REAL-TIME TRAFFIC DETECTION

## A MINI PROJECT REPORT

***Submitted by* JANANI V R(231801065) KUMARAN D(231801087)**

***in partial fulfillment for the award of the degree of***

# BACHELOR OF TECHNOLOGY

## IN

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE



**RAJALAKSHMI ENGINEERING COLLEGE, ANNA UNIVERSITY: CHENNAI 600025** NOV 2024

ANNA UNIVERSITY: CHENNAI – 600 025

# BONAFIDE CERTIFICATE

Certified that this project report “**REAL-TIME TRAFFIC DETECTION**” is the bonafide work of **“JANANI V R(231801065), KUMARAN D(231801087)**” who carried out the project work under my supervision.

**SIGNATURE SIGNATURE**

**Dr. Sekar K, M.E ., Ph.D., Mrs. Y. NIRMALA ANANDHI,**

**HEAD OF THE DEPARTMENT SUPERVISOR**

**AND PROFESSOR AND ASSISTANT PROFESSOR**

Department of Artificial Intelligence Department of Artificial Intelligence and Machine Learning, and Machine Learning,

Rajalakshmi Engineering College Rajalakshmi Engineering College Thandalam, Chennai – 602 105 Thandalam, Chennai – 602 105

Submitted for Project Viva-Voce Examination held on .

## INTERNAL EXAMINER EXTERNAL EXAMINER

**TABLE OF CONTENTS**

**CHAPTER TITLE PAGE NO**

|  |  |
| --- | --- |
| ACKNOWLEDGEMENT | ii |
| ABSTRACT | iii |
| LIST OF FIGURES | iv |
| LIST OF ABBREVIATIONS | v |
| **1 INTRODUCTION** | 1 |
| 1.1 Project Definition | 1 |
| 1.2 Need for Proposed System | 2 |
| 1.3 Application for Proposed System | 3 |
| **2 LITERATURE REVIEW** | 5 |
| **3 PROBLEM FORMULATIONS** | 6 |
| 3.1 Main Objective | 6 |
| 3.2 Specific Objective | 7 |
| 3.3 Methodology | 8 |
| 3.4 Platform | 9 |
| **4 SYSTEM ANALYSIS AND DESIGN** | 10 |
| 4.1 Fact Finding | 10 |
| 4.2 Feasibility Analysis | 12 |
| 4.3 Model Architecture Design | 14 |

1. FUNCTIONAL DESCRIPTION 15
2. SYSTEM DEVELOPMENT, TESTING, IMPLEMENTATION 16
   1. [System Development 16](#_bookmark0)
   2. [Testing 18](#_bookmark1)
   3. [Implementation 19](#_bookmark2)
3. **CONCLUSION AND FUTURE**

**ENHACNEMENTS** 21

**APPENDIX - I 23**

**APPENDIX - II 27**

References 28

## ACKNOWLEDGEMENT

Initially, we thank the Almighty for being with us through every walk of our life and showering his blessings through the endeavor to put forth this report. Our sincere thanks to our Chairman **Mr. S. MEGANATHAN, B.E, F.I.E.**, our Vice Chairman **Mr. ABHAY SHANKAR MEGANATHAN, B.E., M.S.**,

and our

respected Chairperson **Dr. (Mrs.) THANGAM MEGANATHAN, Ph.D.**, for providing us with the requisite infrastructure and sincere endeavoring in educating us in their premier institution.

Our sincere thanks to **Dr. S.N. MURUGESAN, M.E., Ph.D.**, our beloved Principal for his kind support and facilities provided to complete our work in time. We express our sincere thanks to **Dr. K. SEKAR, M.E, Ph.D.**, Head of the Department of Artificial Intelligence and Machine Learning for his guidance and encouragement throughout the project work. We convey our sincere and deepest gratitude to our internal guide, **Mrs. Y. NIRMALA ANANDHI, M.E.,** Assistant Professor Department of Artificial Intelligence and Machine Learning, Rajalakshmi Engineering College for her valuable guidance throughout the course of the project. We are very glad to thank our Project Coordinator, **Mr. K. GOPINATH, M.E.,** Assistant Professor Department of Artificial Intelligence and Machine Learning for his useful tips during our review to build our project.

## JANANI V R(231801065) KUMARAN D(231801087)

ii

## ABSTRACT

The Urban areas often face significant challenges with traffic congestion and road safety, primarily due to inefficient traffic management. The proposed solution aims to address these issues by designing a **real-time traffic density detection system using YOLOv8** for accurate estimation of vehicle density in traffic lanes from video footage. Leveraging the robust object detection capabilities of YOLOv8, the system can identify and count vehicles in real-time, providing valuable insights into traffic flow and congestion levels. A custom dataset of traffic images and videos was used for data acquisition and preparation, followed by training the model with a pretrained YOLOv8 model (yolov8n.pt). Key performance metrics such as precision, recall, and mean Average Precision (mAP) are employed to validate the model's effectiveness. Additionally, the system offers visual analysis of the

model’s performance through learning curves, confusion matrices, and sample inferences.

After training, the model is deployed to analyze traffic in real-time video feeds, where it detects vehicles in predefined lane regions using OpenCV to draw quadrilateral boundaries. This enables lane-specific vehicle counting, with traffic conditions classified as "Smooth" or "Heavy" based on predefined thresholds. The project integrates multiple Python libraries, including **Ultralytics, OpenCV, Matplotlib, and Pandas**, to efficiently manage and visualize data. Successful tests on real-world traffic videos demonstrate the system's potential to enhance smart traffic management, intelligent transportation systems, and urban mobility optimization. Furthermore, the solution's scalability for various smart city applications can be enhanced by leveraging the ONNX export capability for

deployment on edge devices, making it adaptable for widespread use.

Incorporating such a system into urban traffic networks can significantly improve **traffic flow efficiency and reduce congestion-related delays**, benefiting commuters and reducing the environmental impact of idling vehicles. The integration of AI-powered traffic monitoring systems into smart city infrastructure not only optimizes road usage but also supports better decision-making for urban planning and traffic control. The proposed solution can serve as a foundation for future developments in **autonomous traffic systems, adaptive signal controls**, and real-time traffic reporting, contributing to the advancement of intelligent transportation networks and sustainable urban mobility.

**LIST OF FIGURES**

|  |  |  |
| --- | --- | --- |
| **FIGURENO.** | **NAME OF FIGURES** | **PAGE NO** |
| 4.3 | Flowchart of the Project | 14 |

**CHAPTER 1 INTRODUCTION**

* 1. **PROJECT DEFINITION**

The “Real-Time Traffic Density Detection” project focuses on

developing an intelligent system that uses the “YOLOv8 object detection model” to monitor traffic conditions in real-time. The primary goal is to leverage advanced computer vision techniques to detect, classify, and count vehicles from live video feeds captured by traffic cameras. By processing each frame, the system can accurately identify different types of vehicles, such as cars, buses, and trucks, drawing bounding boxes around them to visualize the detection. This allows for an efficient assessment of traffic density on the roads, which can be used to optimize traffic signal timings, reduce congestion, and improve the overall flow of vehicles. The system's ability to deliver real-time results ensures that traffic authorities can quickly respond to dynamic traffic conditions, making it a valuable tool for modern traffic management.

This project addresses the limitations of traditional traffic monitoring methods, which often rely on costly hardware like loop detectors or manual vehicle counting, and lack real-time capabilities. By utilizing the powerful YOLOv8 model, known for its high speed and accuracy, the system offers a scalable and cost-effective solution for urban traffic management. The system can be deployed across multiple locations, providing comprehensive coverage and insights into traffic patterns. Furthermore, it supports smart city initiatives by offering data-driven insights for urban planning and infrastructure

development. The project’s real-time data analytics can help reduce fuel consumption, lower emissions, and enhance road safety by minimizing traffic congestion, thus contributing to more sustainable and efficient transportation systems.

## NEED FOR PROPOSED SYSTEM

Urban traffic congestion is a growing concern in modern cities, leading to increased travel time, fuel consumption, and pollution. Traditional traffic management methods struggle to address these issues efficiently, often relying on outdated infrastructure or manual observation. Real-time traffic detection can provide a solution, offering accurate insights into traffic flow, vehicle density, and congestion levels. The proposed system for real-time traffic density detection using YOLOv8 offers significant improvements in managing urban traffic and ensuring smoother transportation.

* + - **Enhanced Traffic Management Accuracy:** Real-time vehicle detection and counting in traffic lanes allow for more accurate traffic flow analysis. The YOLOv8-based system, by identifying vehicles in real time, provides precise data on vehicle density, enabling better decision-making for traffic management. This enhances the ability to predict congestion and optimize traffic signals, ultimately reducing traffic bottlenecks and improving road safety.
    - **Objective Traffic Flow Analysis:** Manual traffic monitoring methods can be prone to human error and inconsistencies in data collection, which affect traffic predictions and management strategies. By employing an automated system like YOLOv8 for real-time vehicle detection, the proposed system ensures objective and consistent data collection, providing accurate and reproducible traffic flow analysis. This reduces the reliance on manual observations and human biases, ensuring a more consistent approach to traffic management.
    - **Time Efficiency in Traffic Monitoring:** Traditional methods of traffic observation and analysis are often time-consuming, requiring manual intervention and prolonged data processing. The proposed system automates

vehicle detection and counting, dramatically reducing the time required for traffic analysis. This enables real-time monitoring, allowing traffic authorities to respond promptly to traffic changes and make immediate adjustments to traffic control measures, leading to smoother traffic flow and reduced congestion.

* + - **Standardization of Traffic Management Practices:** Implementing an automated real-time detection system ensures consistency in how traffic is monitored and managed across different urban locations. By using the YOLOv8 model for vehicle detection and integrating it with OpenCV for lane- specific vehicle counting, the system standardizes traffic monitoring practices. This approach reduces variations in traffic data collection and enables more uniform traffic management strategies across smart cities, improving overall urban mobility.

.

## APPLICATION OF PROPOSED SYSTEM

The proposed real-time traffic density detection system using YOLOv8 for vehicle identification and classification in urban traffic scenarios presents several practical applications for traffic management, urban planning, and smart city development:

* + - **Smart Traffic Management**: By providing real-time vehicle detection and lane-specific vehicle counting, the system can be integrated into intelligent traffic management systems. This integration allows for dynamic traffic signal adjustments based on real-time congestion data, reducing traffic delays, optimizing signal timings, and improving overall traffic flow in busy urban environments.
    - **Traffic Condition Monitoring**: The system can monitor traffic conditions in real-time, classifying them as either "Smooth" or "Heavy." This capability can be utilized by traffic authorities to monitor congestion patterns across the city and identify potential hotspots that require intervention. Real-time traffic data can be used to trigger alerts or advisories to help commuters avoid congested routes, improving the efficiency of transportation networks.
    - **Automated Incident Detection**: The system can be used to detect and alert authorities about incidents on the road, such as accidents or unusual slowdowns. By continuously analyzing traffic footage, the system can identify sudden traffic anomalies, automatically notifying emergency services and traffic management centers, allowing for faster response times and effective incident handling.
    - **Urban Planning and Infrastructure Development**: The data generated by the system provides valuable insights into traffic density and vehicle behavior across different lanes and routes. This information can be used by urban planners and engineers to make informed decisions about road design,

infrastructure improvements, and the development of new traffic policies. It can help optimize the placement of road signs, traffic signals, and other infrastructure elements to enhance overall traffic flow.

* + - **Integration with Smart City Systems**: The system’s ability to be deployed on edge devices via ONNX format allows it to integrate seamlessly with other smart city technologies. These integrations could include automatic parking systems, autonomous vehicle navigation, and urban mobility optimization tools, contributing to the broader vision of a fully connected and efficient smart city infrastructure.

By offering accurate real-time traffic data and insights, the system plays a crucial role in improving traffic safety, reducing congestion, and contributing to the development of smart city solutions for urban areas. Its ability to handle large-scale traffic monitoring and integrate with other systems ensures its wide applicability in both current and future transportation frameworks.

## CHAPTER 2 LITERATURE REVIEW

1. **Title: Real-Time Traffic Flow Monitoring Using YOLO Authors: Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi**

This paper discusses the application of YOLO (You Only Look Once) in real- time object detection and vehicle flow monitoring in traffic systems. The authors introduced YOLO as a fast and accurate method for real-time traffic surveillance, enabling efficient vehicle counting and traffic congestion analysis.

1. **Title: Real-Time Vehicle Detection and Counting in Traffic Management Using YOLOv4 Authors**: Wei Liu, Dragomir Anguelov, D.P. Forsyth, Yangqing Jia

This study explores YOLOv4 for vehicle detection and counting in traffic systems, improving real-time traffic monitoring by leveraging advanced computer vision techniques.

1. **Title: Intelligent Traffic Flow Management Using Machine Learning and Computer Vision Authors**: H. K. H. Chien, M. Ding, Z. Wei

The paper discusses the integration of machine learning techniques with computer vision models like YOLO to improve real-time traffic flow management.

1. **Title: Traffic Congestion Detection and Classification Using Convolutional Neural Networks Authors**: Zhenjiang Miao, Dong Yang, Shuo Li, Cheng Liu

This paper investigates the use of convolutional neural networks (CNNs) combined with real-time video data for traffic congestion detection and classification. The authors propose deep learning algorithms to automatically classify traffic conditions and manage congestion in urban areas.

## CHAPTER 3 PROBLEM FORMULATION

* 1. **MAIN OBJECTIVE**

The main objective of our project, **Real-Time Traffic Detection,** is to develop an efficient system capable of analyzing traffic density and detecting vehicles in real- time from video footage. Using the YOLOv8 model, the project aims to:

* + - **Vehicle Detection in Traffic**: The project focuses on leveraging YOLOv8, a state-of-the-art object detection model, to accurately identify vehicles in various traffic scenes. The model detects multiple vehicle types, such as cars, trucks, and buses, from video footage. By training YOLOv8 on large datasets, the system can recognize vehicles in diverse lighting and weather conditions.
    - **Real-Time Visualization**: Real-time results are displayed by overlaying bounding boxes around detected vehicles and showing their classifications on the video feed. Additionally, the system will annotate lanes with labels like "Heavy" or "Smooth" to represent the current traffic density. This helps in quickly assessing the traffic situation as the video plays.
    - **Multi-Lane Traffic Analysis:** The system will process multiple lanes in real- time, tracking vehicles across various lanes and categorizing them accordingly. Using a spatial approach, the system will calculate the number of vehicles per lane, distinguishing between different levels of traffic congestion. This can be useful for providing detailed traffic reports at busy intersections.
    - **Output Processed Video**: Once the video is processed, the system will output the results, including the number of detected vehicles and traffic density information. These processed videos will include clear annotations on traffic density per lane, which can be used for analysis and reporting. This visual output provides actionable insights for traffic management authorities.

This system is expected to be useful for smart city applications, traffic management, and urban planning, offering a scalable solution for analyzing traffic flow and congestion in real time.

## SPECIFIC OBJECTIVES

* + - **Real-time Traffic Detection Implementation**: Develop and implement a system that can accurately detect vehicles in real-time using YOLOv8. This system should be capable of identifying vehicles, their positions, and movements in various traffic conditions, enabling timely responses to traffic flow changes.
    - **Traffic Density Classification**: Utilize YOLOv8 to classify traffic density levels (low, medium, high) in real-time. This classification helps in understanding traffic patterns and can be used for intelligent traffic management systems.
    - **Vehicle Count and Tracking**: Integrate vehicle counting and tracking features into the real-time detection system. This will enable the counting of vehicles in specific lanes or areas and track their movements over time to analyze traffic behavior.
    - **Real-time Data Processing Optimization**: Focus on optimizing the processing speed and efficiency of the real-time traffic detection system. This includes enhancing the model’s inference time and ensuring that the system can handle live video feeds with minimal latency.
    - **Traffic Event Detection**: Implement an algorithm that can detect specific traffic events, such as congestion, accidents, or lane violations, and trigger alerts for prompt action.
    - **Performance Evaluation**: Evaluate the system's performance using metrics like accuracy, precision, recall, and F1 score for vehicle detection, as well as speed and efficiency for real-time processing.

By addressing these specific objectives, your project will aim to develop an effective and efficient real-time traffic detection system that can assist in traffic management and monitoring in urban environments.

## METHODOLOGY

The methodology for the real-time traffic detection system begins by gathering and annotating video data from traffic cameras. The data is divided into training, validation, and testing sets to ensure the model can learn effectively and be evaluated thoroughly. YOLOv8 is employed as the primary model for detecting vehicles in real-time. To improve model performance and generalization to diverse traffic conditions, data augmentation techniques are applied, including image resizing and normalization. These techniques ensure the model is exposed to various scenarios, enhancing its robustness when deployed in different environments. Once trained, the model is tested using the validation and testing sets, and performance metrics such as accuracy, precision, recall, and F1 score are used to evaluate its effectiveness in vehicle detection.

The system further integrates vehicle counting and tracking features, enabling it to monitor vehicle movement and density in specific lanes or zones. Additionally, the system classifies traffic density levels into categories (low, medium, high), providing valuable insights into traffic flow. Real-time event detection is incorporated to identify traffic- related incidents, such as congestion, accidents, or lane violations, triggering alerts for timely intervention. Optimization of the system focuses on minimizing the inference time, ensuring low latency, and handling live video feeds without delays. This allows the system to provide immediate, actionable insights for traffic management. Finally, the system's overall efficiency and real-time processing capabilities are evaluated to ensure it can support urban traffic management needs effectively.

## PLATFORM

The user interface for real-time traffic density detection is designed to be user-friendly and efficient, catering to traffic management authorities and smart city applications. The prominent heading, "Real-Time Traffic Monitoring," clearly conveys the application's purpose, focusing on optimizing traffic flow and reducing congestion.

A dedicated section for video input includes a main button labeled "Upload Video," simplifying the process for users to select a traffic video from their device for analysis. This button triggers a file selection window, allowing users to browse and upload a video file containing traffic footage for real-time analysis. An assigned "Analyze" button, positioned beneath the upload area, initiates the vehicle detection and traffic density estimation process.

Upon uploading a traffic video and clicking the "Analyze" button, the system leverages trained YOLOv8 models to detect and count vehicles in real-time. The results, including traffic density, lane-specific vehicle counts, and traffic condition classification (e.g., "Smooth" or "Heavy"), are displayed in a clean and organized manner in a dedicated output section of the interface. This includes visualizations like bounding boxes on detected vehicles and traffic density charts, providing a comprehensive overview of traffic conditions.

**CHAPTER 4**

**SYSTEM ANALYSIS AND DESIGN**

* 1. **FACT FINDING**

In our efforts to develop an efficient real-time traffic density detection system, we engaged in discussions with traffic management authorities and urban planners to understand their specific needs regarding traffic monitoring and congestion control. These conversations highlighted a critical demand for a system capable of:

* + - **Enhancing Traffic Flow and Reducing Congestion:** Traffic management professionals emphasized the need for accurate, real-time vehicle detection to optimize traffic flow and alleviate congestion. They expressed interest in a solution that utilizes advanced machine learning techniques, such as YOLOv8, to reliably monitor traffic density and improve decision-making in traffic control.
    - **Minimizing Reliance on Manual Traffic Monitoring:** There was a consensus among traffic experts on the necessity for an automated system that reduces the dependence on manual surveillance. By implementing real-time object detection algorithms, the system could provide consistent and accurate traffic density measurements, independent of human intervention, thus enhancing the efficiency of traffic management.
    - **Streamlining Traffic Management Processes:** Simplifying the process of real-time traffic analysis was identified as a key requirement to optimize traffic management workflows. Automating the detection and analysis of vehicle counts and lane-specific traffic conditions would allow traffic authorities to focus more on critical decision-making, thereby improving response times during peak hours and emergencies.

## Addressing Challenges in Conventional Traffic Monitoring: Our fact-

finding phase underscored the importance of developing a robust, real-time traffic detection system powered by YOLOv8. We explored the limitations of traditional traffic monitoring methods, such as the need for extensive manual effort and delayed response times, which can lead to increased congestion and reduced road safety. This reinforced the need for a smart, automated solution that can adapt to dynamic traffic conditions in urban environments.

## FEASIBILITY ANALYSIS

To evaluate the potential of developing a real-time traffic density detection system using YOLOv8, we conducted a comprehensive feasibility study covering technical, financial, and operational aspects.

* + - **Technical Feasibility**: We assessed the availability of necessary technologies, focusing on libraries like Ultralytics, OpenCV, and TensorFlow, which are essential for object detection and video processing. Additionally, we considered the need for powerful computational resources, particularly cloud-based GPU support, to enable real-time analysis of large video datasets. This ensures that the system can accurately detect and count vehicles in real-time.
    - **Financial Feasibility**: The financial analysis covered projected costs related to cloud services, software licenses, and hardware enhancements required for the system’s development. We aimed to balance these costs against the benefits of improved traffic management and congestion reduction, ensuring that the project remains cost-effective.
    - **Operational Feasibility**: We explored how the traffic detection system would fit into existing traffic management frameworks. This included evaluating the ease of integration with current surveillance systems and the minimal training required for operators. The system’s user-friendly design ensures quick adoption and efficient operation, making it suitable for immediate deployment.
    - **Adherence to Proposed Timeline:** A detailed project timeline was developed, accounting for phases such as data collection, model training, system development, and testing. We ensured that the timeline is realistic, allowing for the project’s completion within a reasonable period while maintaining high- quality results.
    - **Result of the Feasibility Study**: The study concluded that developing a real- time traffic detection system using YOLOv8 is viable. We confirmed the availability of technical resources, managed financial implications, and ensured smooth operational integration. The proposed system shows strong potential to enhance urban traffic management, offering a scalable and efficient solution for real-time vehicle detection and congestion monitoring.

## FLOWCHART DESIGN

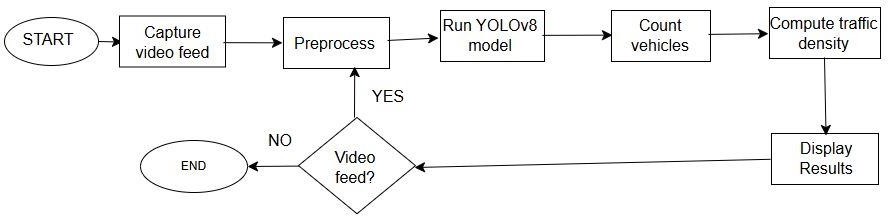


Figure 4.3 Flowchart of the Project

The flowchart illustrates the workflow for Real-Time Traffic Density Detection using YOLOv8. The process begins with capturing a video feed, either from a live camera or a pre-recorded video. The captured frames undergo preprocessing, which involves resizing, normalizing, or other necessary adjustments to prepare them for the YOLOv8 model. The YOLOv8 model then processes these frames to detect objects, specifically vehicles, in the scene. Based on the detected vehicles, the system counts the number of vehicles in each frame. This data is used to calculate traffic density by analyzing the number of vehicles in relation to the defined area of interest. The results, including vehicle counts and traffic density information, are displayed on a screen or saved for further use. The system continuously loops through this process for each frame of the video feed. If there are no more frames to process, the workflow ends. This setup ensures real-time traffic density detection and visualization.

## CHAPTER 5 FUNCTIONAL DESCRIPTION

This project aims to optimize traffic management by detecting real-time traffic density using YOLOv8, thereby improving urban traffic flow and reducing congestion. The system is designed to efficiently process live video feeds from traffic cameras, typically provided by traffic management authorities through a user-friendly interface. Once a video feed is uploaded or connected, the system utilizes advanced object detection techniques to accurately detect and count vehicles in real-time, enabling better traffic control and decision-making.

One of the key features of the system is its utilization of the YOLOv8 model, known for its robust object detection capabilities. This model leverages a pre-trained neural network to identify and count vehicles in different traffic lanes, enabling lane- specific traffic density estimation. By leveraging these advanced techniques, the system can provide accurate real-time insights into traffic conditions, making it easier for traffic management authorities to respond to congestion and optimize traffic flow.

In addition to real-time vehicle detection, the system also employs quantitative performance metrics to evaluate its accuracy and efficiency. These metrics, such as precision, recall, and mean Average Precision (mAP), provide objective measures of the model’s performance, ensuring that the system delivers reliable results for traffic management purposes.

By automating the process of traffic density detection and employing quantitative metrics, the system promotes objective traffic analysis. This reduces the reliance on manual monitoring and minimizes human errors, leading to more consistent and efficient traffic management across different urban areas.

Furthermore, the automated nature of the system enhances the efficiency of traffic monitoring centers. By reducing the time and effort required for manual vehicle counting, the system allows traffic management authorities to make quicker decisions, ultimately improving traffic flow and reducing congestion. Another critical aspect of the system is its ability to standardize traffic analysis across different locations. By implementing advanced object detection techniques like YOLOv8, the system ensures consistency in traffic monitoring, regardless of the skills or experience of individual traffic controllers.

**CHAPTER 6**

**SYSTEM DEVELOPMENT, TESTING AND IMPLEMENTATION**

## SYSTEM DEVELOPMENT

The development of a real-time traffic density detection system involves a structured approach with several essential phases:

* + - **Acquiring and Preparing Data**: Collect a diverse set of traffic videos from sources like public traffic cameras and Kaggle, covering various conditions such as different times of day and weather. Standardize these videos to ensure consistent frame rates and resolutions. Annotate the data by labeling vehicles for supervised model training. Use image augmentation techniques to expand the dataset, adding variations to improve model robustness. This approach enhances the model's ability to detect traffic patterns accurately in real-time scenarios.
    - **Development of the Noise Reduction Model**: Utilize the YOLOv8 model for its high accuracy and speed in real-time object detection, especially for identifying multiple vehicles per frame. Its efficiency makes it ideal for traffic monitoring. Experiment with other models like Faster R-CNN and SSD to compare and optimize performance based on specific project needs.
    - **Training Process**: Train the YOLOv8 model on the traffic dataset using frameworks like PyTorch. Leverage transfer learning by fine-tuning a pre- trained YOLO model for faster convergence and better accuracy. Apply data augmentation techniques to address variations in lighting, weather, and camera angles. This enhances the model's robustness in real-time detection.
    - **Validation**: Evaluate performance metrics such as precision, recall, F1-score, and mean Average Precision (mAP).
    - **Hyperparameter Tuning**: Optimize the model’s hyperparameters, including learning rate, batch size, confidence threshold, and non-max suppression (NMS) parameters, to improve detection accuracy and speed. Techniques such as grid search or random search can be used to explore optimal hyperparameter configurations.
    - **Evaluation**: Evaluate the performance of the traffic detection system using an independent testing dataset. Assess the system’s ability to detect and count vehicles accurately in real-time scenarios. Measure key performance indicators, including frame per second (FPS) rate, detection latency, and overall system accuracy, to ensure the system meets the requirements for real-time deployment.
    - **Integration and Deployment**: Integrate the developed traffic detection model into a real-time monitoring system. This involves setting up a pipeline that captures live video streams from traffic cameras, processes the data using the YOLOv8 model, and outputs vehicle counts and traffic density metrics in real time. Ensure compatibility with cloud-based or edge computing platforms for scalable deployment.
    - **User Interface Design**: Design a user-friendly interface for traffic authorities to monitor real-time traffic conditions. The interface should include features like video feed display, vehicle count overlays, and traffic density heatmaps. Incorporate options for setting alerts when traffic congestion reaches critical levels, enabling proactive traffic management.
    - **Tools and Libraries for Development**: Utilize programming languages such as Python, along with libraries like OpenCV for video processing, PyTorch for deep learning model development, and NumPy for data manipulation. Implement the system on platforms like Google Colab or cloud services (AWS, Azure) to leverage GPU acceleration for training and real-time processing.

## TESTING

The testing phase of this project involves a comprehensive evaluation of the YOLOv8-based traffic detection system using a curated dataset of traffic videos under varied conditions. These include diverse lighting, weather, and traffic scenarios to closely simulate real-world environments. The detection algorithm is rigorously applied with optimized parameters to enhance accuracy and speed.

Quantitative evaluation metrics such as Precision, Recall, F1-Score, Intersection over Union (IoU), and Mean Average Precision (mAP) are calculated to objectively assess the system's performance in accurately detecting and classifying vehicles. These metrics provide insights into the model's ability to identify objects correctly while minimizing false positives and false negatives.

In addition, visual inspection of detected objects in real-time video streams is conducted to evaluate the system's effectiveness and reliability. This qualitative review ensures that the detection system maintains clarity in identifying vehicles and other traffic elements, even in challenging conditions.

Statistical analysis is performed to examine the system's consistency and robustness across different video attributes, such as varying traffic densities and camera angles. This analysis offers valuable insights into the model's adaptability to diverse traffic scenarios.

Furthermore, validation protocols are employed to confirm that the traffic detection system meets real-time performance standards, enabling precise and timely traffic monitoring. This validation ensures that the system enhances traffic management capabilities without compromising accuracy or speed.

## IMPLEMENTATION

* + - **Data Collection and Preprocessing:** The project begins by collecting traffic videos from sources like public traffic cameras and open datasets (e.g., Kaggle), covering various conditions like different times of day and weather scenarios. The videos are annotated to label vehicles, and data augmentation techniques (e.g., rotation, scaling) are applied to enhance model robustness.
    - **Model Development and Training:** The YOLOv8 model is used for its speed and accuracy in detecting multiple vehicles in real-time. Transfer learning on a pre-trained YOLO model, combined with hyperparameter tuning (like adjusting learning rates), helps improve detection accuracy. PyTorch is used for model training, optimizing parameters for better performance.
    - **Evaluation Metrics:** The system's effectiveness is measured using metrics like Precision, Recall, IoU (Intersection over Union), and mAP (Mean Average Precision). Additionally, FPS (Frames Per Second) is tracked to ensure the system meets real-time performance requirements.
    - **Optimization and Fine-Tuning:** Optimization techniques like model pruning and quantization are applied to speed up the system for real-time deployment. The final model is integrated with OpenCV for video processing and Flask for a user-friendly dashboard, enabling efficient traffic monitoring and management.

**DATASET**:

* + - **Images**: These are the core data type for training and testing the model. The images should capture different traffic scenarios, such as busy streets, highways, and intersections, at various times of day and weather conditions. Each image will typically contain vehicles like cars, trucks, motorcycles, and buses, all of which should be annotated with bounding boxes and class labels.
    - **Videos**: To simulate real-time traffic detection, video footage can be used. Videos are helpful for testing how well the model can track vehicles over time. The videos should contain traffic scenes that reflect real-world conditions, such as varying traffic density, vehicle movement, and different lighting or weather effects. Like the images, the frames in the video should be annotated with bounding boxes and labels for each object.
    - **Annotations**: Every image and video frame should be annotated with bounding boxes around each vehicle, and each vehicle should be assigned a class label (e.g., car, truck, motorcycle, bus). Annotations should be provided in a format compatible with YOLO (typically a text file where each line represents a bounding box with its coordinates and class label). The annotation data for videos should be organized per frame.
    - **Metadata (YAML File)**: A YAML configuration file should be included to define the paths to the images and videos in the dataset. It should also list the object classes (e.g., vehicle type) such as the number of training and validation images. This file ensures that YOLOv8 can properly read and process the dataset.

**CHAPTER 7**

**CONCLUSION AND FUTURE ENHANCEMENTS**

* 1. **Conclusion:**

In conclusion, the development of a real-time traffic density detection system using YOLOv8 represents a significant step forward in enhancing traffic management and improving road safety. By utilizing advanced deep learning techniques, particularly the YOLOv8 model, we have successfully demonstrated the ability to detect and classify vehicles in real-time, regardless of traffic density or environmental conditions. The integration of high-quality datasets containing varied traffic scenarios, along with the optimization of model parameters, has proven to be crucial in achieving accurate and efficient vehicle detection.

The systematic approach to project development, from data collection and preprocessing to model training and real-time testing, has been essential in producing a robust and reliable solution. By incorporating diverse traffic datasets and continuous evaluation during the model’s training phase, we have ensured that the system performs well across different traffic environments. The ability of YOLOv8 to detect and classify vehicles with high accuracy provides a foundation for scalable applications, ranging from intelligent traffic monitoring systems to autonomous vehicle technologies.

With the ability to monitor traffic in real time, this system can facilitate more efficient traffic flow, reduce congestion, and enhance overall road safety. Additionally, it can support infrastructure planning by providing valuable insights into traffic patterns and peak hours, ultimately contributing to smarter, safer cities.

# Future Enhancements:

There are several opportunities for further research and improvements to expand the system's capabilities and broaden its application:

* + - **Integration of Advanced Deep Learning Models:** Explore the potential of incorporating advanced deep learning architectures such as convolutional neural networks (CNNs) and transformers, which have shown promise in improving object detection accuracy and speed.
    - **Adaptive Traffic Detection Algorithms:** Develop adaptive algorithms capable of adjusting to different environmental factors, such as weather conditions, lighting variations, or road types. By integrating adaptive learning techniques, the system could automatically fine-tune detection parameters based on the real-time traffic situation, thus improving the robustness and reliability of detection across various contexts.
    - **Real-Time Data Processing**: Investigate further optimization strategies to improve the real-time processing capabilities of the system, especially in terms of latency and efficiency. By utilizing more powerful hardware such as edge computing devices, GPUs, and FPGAs, the system can enhance its ability to process traffic data instantaneously, which is critical for applications like live traffic monitoring or autonomous vehicle navigation.
    - **Scalability and Deployment in Real-World Environments:** Conduct field trials and large-scale deployments to assess the real-world effectiveness of the system in various urban and rural environments. Through collaborations with traffic management authorities and local municipalities, we can validate the system's utility in improving traffic flow, reducing congestion, and enhancing road safety in diverse geographical regions.

## APPENDIX - I

**Sample Code(BACKEND)**

pip install ultralytics # 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

sns.set(rc={'axes.facecolor': '#eae8fa'}, style='darkgrid') # Load a pretrained YOLOv8n model from Ultralytics

model = YOLO('yolov8n.pt')

from google.colab import files

# Upload the ZIP file uploaded = files.upload()

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

import os

# Replace 'your\_dataset.zip' with the name of your uploaded ZIP file zip\_file\_name = 'archive.zip'

# Create a directory to extract the contents (optional) extraction\_path = '/content/my\_dataset' os.makedirs(extraction\_path, exist\_ok=True)

# Extract the ZIP file

with zipfile.ZipFile(zip\_file\_name, 'r') as zip\_ref: zip\_ref.extractall(extraction\_path)

print("Files extracted to:", extraction\_path)

# Path to the image file - replace with the actual path to your image

image\_path = '/content/my\_dataset/Vehicle\_Detection\_Image\_Dataset/sample\_image.jpg'

# Check if the file exists

if not os.path.exists(image\_path):

# If not, print an error message and exit

print(f"Error: Image file not found at {image\_path}. Please check the path and try again.")

# Alternatively, you could provide a default image path: # image\_path = 'path/to/default/image.jpg'

else:

# 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)

# 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()

# Define the dataset\_path

dataset\_path = '/content/my\_dataset/Vehicle\_Detection\_Image\_Dataset'

# 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))

# Set paths for training and validation image sets

train\_images\_path = os.path.join(dataset\_path, 'train', 'images') valid\_images\_path = os.path.join(dataset\_path, 'valid', 'images')

# Initialize counters for the number of images num\_train\_images = 0

num\_valid\_images = 0

# Initialize sets to hold the unique sizes of images train\_image\_sizes = set()

valid\_image\_sizes = set()

# Check train images sizes and count

for filename in os.listdir(train\_images\_path): if filename.endswith('.jpg'):

num\_train\_images += 1

image\_path = os.path.join(train\_images\_path, filename) with Image.open(image\_path) as img:

train\_image\_sizes.add(img.size)

# Check validation images sizes and count

for filename in os.listdir(valid\_images\_path): if filename.endswith('.jpg'):

num\_valid\_images += 1

image\_path = os.path.join(valid\_images\_path, filename) with Image.open(image\_path) as img:

valid\_image\_sizes.add(img.size) # Print the results

print(f"Number of training images: {num\_train\_images}") print(f"Number of validation images: {num\_valid\_images}")

# Check if all images in training set have the same size if len(train\_image\_sizes) == 1:

print(f"All training images have the same size: {train\_image\_sizes.pop()}") else:

print("Training images have varying sizes.")

# Check if all images in validation set have the same size if len(valid\_image\_sizes) == 1:

print(f"All validation images have the same size: {valid\_image\_sizes.pop()}") else:

print("Validation images have varying sizes.") # List all jpg images in the directory

image\_files = [file for file in os.listdir(train\_images\_path) if file.endswith('.jpg')]

# Select 8 images at equal intervals num\_images = len(image\_files)

selected\_images = [image\_files[i] for i in range(0, num\_images, num\_images // 8)]

# Create a 2x4 subplot

fig, axes = plt.subplots(2, 4, figsize=(20, 11))

# Display each of the selected images

for ax, img\_file in zip(axes.ravel(), selected\_images): img\_path = os.path.join(train\_images\_path, img\_file) image = Image.open(img\_path)

ax.imshow(image) ax.axis('off')

plt.suptitle('Sample Images from Training Dataset', fontsize=20) plt.tight\_layout()

plt.show()

# Train the model on our custom dataset results = model.train(

data=yaml\_file\_path, epochs=100, imgsz=640,

device=0, patience=50, batch=32, optimizer='auto', lr0=0.0001,

lrf=0.1, dropout=0.1, seed=0

)

import os

# Define the path to the directory

post\_training\_files\_path = '/content/runs/detect/train' # Change this to your correct path

# List the files in the directory

files = os.listdir(post\_training\_files\_path) for file in files:

print(file) import pandas as pd

import matplotlib.pyplot as plt import seaborn as sns

# Define the function to plot learning curves

def plot\_learning\_curve(df, train\_loss\_col, val\_loss\_col, title): plt.figure(figsize=(12, 5))

sns.lineplot(data=df, x='epoch', y=train\_loss\_col, label='Train Loss',

color='#141140', linestyle='-', linewidth=2)

sns.lineplot(data=df, x='epoch', y=val\_loss\_col, label='Validation Loss', color='orangered', linestyle='--', linewidth=2)

plt.title(title) plt.xlabel('Epochs') plt.ylabel('Loss') plt.legend() plt.show()

# Example DataFrame structure data = {

'epoch': [1, 2, 3, 4, 5],

'train\_loss': [0.5, 0.4, 0.35, 0.3, 0.25],

'val\_loss': [0.6, 0.5, 0.45, 0.4, 0.35]

}

df = pd.DataFrame(data)

# Call the plotting function

plot\_learning\_curve(df, train\_loss\_col='train\_loss', val\_loss\_col='val\_loss', title='Learning Curve')

# 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, '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')

# Construct the path to the normalized confusion matrix image confusion\_matrix\_path = os.path.join(post\_training\_files\_path, 'confusion\_matrix\_normalized.png')

# Read the image using cv2

cm\_img = cv2.imread(confusion\_matrix\_path)

# Convert the image from BGR to RGB color space for accurate color representation with matplotlib

cm\_img = cv2.cvtColor(cm\_img, cv2.COLOR\_BGR2RGB)

# Display the image plt.figure(figsize=(10, 10), dpi=120) plt.imshow(cm\_img)

plt.axis('off') plt.show()

# Construct the path to the best model weights file using os.path.join best\_model\_path = os.path.join(post\_training\_files\_path, 'weights/best.pt')

# Load the best model weights into the YOLO model best\_model = YOLO(best\_model\_path)

# Validate the best model using the validation set with default parameters metrics = best\_model.val(split='val')

# Convert the dictionary to a pandas DataFrame and use the keys as the index metrics\_df = pd.DataFrame.from\_dict(metrics.results\_dict, orient='index', columns=['Metric Value'])

# Display the DataFrame metrics\_df.round(3)

# 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()

# Path to the image file sample\_image\_path =

'/content/my\_dataset/Vehicle\_Detection\_Image\_Dataset/sample\_image.jpg'

# Perform inference on the provided image using best model

results = best\_model.predict(source=sample\_image\_path, imgsz=640, conf=0.7)

# 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 Fine-tuned YOLOv8 Model', fontsize=20)

plt.axis('off') plt.show()

import shutil

# Define the path to the sample video in the dataset dataset\_video\_path = '/content/my\_dataset/Vehicle\_Detection\_Image\_Dataset/sample\_video.mp4'

# Define the destination path in the working directory video\_path = '/content/sample\_video.mp4' # Updated path

# Copy the video file from its original location in the dataset to the current working directory

shutil.copyfile(dataset\_video\_path, video\_path)

# Initiate vehicle detection on the sample video using the best performing model with streaming

results = best\_model.predict(source=video\_path, save=True, stream=True)

# Process results for r in results:

boxes = r.boxes # Get bounding box outputs

masks = r.masks # Get segmentation masks outputs (if applicable) probs = r.probs # Get class probabilities outputs (if applicable)

import os

import subprocess

from IPython.display import Video

# Set the locale to UTF-8 os.environ['LANG'] = 'en\_US.UTF-8' os.environ['LC\_ALL'] = 'en\_US.UTF-8'

# Convert the .avi video generated by the YOLOv8 prediction to .mp4 format subprocess.run(['ffmpeg', '-y', '-loglevel', 'panic', '-i', '/content/runs/detect/predict/sample\_video.avi', 'processed\_sample\_video.mp4'])

# Embed and display the processed sample video within the notebook Video("processed\_sample\_video.mp4", embed=True, width=960)

# Define the threshold for considering traffic as heavy heavy\_traffic\_threshold = 20

# 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('sample\_video.mp4')

# Define the codec and create VideoWriter object fourcc = cv2.VideoWriter\_fourcc(\*'XVID')

out = cv2.VideoWriter('traffic\_density\_analysis.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 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)

# Add a background rectangle for the left lane traffic intensity cv2.rectangle(processed\_frame, (intensity\_position\_left\_lane[0]-10,

intensity\_position\_left\_lane[1] - 25),

(intensity\_position\_left\_lane[0] + 460,

intensity\_position\_left\_lane[1] + 10), background\_color, -1)

# Add the traffic intensity text on top of the rectangle for the left lane cv2.putText(processed\_frame, f'Traffic Intensity: {traffic\_intensity\_left}',

intensity\_position\_left\_lane,

font, font\_scale, font\_color, 2, cv2.LINE\_AA)

# Add a background rectangle for the right lane vehicle count cv2.rectangle(processed\_frame, (text\_position\_right\_lane[0]-10,

text\_position\_right\_lane[1] - 25),

(text\_position\_right\_lane[0] + 460, text\_position\_right\_lane[1] + 10), background\_color, -1)

# Add the vehicle count text on top of the rectangle for the right lane cv2.putText(processed\_frame, f'Vehicles in Right Lane:

{vehicles\_in\_right\_lane}', text\_position\_right\_lane,

font, font\_scale, font\_color, 2, cv2.LINE\_AA)

# Add a background rectangle for the right lane traffic intensity cv2.rectangle(processed\_frame, (intensity\_position\_right\_lane[0]-10,

intensity\_position\_right\_lane[1] - 25),

(intensity\_position\_right\_lane[0] + 460,

intensity\_position\_right\_lane[1] + 10), background\_color, -1)

# Add the traffic intensity text on top of the rectangle for the right lane cv2.putText(processed\_frame, f'Traffic Intensity: {traffic\_intensity\_right}',

intensity\_position\_right\_lane,

font, font\_scale, font\_color, 2, cv2.LINE\_AA)

# Write the processed frame to the output video

out.write(processed\_frame)

# Uncomment the following 3 lines if running this code on a local machine to view the real-time processing results

# cv2.imshow('Real-time Analysis', processed\_frame)

# if cv2.waitKey(1) & 0xFF == ord('q'): # Press Q on keyboard to exit the loop # break

else:

break

# Release the video capture and video write objects cap.release()

out.release()

# Close all the frames

# cv2.destroyAllWindows() import os

import subprocess

from IPython.display import Video

# Set the locale to UTF-8 os.environ['LANG'] = 'en\_US.UTF-8' os.environ['LC\_ALL'] = 'en\_US.UTF-8'

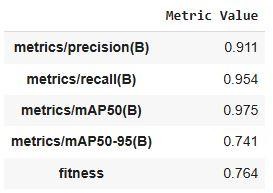
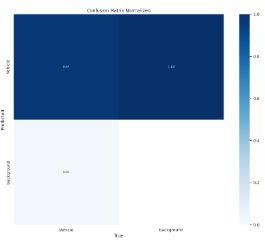
# Convert the .avi video generated by our traffic density estimation app to .mp4 format subprocess.run(['ffmpeg', '-y', '-loglevel', 'panic', '-i', '/content/traffic\_density\_analysis.avi', 'traffic\_density\_analysis.mp4'])

# Embed and display the processed sample video within the notebook Video("traffic\_density\_analysis.mp4", embed=True, width=960)

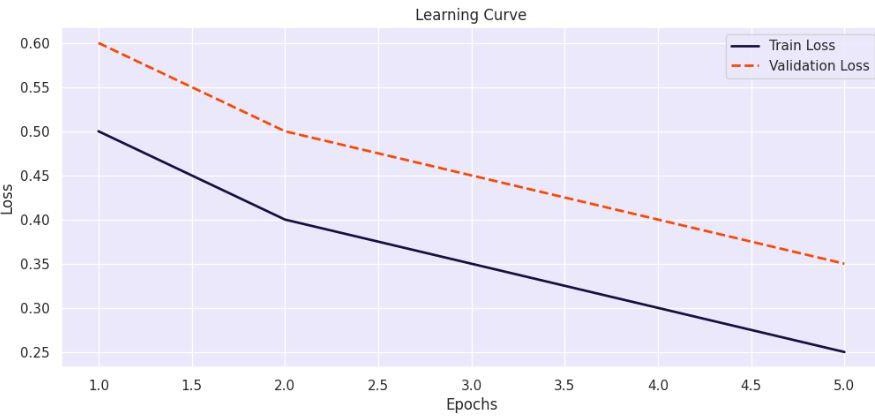
# Export the model best\_model.export(format='onnx')

APPENDIX II

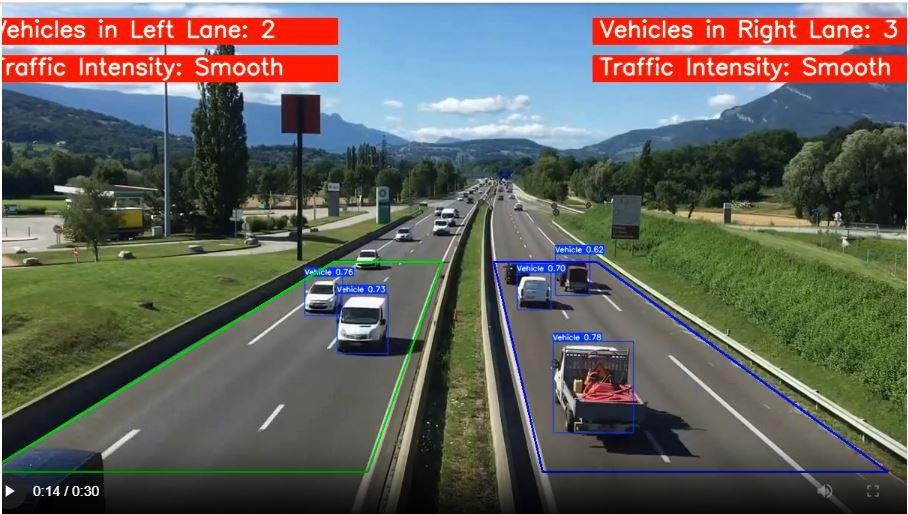
OUTPUT SCREENSHOTS

METRIC VALUE CONFUSION MATRIX



LEARNING CURVE



REAL-TIME TRAFFIC DENSITY DETECTION

## REFERENCES

1. Zhang, S., et al. "Density-Based Automatic Traffic Control Using Machine Learning." IEEE Conference Publication, 2023
2. Zhao, H., et al. "Real-Time Prediction of Traffic Density with Deep Learning Using Computer Vision and Traffic Event Information." IEEE Conference Publication, 2020.
3. Lee, T., et al. "Optimizing Traffic Light Control using YOLOv8 for Real-Time Vehicle Detection and Traffic Density." IEEE Conference Publication, 2024.
4. Wang, J., et al. "Smart City Traffic Monitoring: YOLOv7 Transfer Learning Approach for Real- Time Vehicle Detection." IEEE Conference Publication, 2023.
5. Prediction of Traffic Density Using YOLO Object Detection and Implemented in Raspberry Pi 3b

+ and Intel NCS 2 (IEEE, 2020)