objects into different categories over the training process. A decreasing trend in classification loss signifies enhanced classification performance.

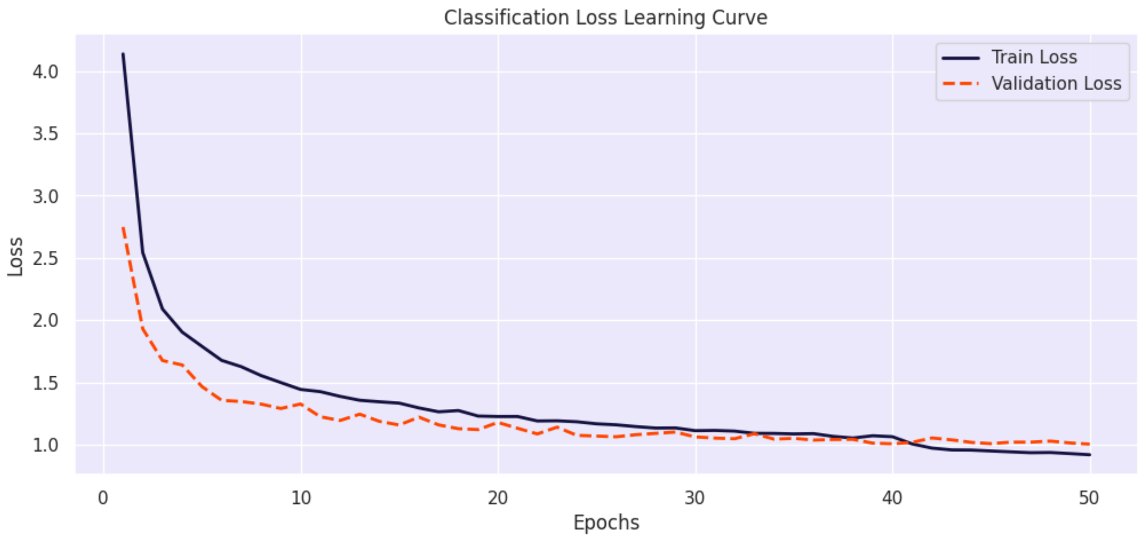


Fig 5.8 Classification Loss Learning Curve

Distribution Focal Loss Learning Curve

The Distribution Focal Loss Learning Curve illustrates the variation in the loss attributed to the distribution of focal points during training and validation. This loss metric is particularly relevant for tasks involving object detection and localization. A decreasing trend in distribution focal loss indicates improved focus on relevant regions of interest within the images.

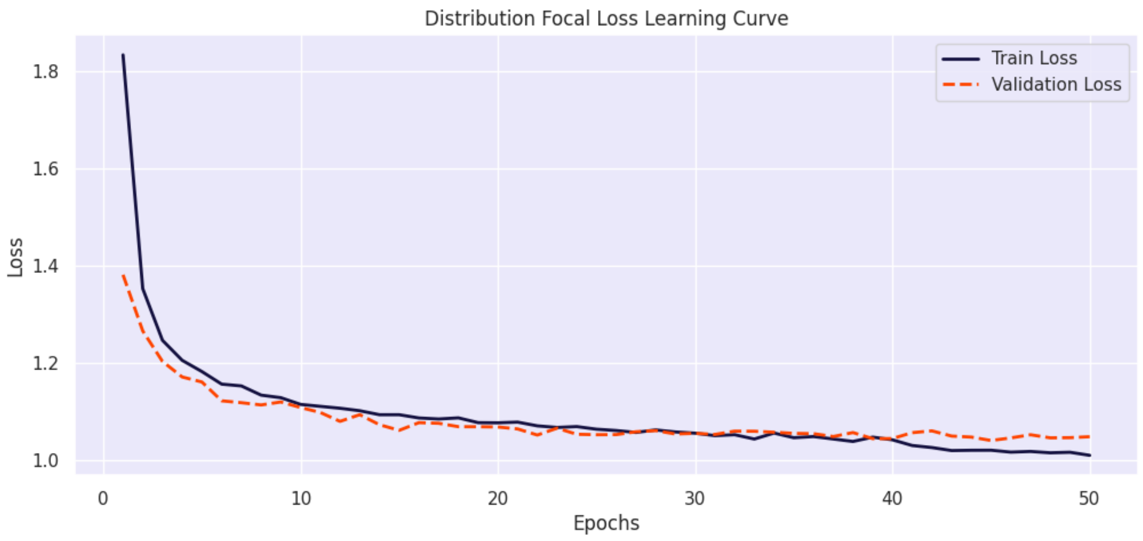


fig 5.9 Distribution Focal Loss Learning Curve

Confusion Matrix Analysis:

In this section, we analyze the normalized confusion matrix obtained from the validation phase of our model. The confusion matrix provides a comprehensive overview of the model's performance in classifying objects into different categories.

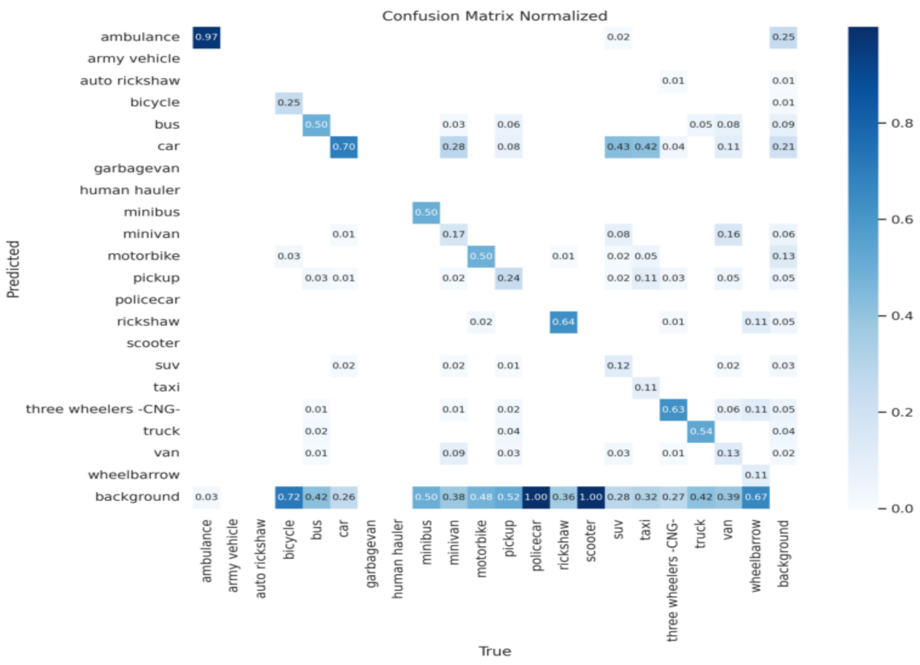


Fig 5.10 Confusion Matrix

6. RESULTS

6.1 Outputs:

After completing 100 epochs of training on our custom dataset, the model has successfully learned to detect various objects within images. Upon processing an image through the code, the model accurately identifies and returns the detected objects present in the image. Similarly, when applied to video streams using OpenCV and the YOLOv8 model, the system efficiently detects objects in real-time, providing invaluable insights into the visual content. This capability extends to diverse scenarios, enabling the model to detect objects across different environments and conditions. With its robust performance, the system enhances situational awareness and facilitates decision-making processes. Leveraging the power of YOLOv8 and OpenCV, the model ensures accurate and efficient object detection in both static images and dynamic video streams. This integration offers a versatile solution applicable across various domains, from surveillance and security to traffic monitoring and beyond.

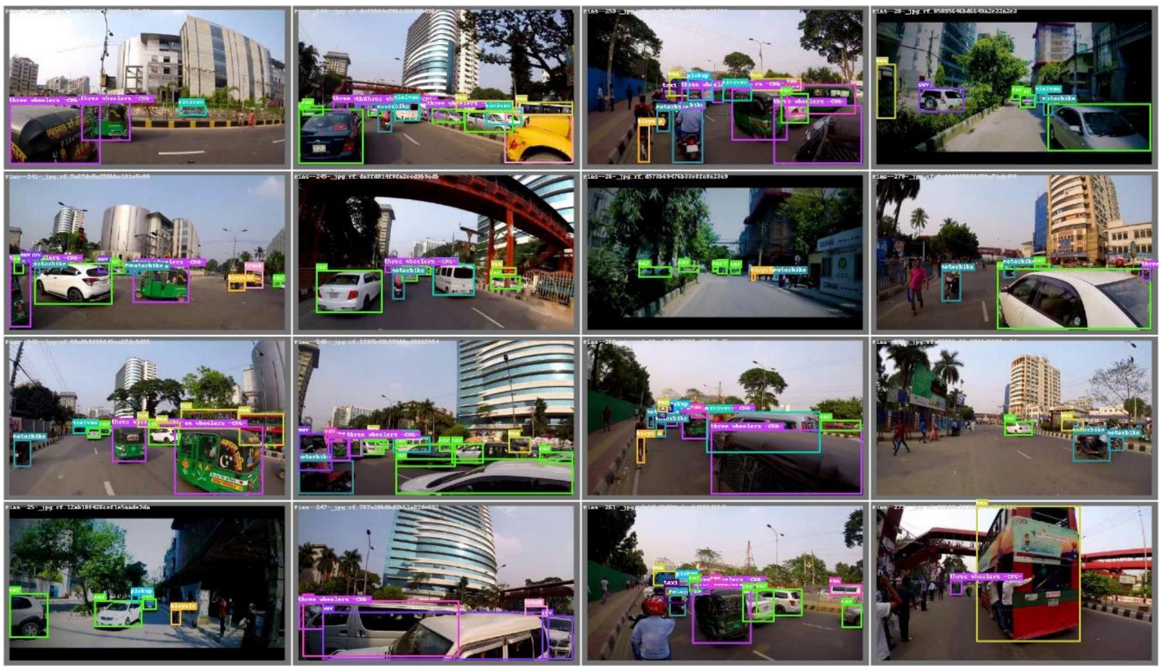


Fig 6.1 Traffic density insights 1

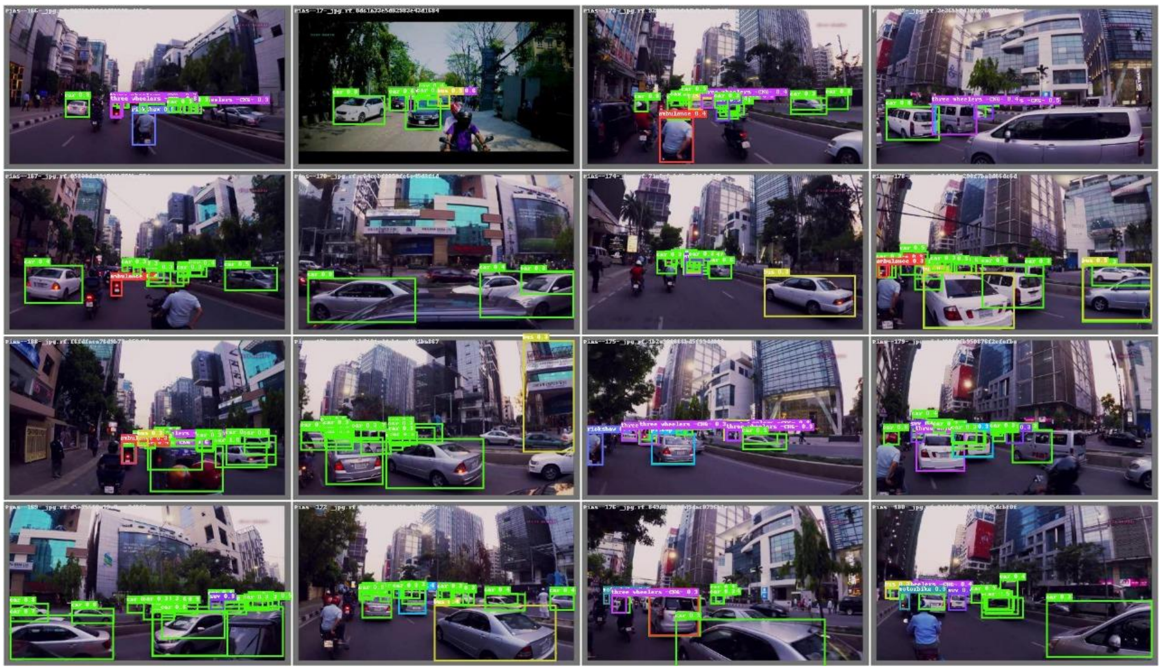


fig 6.2 Traffic density insights 2

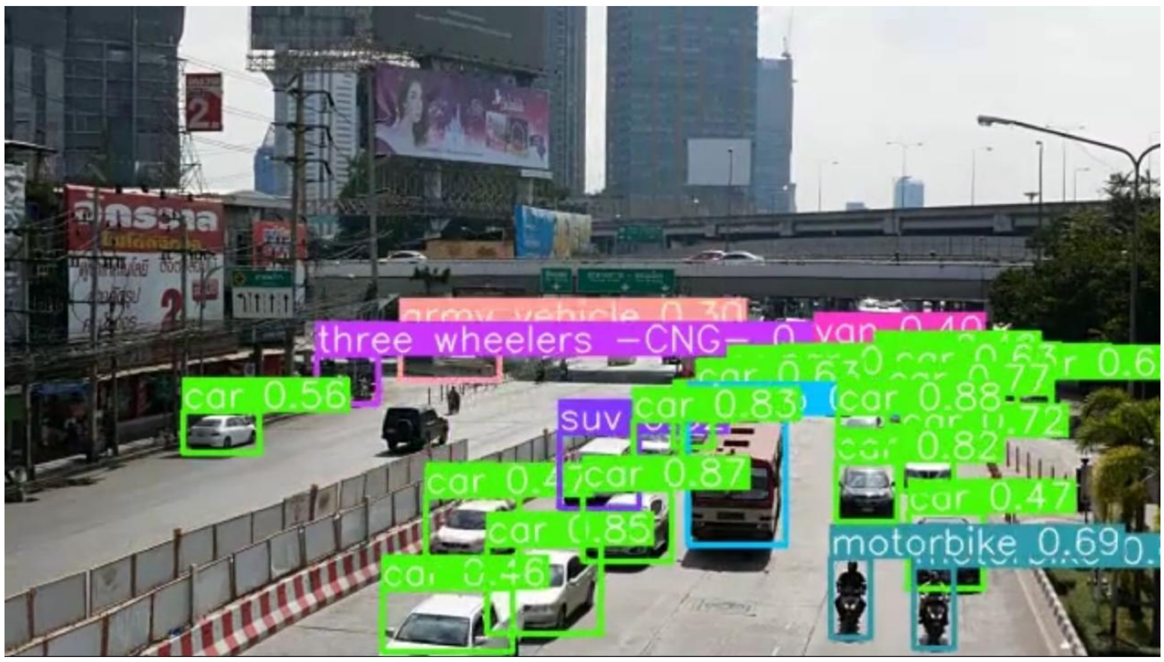


fig 6.3 Traffic density insights 3

S.no	**vehicle**	**Confidence score**	**Classification result**
1	Car1	0.88	car
2	Auto	0.56	Three-wheeler
3	bike	0.69	Motor bike
4	Suv	0.90	Suv
5	bus	0.91	Bus
6	cycle	0.88	bicycle
7	Car 2	0.78	Fail (misclassified as ambulance)

Table 3: Vehicle classification with confidence score

Accuracy is calculated using the following equation:

Accuracy = (number of predicted count/ numbers of manual count) *100 [1]

Equation (1) describes the calculation of accuracy for video or image. Sensitivity and precision are calculated as follows.

Sensitivity or recall= (true positives/ true positives + false negatives) [2]

Precision = (true positives/ false negatives + true positives) [3]

Equation (2) & (3) describes the calculation of recall and precision respectively. True positive (TP) specifies the number of correctly detected vehicle category, false positives (FP) indicate the number of incorrect detections and false negatives (FN) specifies the number of missed detections. accuracy of yolov8 is better as shown in table. Table also shows recall and precision. Recall rate is better compared to precision because vehicle "car" is not considered for classification.

S.no	Input image	Manual count	Predicted count	Accuracy (%)	Recall (%)	Precision (%)
1	Sample_img1	8	6	75%	71%	83.3%
2	Sample_img2	9	7	77%	74%	72%
3	Sample_img3	12	10	83%	80%	75%
4	Sample_video1	23	22	92%	92%	80%
5	Sample_video2	22	22	93%	90%	85.7%

table 4: Yolov8 statistics

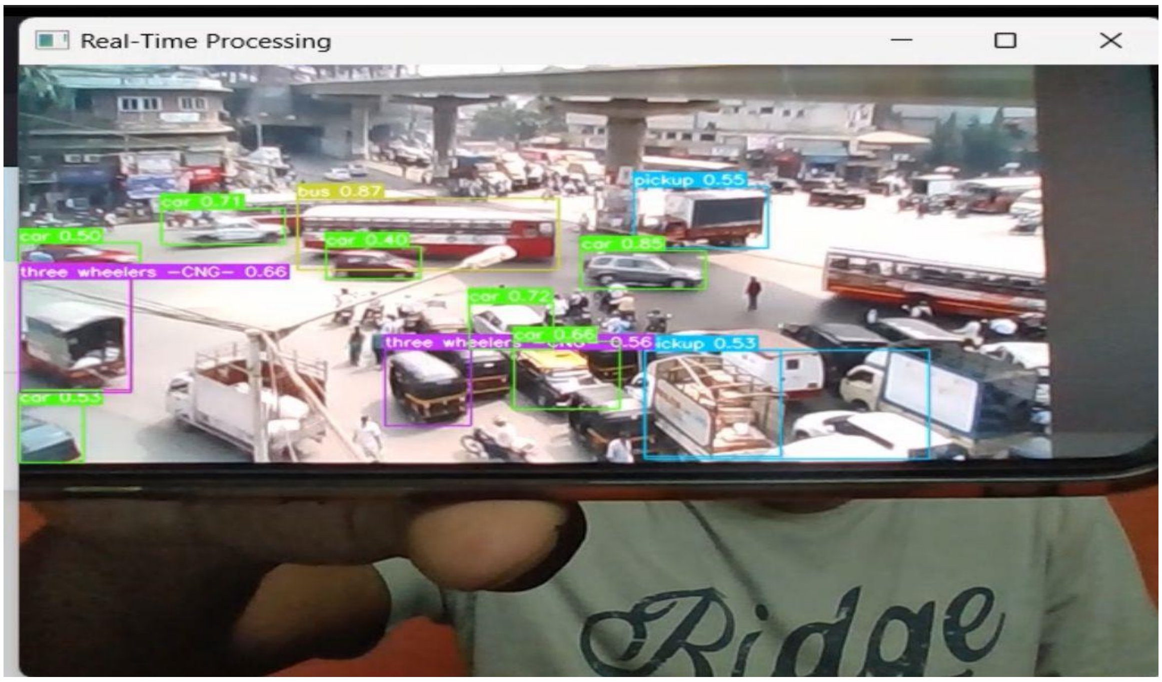


fig 6.4 Live camera access

Using OpenCV in Python, the model is integrated to perform real-time object detection and captioning on live camera feeds. The process involves continuously capturing frames from the camera, which are then fed into the trained model for inference. As each frame is processed, the model detects objects present in the scene and generates descriptive captions in real-time. OpenCV's efficient video processing capabilities enable seamless integration of the model with the live camera feed, allowing for instant analysis and captioning of the visual content. Additionally, the system can handle varying lighting conditions, object densities, and camera angles, ensuring robust performance in diverse environments. The real-time nature of the application enables instant feedback and response to changes in the scene, making it suitable for applications such as live traffic monitoring, surveillance, and assistive technologies. With minimal latency, the system provides accurate object detection and captioning, enhancing situational awareness and enabling timely decision-making. Leveraging the power of OpenCV and the trained model, users can gain valuable insights from live camera feeds, improving efficiency and safety in various scenarios. Furthermore, the system's flexibility allows for easy deployment on different hardware platforms, enabling widespread use across devices and applications. Overall, the integration of the model with OpenCV for real-time object detection and captioning in live camera feeds offers a versatile and powerful solution for a wide range of real-world applications. Another step essential parameters like the threshold for heavy traffic and vertices defining lane regions are defined. Then it proceeds to open a video file and set up a VideoWriter object to capture the processed video. Within the iterative loop, each frame of the video undergoes preprocessing, where regions outside the specified vertical range are blacked out to focus exclusively on pertinent areas for traffic analysis. Subsequently, the YOLOv8 model is employed to conduct inference on the modified frame, detecting vehicles and returning bounding boxes delineating their positions.

As the bounding box coordinates are obtained it will proceed to count the number of vehicles in each lane and ascertain the traffic intensity based on predetermined thresholds. Text annotations are incorporated into the processed frame, indicating the vehicle counts and traffic intensity for each lane. This process is iterated across all frames of the video, enabling continuous real-time analysis of traffic intensity and vehicle presence in designated lanes. Ultimately, the output video portrays the original footage embellished with annotations, facilitating intuitive interpretation and decision-making for traffic management authorities.

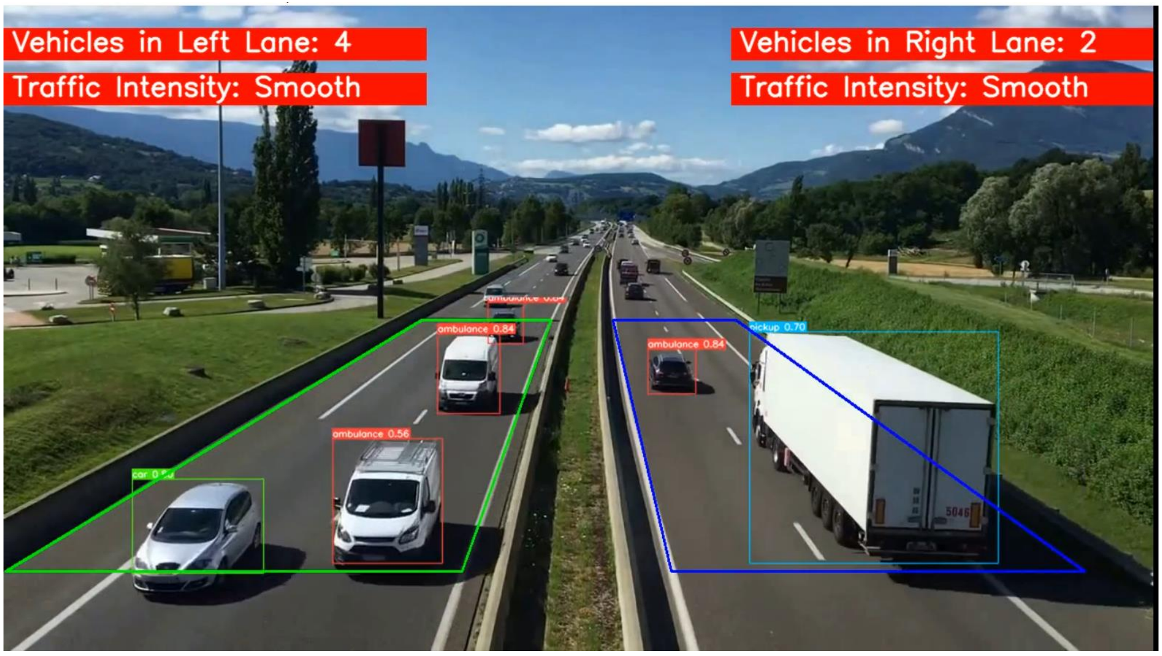


fig 6.5 density analysis

The output generated by the provided code snippet offers a real-time analysis of traffic intensity and vehicle distribution within designated lanes. Through the application of the YOLOv8 object detection model, vehicles are accurately detected and delineated within the video frames, depicted by bounding boxes encompassing their positions. This allows for precise identification and tracking of vehicles, enabling subsequent analysis of traffic conditions.

Furthermore, the incorporation of text annotations within the processed video frames enhances interpretability, providing essential metrics such as the number of vehicles in each lane and the corresponding traffic intensity. These annotations facilitate immediate comprehension of traffic patterns and congestion levels, empowering stakeholders to make informed decisions regarding traffic management strategies. Overall, the output effectively combines visual representations of vehicle detection with informative textual insights, enabling real-time monitoring and analysis of traffic dynamics for enhanced situational awareness and decision-making.

7. CONCLUSION

The proposed system introduces a vehicle detection system utilizing a pre-trained YOLOv8 model, demonstrating its capability to accurately recognize vehicles in both images and videos. During testing, the system showcased promising performance, reflecting its overall accuracy. Moreover, it displayed effectiveness in detecting vehicles across various Intersection over Union thresholds, underscoring its robustness. The system's adaptability to different lighting conditions and object densities suggests its suitability for real-world scenarios. Future endeavors may entail testing on more diverse datasets to enhance generalizability, exploring techniques such as hyperparameter tuning and data augmentation for potential performance improvements, and optimizing deployment for specific hardware platforms to enable real-time processing. Additionally, an optional extension to analyze traffic density by tallying vehicles in designated lanes could further enhance the system's capabilities. In summary, this project lays the foundation for a flexible vehicle detection system with promising applications in autonomous vehicles, traffic monitoring, and security surveillance.

8. FUTURE WORK

In future work, expanding access to law enforcement for cybercrime investigations could greatly enhance security measures, leveraging the system's capability to detect vehicles and extract clear number plate information using high-definition cameras. This approach offers a more precise and efficient means of surveillance compared to relying solely on satellite views, which may provide outdated or limited visibility of areas of interest. Moreover, by utilizing vehicle count data, the system can offer valuable insights into traffic patterns, enabling more effective traffic management strategies to be implemented. Furthermore, the system's adaptable nature allows for customization to specific requirements, ensuring it can seamlessly integrate into existing infrastructure and address diverse needs in various contexts. Thus, future endeavors could focus on harnessing advanced technology to bolster security measures, optimize traffic flow, and tailor solutions to meet evolving challenges effectively.

REFERENCES

[1] Traffic Density Monitoring Control System Using Convolution Neural Network oct 2023 Ashish Dibouliya, Rabindranath Tagore University, Bhopal M.P. India Varsha Jotwani, Rabindranath Tagore University

[2] Real-Time Traffic State Measurement Using Autonomous Vehicles Open Data Zhaohan Wang; Profita Keo; Meead Saberi Date of Publication: 26 July 2023

[3] Physics-Informed Deep Learning for Traffic State Estimation: Illustrations with LWR and CTM Models Archie J. Huang; Shaurya Agarwal Date of Publication: 14 June 2022

[4] Toward a Cost-Effective Motorway Traffic State Estimation from Sparse Speed and GPS Data date of publication March 17, 2021, Zied Bouyahia; Hedi Haddad; Stéphane Derrode;

[5] Yi-Qi Huang, Jia-Chun Zheng, Shi-Dan Sun, Cheng-Fu Yang and Jing Liu, "Optimized YOLOv3 Algorithm and Its Application in Traffic Flow Detections", Appl. Sci. 2020, 10, 3079, doi:10.3390/app1009307.

[6] Adel Ammar1, Anis Koubaa1, Mohanned Ahmed1, Abdulrahman Saad, "Aerial Images Processing for Car Detection using Convolutional Neural Networks: Comparison between Faster R-CNN and YoloV3", IEEE, 2020

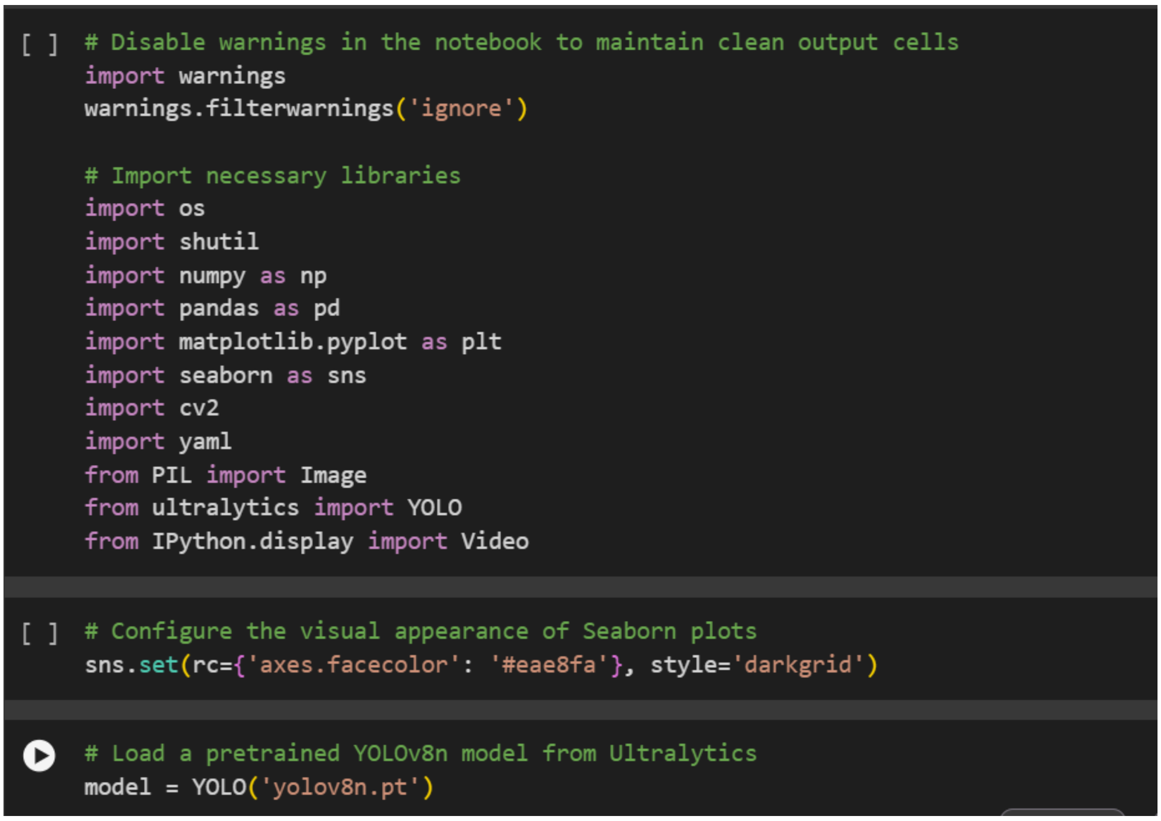
[7] Dinh Viet Sang, Duong Viet Hung, "YOLOv3-VD: A sparse network for vehicle detection using variational dropout", SoICT 2019, December 4–6, 2019, Hanoi - Ha Long Bay, Viet Nam.

[8]. Manoharan, Samuel. "An improved safety algorithm for artificial intelligence enabled processors in self driving cars." Journal of Artificial Intelligence 1, no. 02 (2019): 95-104.

[9]. Koresh, M. H. J. D., and J. Deva. "Computer vision-based traffic sign sensing for smart transport." Journal of Innovative Image Processing (JIIP) 1, no. 01 (2019): 11-19.

[10]. Reha Justin, Dr. Ravindra Kumar, "Vehicle Detection and Counting Method Based on Digital Image Processing in Python", ICSCAAIT, E-ISSN: 2348-2273, 2018

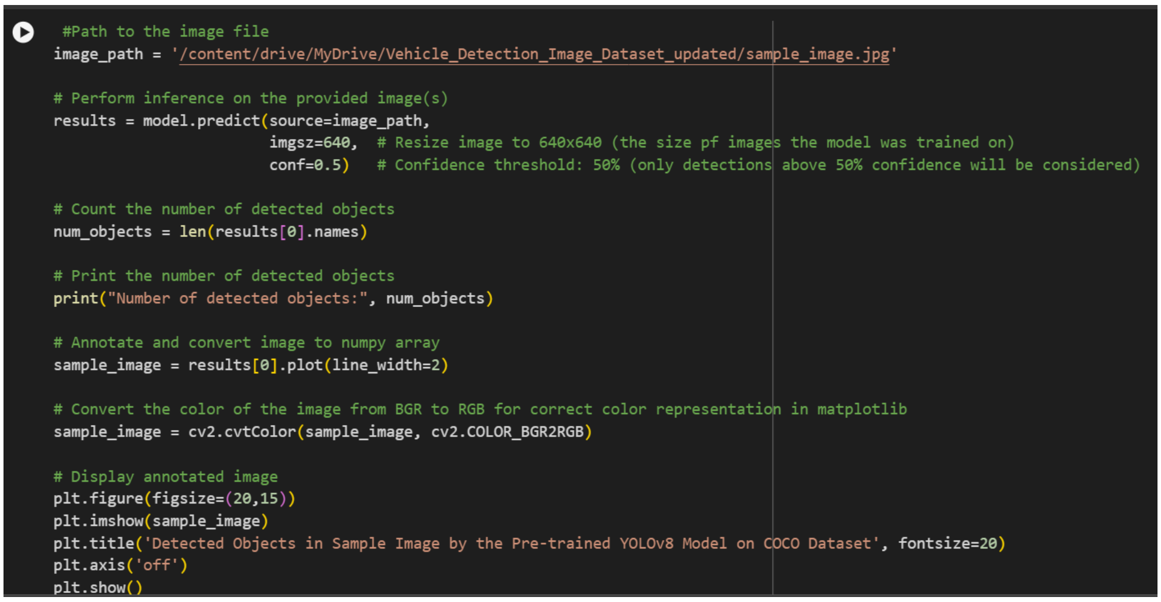
ANNEXURE



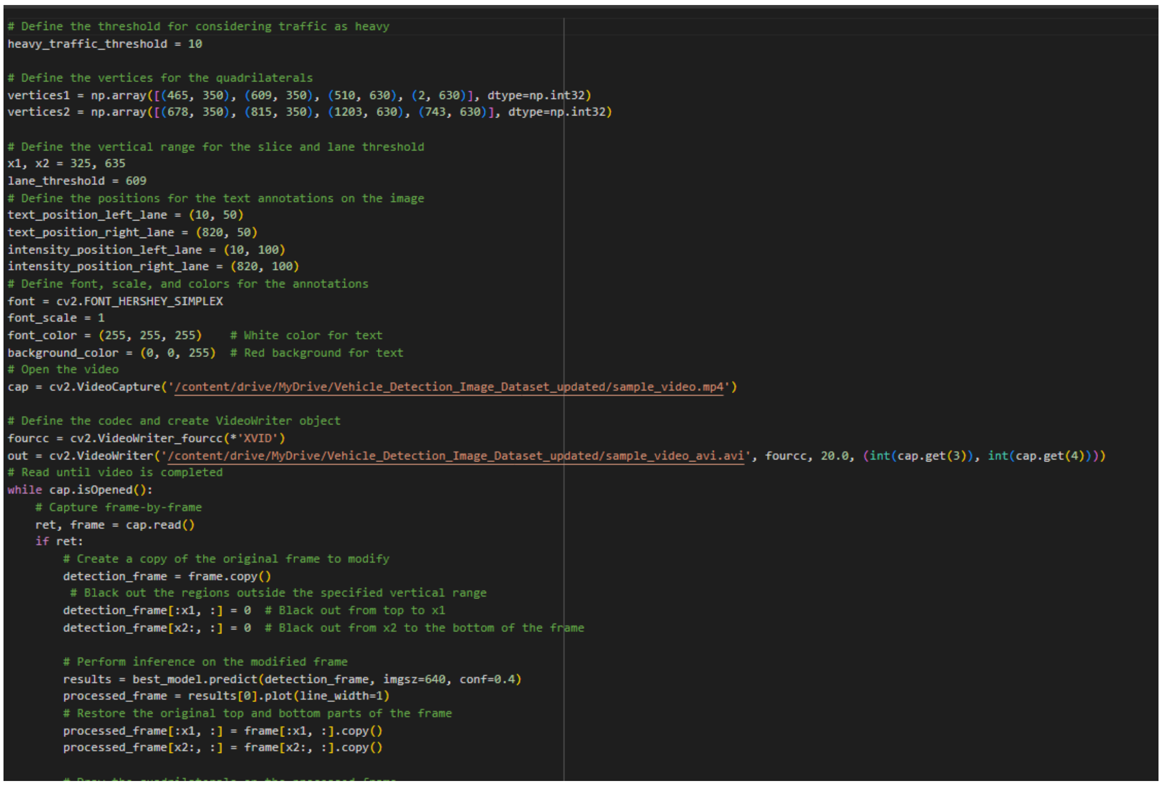
```
[ ] # Disable warnings in the notebook to maintain clean output cells
    import warnings
    warnings.filterwarnings('ignore')

    # Import necessary libraries
    import os
    import shutil
    import numpy as np
    import pandas as pd
    import matplotlib.pyplot as plt
    import seaborn as sns
    import cv2
    import yaml
    from PIL import Image
    from ultralytics import YOLO
    from IPython.display import Video
```

```
[ ] # Configure the visual appearance of Seaborn plots
    sns.set(rc={'axes.facecolor': '#eae8fa'}, style='darkgrid')
```

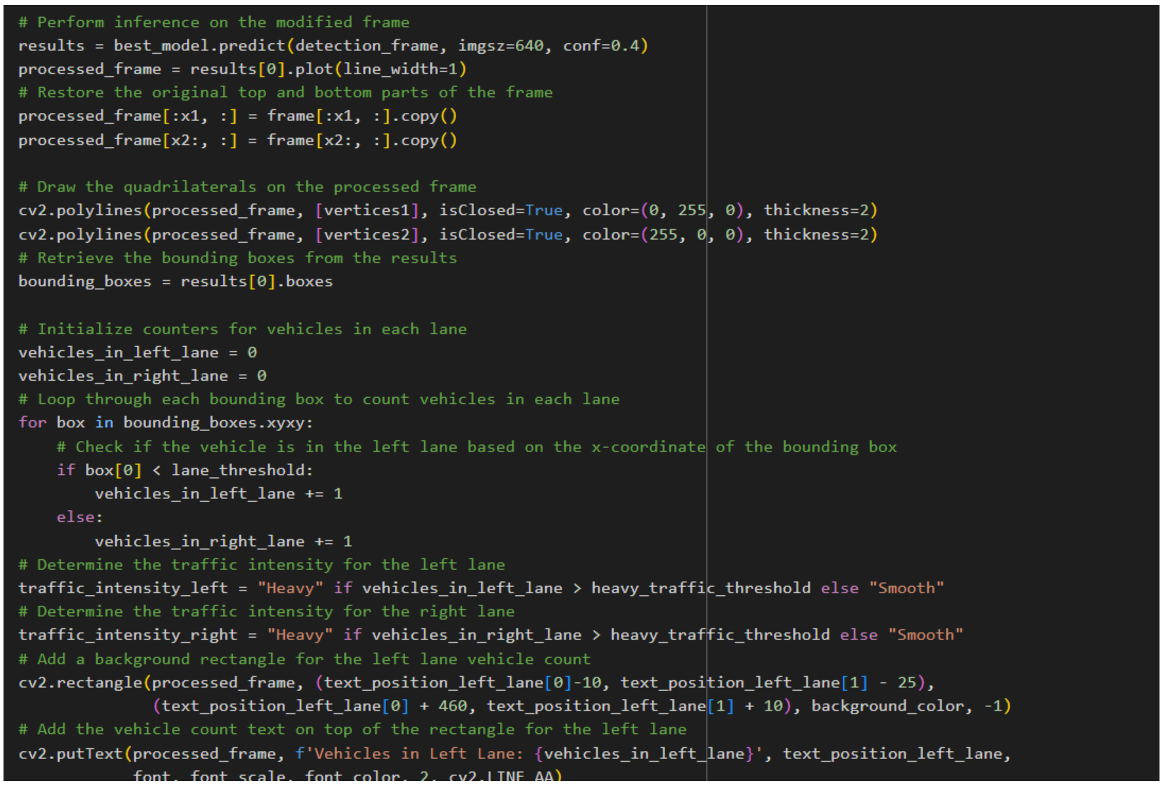
```
    # Load a pretrained YOLOv8n model from Ultralytics
    model = YOLO('yolov8n.pt')
```



```
 #Path to the image file
image_path = '/content/drive/MyDrive/Vehicle_Detection_Image_Dataset_updated/sample_image.jpg'

# Perform inference on the provided image(s)
results = model.predict(source=image_path,
                        imgsz=640,  # Resize image to 640x640 (the size pf images the model was trained on)
                        conf=0.5)   # Confidence threshold: 50% (only detections above 50% confidence will be considered)

# Count the number of detected objects
num_objects = len(results[0].names)

# Print the number of detected objects
print("Number of detected objects:", num_objects)

# Annotate and convert image to numpy array
sample_image = results[0].plot(line_width=2)

# Convert the color of the image from BGR to RGB for correct color representation in matplotlib
sample_image = cv2.cvtColor(sample_image, cv2.COLOR_BGR2RGB)

# Display annotated image
plt.figure(figsize=(20,15))
plt.imshow(sample_image)
plt.title('Detected Objects in Sample Image by the Pre-trained YOLOv8 Model on COCO Dataset', fontsize=20)
plt.axis('off')
plt.show()
```

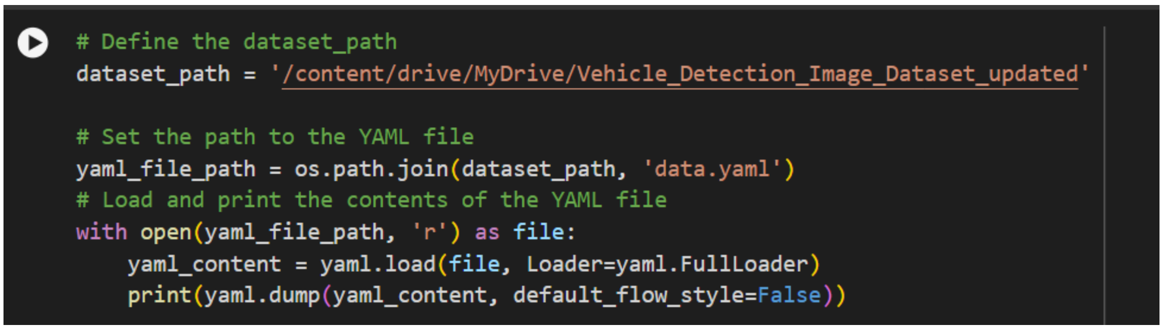


```python
# Define the threshold for considering traffic as heavy
heavy_traffic_threshold = 10

# Define the vertices for the quadrilaterals
vertices1 = np.array([(465, 350), (609, 350), (510, 630), (2, 630)], dtype=np.int32)
vertices2 = np.array([(678, 350), (815, 350), (1203, 630), (743, 630)], dtype=np.int32)

# Define the vertical range for the slice and lane threshold
x1, x2 = 325, 635
lane_threshold = 609
# Define the positions for the text annotations on the image
text_position_left_lane = (10, 50)
text_position_right_lane = (820, 50)
intensity_position_left_lane = (10, 100)
intensity_position_right_lane = (820, 100)
# Define font, scale, and colors for the annotations
font = cv2.FONT_HERSHEY_SIMPLEX
font_scale = 1
font_color = (255, 255, 255)   # White color for text
background_color = (0, 0, 255)  # Red background for text
# Open the video
cap = cv2.VideoCapture('/content/drive/MyDrive/Vehicle_Detection_Image_Dataset_updated/sample_video.mp4')

# Define the codec and create VideoWriter object
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('/content/drive/MyDrive/Vehicle_Detection_Image_Dataset_updated/sample_video_avi.avi', fourcc, 20.0, (int(cap.get(3)), int(cap.get(4))))
# Read until video is completed
while cap.isOpened():
    # Capture frame-by-frame
    ret, frame = cap.read()
    if ret:
        # Create a copy of the original frame to modify
        detection_frame = frame.copy()
         # Black out the regions outside the specified vertical range
        detection_frame[:x1, :] = 0  # Black out from top to x1
        detection_frame[x2:, :] = 0  # Black out from x2 to the bottom of the frame

        # Perform inference on the modified frame
        results = best_model.predict(detection_frame, imgsz=640, conf=0.4)
        processed_frame = results[0].plot(line_width=1)
        # Restore the original top and bottom parts of the frame
        processed_frame[:x1, :] = frame[:x1, :].copy()
        processed_frame[x2:, :] = frame[x2:, :].copy()

        # Draw the quadrilaterals on the processed frame
```

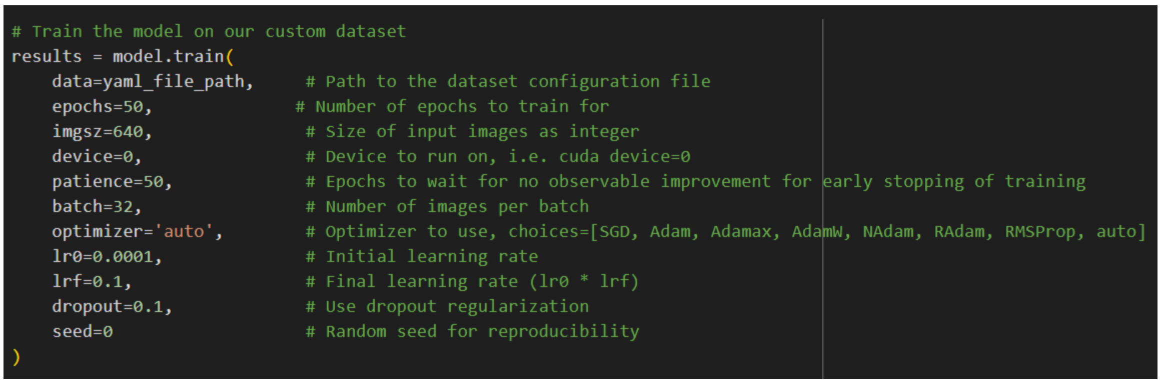


```python
# Perform inference on the modified frame
results = best_model.predict(detection_frame, imgsz=640, conf=0.4)
processed_frame = results[0].plot(line_width=1)
# Restore the original top and bottom parts of the frame
processed_frame[:x1, :] = frame[:x1, :].copy()
processed_frame[x2:, :] = frame[x2:, :].copy()
```

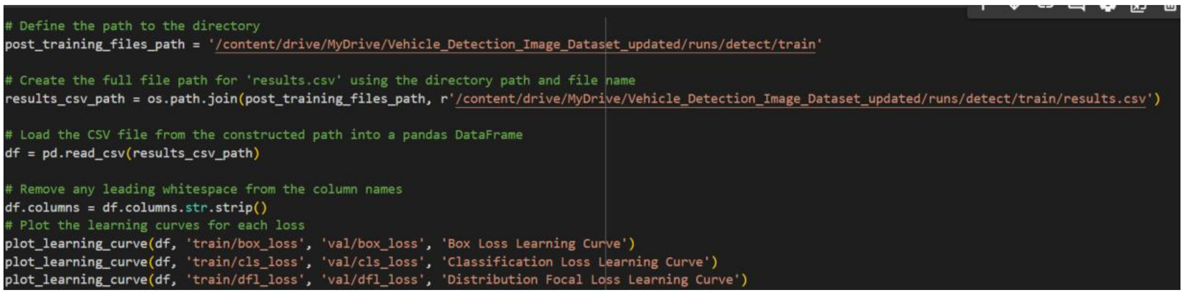
```python
# Draw the quadrilaterals on the processed frame
cv2.polylines(processed_frame, [vertices1], isClosed=True, color=(0, 255, 0), thickness=2)
cv2.polylines(processed_frame, [vertices2], isClosed=True, color=(255, 0, 0), thickness=2)
# Retrieve the bounding boxes from the results
bounding_boxes = results[0].boxes

# Initialize counters for vehicles in each lane
vehicles_in_left_lane = 0
vehicles_in_right_lane = 0
# Loop through each bounding box to count vehicles in each lane
for box in bounding_boxes.xyxy:
    # Check if the vehicle is in the left lane based on the x-coordinate of the bounding box
    if box[0] < lane_threshold:
        vehicles_in_left_lane += 1
    else:
        vehicles_in_right_lane += 1
# Determine the traffic intensity for the left lane
traffic_intensity_left = "Heavy" if vehicles_in_left_lane > heavy_traffic_threshold else "Smooth"
# Determine the traffic intensity for the right lane
traffic_intensity_right = "Heavy" if vehicles_in_right_lane > heavy_traffic_threshold else "Smooth"
# Add a background rectangle for the left lane vehicle count
cv2.rectangle(processed_frame, (text_position_left_lane[0]-10, text_position_left_lane[1] - 25),
              (text_position_left_lane[0] + 460, text_position_left_lane[1] + 10), background_color, -1)
# Add the vehicle count text on top of the rectangle for the left lane
cv2.putText(processed_frame, f'Vehicles in Left Lane: {vehicles_in_left_lane}', text_position_left_lane,
            font, font_scale, font_color, 2, cv2.LINE_AA)
```



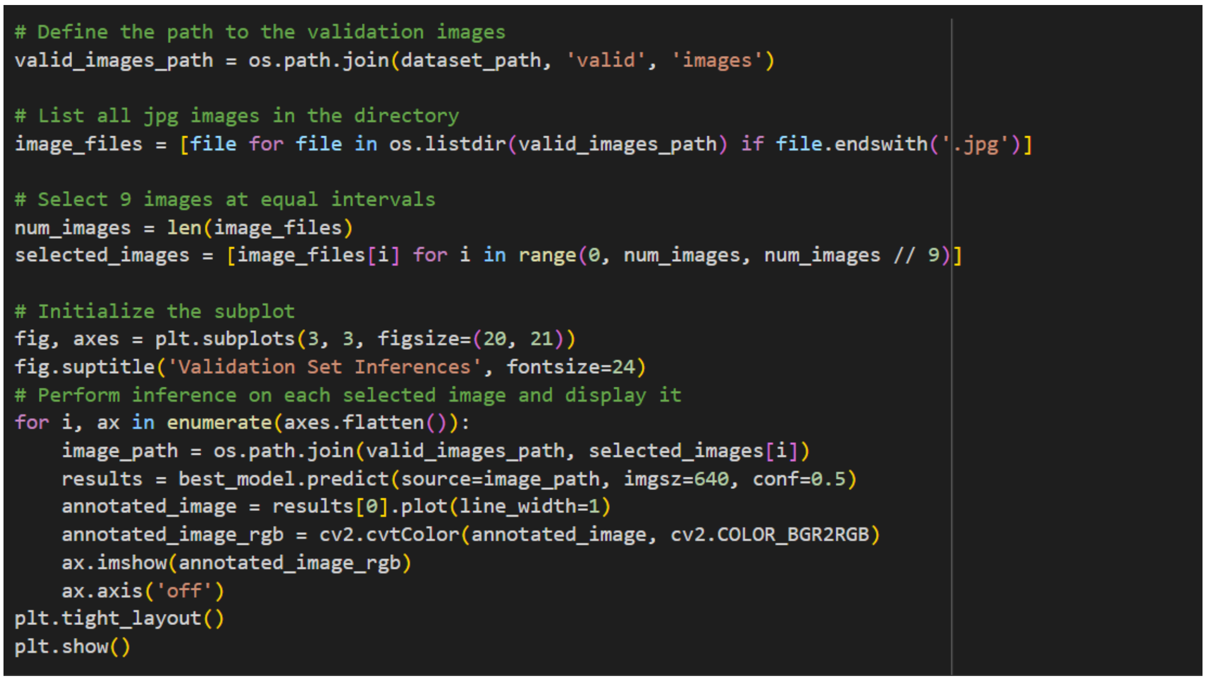
```python
# Define the dataset_path
dataset_path = '/content/drive/MyDrive/Vehicle_Detection_Image_Dataset_updated'

# Set the path to the YAML file
yaml_file_path = os.path.join(dataset_path, 'data.yaml')
# Load and print the contents of the YAML file
with open(yaml_file_path, 'r') as file:
    yaml_content = yaml.load(file, Loader=yaml.FullLoader)
    print(yaml.dump(yaml_content, default_flow_style=False))
```



```python
# Train the model on our custom dataset
results = model.train(
    data=yaml_file_path,     # Path to the dataset configuration file
    epochs=50,              # Number of epochs to train for
    imgsz=640,               # Size of input images as integer
    device=0,                # Device to run on, i.e. cuda device=0
    patience=50,             # Epochs to wait for no observable improvement for early stopping of training
    batch=32,                # Number of images per batch
    optimizer='auto',        # Optimizer to use, choices=[SGD, Adam, Adamax, AdamW, NAdam, RAdam, RMSProp, auto]
    lr0=0.0001,              # Initial learning rate
    lrf=0.1,                 # Final learning rate (lr0 * lrf)
    dropout=0.1,             # Use dropout regularization
    seed=0                   # Random seed for reproducibility
)
```



```python
# Define the path to the directory
post_training_files_path = '/content/drive/MyDrive/Vehicle_Detection_Image_Dataset_updated/runs/detect/train'

# Create the full file path for 'results.csv' using the directory path and file name
results_csv_path = os.path.join(post_training_files_path, r'/content/drive/MyDrive/Vehicle_Detection_Image_Dataset_updated/runs/detect/train/results.csv')

# Load the CSV file from the constructed path into a pandas DataFrame
df = pd.read_csv(results_csv_path)

# Remove any leading whitespace from the column names
df.columns = df.columns.str.strip()
# Plot the learning curves for each loss
plot_learning_curve(df, 'train/box_loss', 'val/box_loss', 'Box Loss Learning Curve')
plot_learning_curve(df, 'train/cls_loss', 'val/cls_loss', 'Classification Loss Learning Curve')
plot_learning_curve(df, 'train/dfl_loss', 'val/dfl_loss', 'Distribution Focal Loss Learning Curve')
```



```python
# Define the path to the validation images
valid_images_path = os.path.join(dataset_path, 'valid', 'images')

# List all jpg images in the directory
image_files = [file for file in os.listdir(valid_images_path) if file.endswith('.jpg')]

# Select 9 images at equal intervals
num_images = len(image_files)
selected_images = [image_files[i] for i in range(0, num_images, num_images // 9)]

# Initialize the subplot
fig, axes = plt.subplots(3, 3, figsize=(20, 21))
fig.suptitle('Validation Set Inferences', fontsize=24)
# Perform inference on each selected image and display it
for i, ax in enumerate(axes.flatten()):
    image_path = os.path.join(valid_images_path, selected_images[i])
    results = best_model.predict(source=image_path, imgsz=640, conf=0.5)
    annotated_image = results[0].plot(line_width=1)
    annotated_image_rgb = cv2.cvtColor(annotated_image, cv2.COLOR_BGR2RGB)
    ax.imshow(annotated_image_rgb)
    ax.axis('off')
plt.tight_layout()
plt.show()
```