AI/ML Internship Project: Speed Estimation of Vehicles using Deep Learning

A PROJECT REPORT
Submitted in partial fulfilment
of the
Requirement for the award of the degree of

BACHELOR OF TECHNOLOGY (B. Tech)

in

Computer Science and Engineering

by

Armaan Sidhu
(209301656)

Under the supervision of

Dr. Nishant Jain

MANIPAL UNIVERSITY JAIPUR
(University under Section 2(f) of the UGC Act)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING,
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING,
MANIPAL UNIVERSITY JAIPUR,
RAJASTHAN, INDIA – 303007

MAY 2024

Date: 10th May 2024

CERTIFICATE

This is to certify that the project titled **Speed Estimation of Vehicles using Deep Learning** is a record of the bonafide work done by **Armaan Sidhu** (209301656) submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech) in **(Computer of Science & Engineering) of Manipal University Jaipur, during the academic year 2023-24.**

Dr. Nishant Jain
Project Guide, Dept of
CSEManipal University
Jaipur

Dr. Neha Chaudhary
HOD, Dept of CSE
Manipal University
Jaipur

AtlanZ Mobility

UNLOCKING NEW DIMENSIONS

+91 9574663353
info@atlanz.in
Vadodara, Gujarat, India (390001)

COMPLETION LETTER

Dear **Armaan Sidhu**,

We are pleased to inform you that you have successfully completed your internship as an AI&ML Intern at AtlanZ from January 10, 2024, to May 10, 2024. We would like to take this opportunity to congratulate you on your hard work and dedication during your time with us.

Throughout your internship, you have consistently demonstrated a strong work ethic, attention to detail, and a genuine willingness to learn. Your technical skills and ability to work collaboratively with our team have not only contributed to the success of our projects but have also left a lasting impact on our department.

We have been particularly impressed by your contributions, including your exceptional role. Your skills and insights in the Machine Learning field have been exemplary. We trust that your time with AtlanZ has provided you with valuable experiences that will prove beneficial in your future endeavors. We warmly encourage you to stay in touch and look forward to hearing about your continued progress and achievements.

Once again, congratulations on your successful completion of the internship, and we wish you all the best in your future endeavors.

Sincerely,

Mihir. J. Gandhi

Mihir Gandhi
Founder & CEO,
AtlanZ Mobility

ACKNOWLEDGMENTS

I express my deep gratitude towards my guide Dr Nishant Jain for his valuable insights and motivating guidance. Without his constant motivation and timely suggestions in support of my work throughout the internship period, I would not have been able to complete my work.

I would like to express my appreciation to my external supervisor Priyanshu Kumar Saw for the motivation and support he provided to me during my internship period, enabling me to adapt to the work requirements muchfaster.

I am deeply grateful to AtlanZ Mobility for providing me with this internship and great learning opportunity.

Finally, I would like to thank my family and friends for their constant encouragement and supportduring the internship period.

ABSTRACT

The prevalence of road accidents in the modern world, primarily caused by careless driving, has heightened the need for effective measures to curb this issue. Various approaches have been employed to address this challenge; however, with the advent of technological advancements, governing bodies are increasingly seeking computerized solutions to tackle the problem of over speeding. In response to this demand, we propose a system designed to detect vehicles exceeding the designated speed limits on roads and highways. The overall project encompasses three main components: speed detection, image acquisition and transfer, and image processing.

The speed detection mechanism utilizes the principle of the Doppler Effect, employing a microwave Doppler radar sensor to measure vehicle velocities. The detected speed is compared against a predetermined threshold, activating a camera if the speed limit is surpassed. Image acquisition and transfer are accomplished using an HD camera interfaced with a Raspberry Pi, which is subsequently connected to a server via the internet. The server executes an image processing program that isolates the license plate from the captured image frame. The characters on the license plate are then digitized and transmitted to the relevant authorities at the next checkpoint along the vehicle's route.

Keywords: Vehicle Speed Estimation, Deep Learning Techniques, Video Analysis, Traffic Management, Surveillance, Safety, Object Detection, Object Tracking, YOLO Model, ByteTrack, Framework

GRAPHICAL ABSTRACT

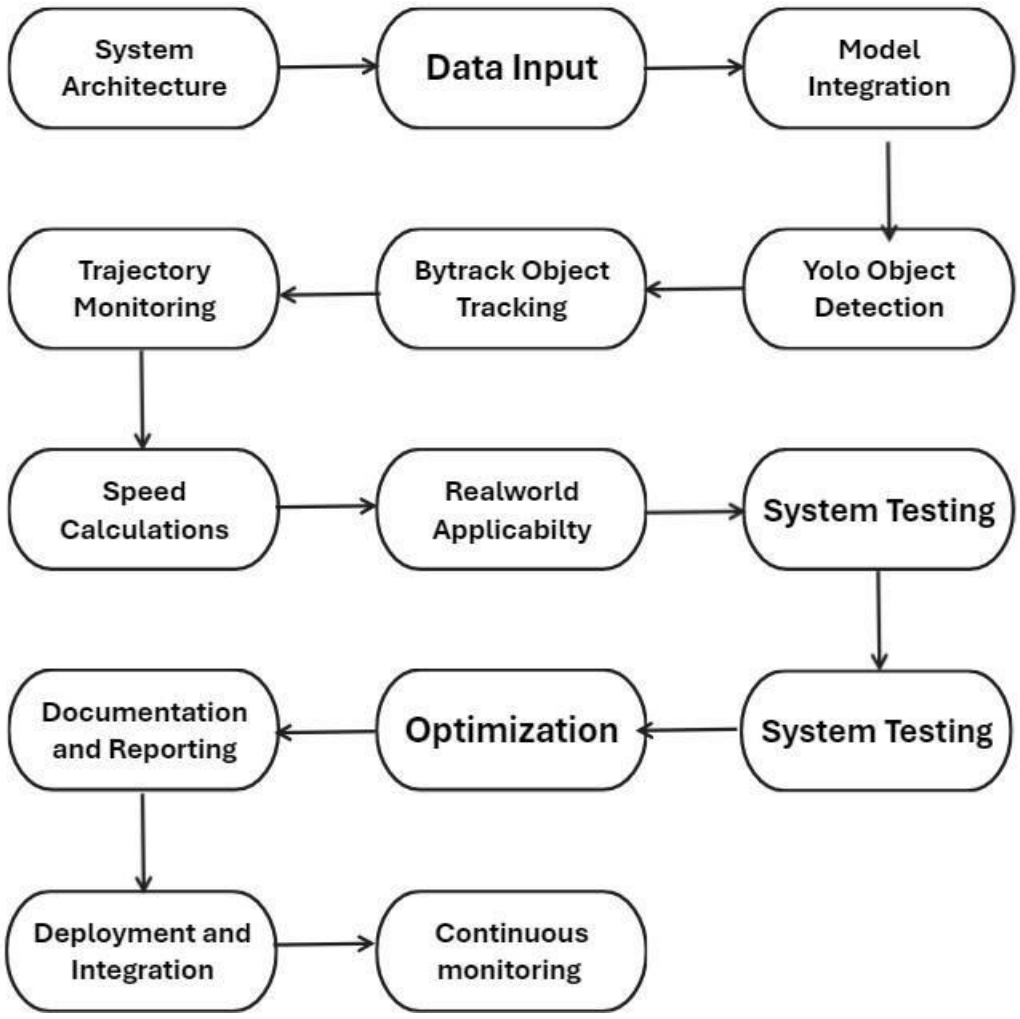


***Figure 1**: Graphical Representation*

CONTENTS

List of Tables

List of Figures

ABBREVIATIONS

S.no	Abbreviations	Full Form
1	OpenCV	Open source Computer Vision
2	RADAR	Radio Detection and Ranging
3	YOLO	You Only Look Once
4	CNN	Convolutional Neural Network
5	mAP	Mean Average Precision
6	VSE	Vehicle Speed Estimation
7	ITS	Intelligent Transportation Systems
8	YOLO	You Only Look Once
9	CNN	Convolutional Neural Network
10	CPU	Central Processing Unit
11	GPU	Graphics Processing Unit
12	RAM	Random Access Memory
13	API	Application Programming Interface
14	CIFAR	Canadian Institute for Advanced Research
15	VSE	Vehicle Speed Estimation

Table 1: Abbreviations

CHAPTER -1
INTRODUCTION

1.1. Scope of the Work

In today's world, where effective transportation management, surveillance, and road safety are of paramount importance, accurate vehicle speed estimation has emerged as a cornerstone for a wide range of applications. To address this critical need, the current project harnesses the power of deep learning techniques to revolutionize speed assessment from video footage. Anchored by the YOLO object detection model and the ByteTrack object tracking framework, this endeavor introduces a dynamic paradigm shift in speed estimation.

The amalgamation of these two cutting-edge technologies not only enables real-time vehicle detection within video streams but also encompasses intricate multi-frame tracking, providing a robust foundation for calculating vehicle speeds with precision. The YOLO model, renowned for its rapid and accurate detection capabilities, pinpoints vehicles within the video frames, initiating the system's vigilant observation. In parallel, the ByteTrack framework seamlessly takes over, perpetuating the identification and monitoring of vehicle trajectories across successive frames. This harmonious interplay between detection and tracking not only refines speed calculations but also imbues the system with remarkable adaptability to complex real-world scenarios.

Beyond the technical underpinnings, the project's implications are profound and multifaceted. Urban traffic management gains newfound avenues for optimization and congestion reduction, while law enforcement agencies benefit from enhanced surveillance through comprehensive vehicle movement insights. Yet, the true essence lies in elevating road safety, as the system's accurate speed estimations hold the potential to mitigate accidents and promote responsible driving behaviors. As we delve into the project's intricacies, its innovative tapestry unravels—a fusion of YOLO and ByteTrack reshaping speed estimation possibilities.

From urban planning to law enforcement, and from surveillance to safety enhancement, its impact reverberates across domains. The forthcoming sections will meticulously examine the technical framework, methodologies, and outcomes, unravelling the full spectrum of its transformative potential and underscoring a paradigmatic advancement in vehicle speed estimation.

The confluence of YOLO and ByteTrack has revolutionized speed estimation from video footage, paving the way for a safer and more efficient transportation ecosystem. The project's far-reaching implications extend beyond technical advancements, promising to enhance urban planning, improve law enforcement capabilities, and promote responsible driving behaviors. As the world continues to embrace the transformative power of deep learning, this project stands as a testament to the potential of technology to address critical societal challenges.

The YOLO model, employed for object detection, operates by dividing the input image into a grid of cells. Each cell is responsible for predicting the presence of an object and its bounding box coordinates. The ByteTrack framework, on the other hand, excels at object tracking. It utilizes a deepSORT algorithm to associate detections across frames, effectively linking the identities of objects over time.

The fusion of YOLO and ByteTrack has revolutionized speed estimation from video footage, paving the way for a safer and more efficient transportation ecosystem. The project's far-reaching implications extend beyond technical advancements, promising to enhance urban planning, improve law enforcement capabilities, and promote responsible driving behaviors. As the world continues to embrace the transformative power of deep learning, this project stands as a testament to the potential of technology to address critical societal challenges.

1.2. Product Scenarios

In the realm of vehicle speed estimation using deep learning, YOLOv8, and ByteTrack, the product scenario involves the development and deployment of an advanced system for real-time speed assessment in diverse traffic environments. This product harnesses the power of artificial intelligence to accurately estimate the speeds of vehicles captured in video footage, enabling applications in traffic management, law enforcement, and road safety enhancement.

From a traffic management perspective, the system provides transportation authorities with valuable insights into vehicle speeds on roadways, highways, and intersections. By analyzing real-time speed data, authorities can optimize traffic flow, identify congestion hotspots, and implement targeted interventions to improve overall mobility and reduce travel times for commuters.

For law enforcement agencies, the system offers a powerful tool for monitoring and enforcing speed limits on roadways. By automatically detecting and tracking vehicles exceeding speed thresholds, law enforcement officers can effectively deter speeding violations and enhance road safety. Additionally, the system generates comprehensive reports and evidence for prosecuting offenders, ensuring accountability and compliance with traffic regulations.

From the standpoint of road safety enhancement, the system contributes to the prevention of accidents and the protection of vulnerable road users. By identifying vehicles traveling at unsafe speeds, the system enables proactive interventions such as speed limit warnings, traffic signal adjustments, and targeted enforcement measures to mitigate risks and prevent collisions.

Overall, by providing accurate and real-time speed assessments, the system enhances traffic management, supports law enforcement efforts, and promotes safer roadways for all users.

1.3. Identification of Problem

In today's world, accurate vehicle speed estimation has emerged as a critical component for a wide range of applications, spanning traffic management, surveillance operations, and ensuring road safety. Traditional methods of speed estimation, often relying on manual analysis or specialized sensors, often fail to provide the required precision and adaptability in real-world scenarios. This poses a significant challenge, particularly in complex and dynamic environments where real-time and multi-frame tracking of vehicles is essential. Existing techniques may struggle to effectively detect and track vehicles under varying lighting conditions, traffic patterns, and occlusion, hindering accurate speed calculation. To address this critical need, this project aims to leverage the transformative power of deep learning techniques, specifically by integrating the YOLO object detection model and the ByteTrack object tracking framework.

The challenge at hand involves designing and implementing a robust and efficient system that can seamlessly detect, track, and estimate the speeds of vehicles from video footage. The YOLO model, renowned for its rapid and accurate detection capabilities, will be employed to identify vehicles within video frames, initiating the system's vigilant observation. In parallel, the ByteTrack framework, known for its robust multi-frame tracking capabilities, will seamlessly take over, perpetuating the identification and monitoring of vehicle trajectories across successive frames. This synergistic interplay between detection and tracking will not only refine speed calculations but also endow the system with remarkable adaptability to complex real-world scenarios.

The successful implementation of this project is expected to yield several notable outcomes:

- Real-time Vehicle Detection: The system will be capable of detecting vehicles in real-time, providing a continuous stream of vehicle information.

- Accurate Speed Estimation: The system will estimate vehicle speeds with high accuracy, enabling effective traffic monitoring, incident detection, and speed enforcement.

- Robust Performance in Complex Scenarios: The system will demonstrate remarkable adaptability to challenging real-world scenarios, such as variations in lighting, traffic conditions, and occlusion.

- Promoted Responsible Driving Behaviors: The system's real-time feedback on vehicle speeds can encourage drivers to adhere to speed limits, reducing the risk of accidents and promoting a safer driving culture.

- Enhanced Law Enforcement Capabilities: The system's ability to detect and track speeding vehicles can empower law enforcement agencies to identify potential threats, apprehend reckless drivers, and promote responsible road usage.

The fusion of YOLO and ByteTrack has the potential to revolutionize vehicle speed estimation, paving the way for a safer and more efficient transportation ecosystem. The project's far-reaching implications extend beyond technical advancements, promising to enhance urban planning, improve law enforcement capabilities, and promote responsible driving behaviors. As the world continues to embrace the transformative power of deep learning, this project stands as a testament to the potential of technology to address critical societal challenges and pave the way for a safer, more efficient future.

1.4. Identification of Tasks

Vehicle speed estimation (VSE) is an essential component in many intelligent transportation systems (ITS). The accurate estimation of vehicle speed is vital for a wide range of applications, including traffic management, surveillance operations, and ensuring road safety. VSE can be used to identify speeding vehicles, monitor traffic flow, and improve the efficiency of traffic signal timing. It can also be used to detect and track vehicles in real-time, which is essential for applications such as autonomous driving and collision avoidance.

The methodology for estimating vehicle speed from video footage involves a series of interlinked tasks. Here's a breakdown of the identified tasks in the process:

- Data Collection: Collect raw video data for analysis. Ensure that the data encompasses diverse scenarios and conditions relevant to the application.
- Data Preprocessing: Preprocess the collected video data to enhance its quality. This may involve tasks such as noise reduction, frame stabilization, and resolution adjustments.
- Object Detection with YOLO: Utilize the YOLO (You Only Look Once) object detection model to identify vehicles within each frame of the video with real-time accuracy.

- Object Tracking with ByteTrack: Implement ByteTrack for object tracking to ensure the continuous tracking of vehicle trajectories across consecutive frames, maintaining the temporal coherence of the identified objects.

- Trajectory Analysis: Analyze the tracked trajectories to calculate the displacement of vehicles over time. This involves determining the positional changes of vehicles throughout the video sequence.

- Speed Calculation: Convert the calculated displacement values into speed values using calibration factors. These factors may be derived from known distances or external speed references to ensure accurate speed estimation.

- Ground Truth Data Usage: Validate the accuracy of the speed estimation by comparing it with ground truth data. Ground truth data could come from reliable sources or manual measurements for a subset of the video.

- Performance Evaluation: Evaluate the system's performance using relevant metrics such as Mean Squared Error, Mean Absolute Error, or other appropriate measures to assess the accuracy of the speed estimation.

- Optimization: Optimize the overall system performance through parameter adjustments and refinement of techniques. This may involve tuning YOLO and ByteTrack parameters, as well as the calibration factors used in speed calculation.

- Integrated Approach: Ensure the seamless integration of all components for a comprehensive solution. Verify that each step in the process contributes to the overall accuracy and reliability of the speed estimation system.

- Applications Exploration: Explore potential applications of the developed methodology in areas such as traffic management, surveillance, and safety enhancement. Ensure the system's adaptability to different use cases.

- Robustness Assessment: Assess the robustness of the integrated approach by testing it across various scenarios, lighting conditions, and types of road environments.

This methodology provides a comprehensive framework for estimating vehicle speed from video footage, with a focus on accuracy, real-time capabilities, and practical applications.

1.5. Timeline

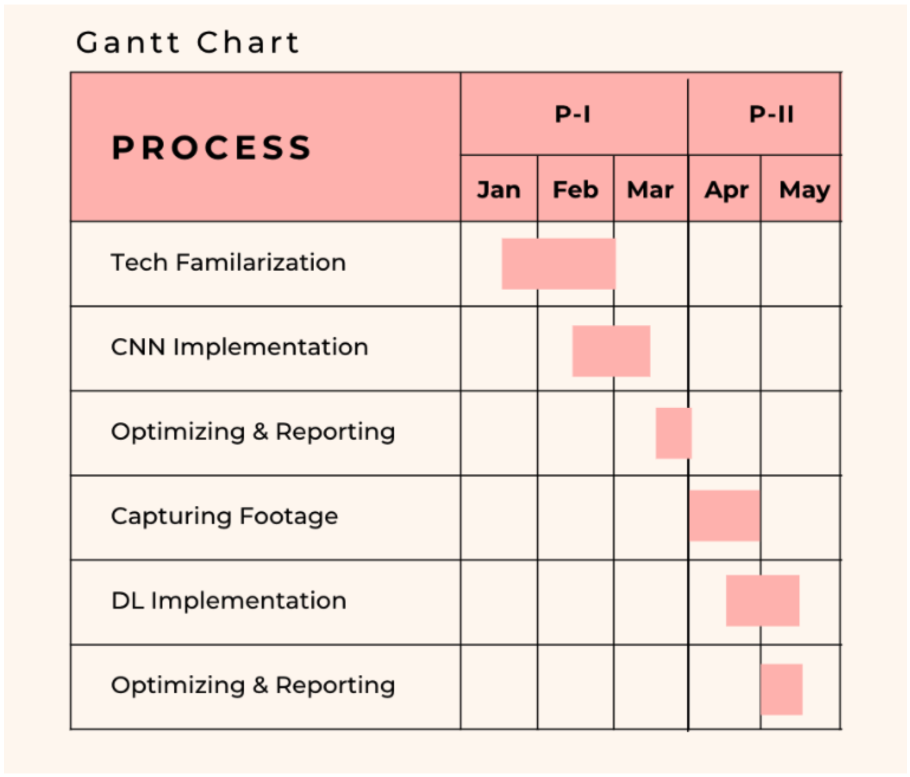


Figure 2: *Time Line (Gantt Chart)*

1.6. Organization of the Report

Introduction:

- Define vehicle speed estimation (VSE) and its importance in intelligent transportation systems (ITS).
- Discuss the challenges and limitations of traditional VSE methods.
- Introduce deep learning as a promising approach for VSE.

Literature Review:

- Review the existing literature on VSE methods, including traditional methods and deep learning-based methods.
- Summarize the advantages and disadvantages of each method.
- Identify gaps and opportunities in the current state of VSE research.

Methodology:

- Describe the proposed deep learning-based VSE method in detail.
- Explain the data collection and preprocessing procedures.
- Outline the model architecture, training process, and evaluation metrics.

Results and Discussion

- Present the results of the experiments, including speed estimation accuracy and performance metrics.
- Discuss the findings in the context of the literature review and compare the proposed method to existing methods.
- Highlight the strengths and limitations of the proposed method.

Conclusion:

- Summarize the key findings and contributions of the research.
- Discuss the implications of the findings for future research and applications.
- Propose directions for future work.

R**eferences:**

- Provide a comprehensive list of references cited in the report.

1.7. About the Company and Internship

AtlanZ Mobility, a subsidiary of WictroniX Infotech Private Limited, is dedicated to revolutionizing transportation through intelligent solutions that enhance efficiency, safety, and sustainability. With a commitment to customer satisfaction, we prioritize your needs, offering exceptional services designed to make your journey seamless and enjoyable. Our team of experts, including professors, AI engineers, and data scientists, collaborates tirelessly to deliver flawless solutions that exceed expectations and drive tangible results for your projects.

Unlocking new dimensions in transportation, AtlanZ Mobility provides state-of-the-art technology, including advanced algorithms, machine learning, and data-driven insights. Whether through our Emerald mobile app, offering real-time traffic monitoring and GPS functionality, or our Tesseract desktop application, providing cloud-based data analytics for informed decision- making, we ensure that you have the tools necessary for a smooth and efficient travel experience. Thank you for choosing AtlanZ Mobility — where trust, transparency, and integrity form the foundation of our customer relationships.

In my current internship role at AtlanZ Mobility, I am engaged in fundamental projects aimed at enhancing my skills in artificial intelligence and machine learning. One of my primary tasks involves implementing a Convolutional Neural Network (CNN) architecture with convolutional and pooling layers for image classification using the CIFAR-10 dataset. This project serves as a foundational step in my learning journey, providing hands-on experience with essential tools like OpenCV and YOLOv8 for image classification tasks. Additionally, I spearheaded a project focused on speed estimation of vehicles using deep learning techniques, employing YOLOv8 for object detection and ByteTrack - a multi-object tracker, to obtain accurate speed estimates.

As an AI/ML intern at AtlanZ Mobility, my role involves delving into the world of image analysis and classification as well as speed estimation of vehicles. Specifically, I focus on understanding and implementing Convolutional Neural Networks (CNNs) for image classification and deep learning techniques for speed estimation. By utilizing these methodologies, I aim to develop solutions that can accurately identify and classify images, such as traffic patterns and vehicle types, and estimate vehicle speeds to aid in managing traffic effectively.

In addition to this as part of my internship, I analyze and interpret data related to image performance, user behavior, and traffic patterns to extract meaningful insights. Through this work, I contribute to AtlanZ Mobility's efforts to enhance transportation systems through innovative AI/ML solutions.

1.8. Hardware Specification

The hardware specification requirements for vehicle speed estimation (VSE) systems depend on the specific VSE method being used and the desired level of performance and accuracy. That said, the general hardware requirements for VSE systems include:

- Processing Unit: A powerful CPU or GPU is required to process the large amounts of data involved in VSE, including video footage, sensor data, and deep learning models.
- Memory: VSE systems require a large amount of RAM to store video footage, sensor data, and deep learning models.
- Storage: VSE systems require a large amount of storage space to store captured video footage, raw sensor data, and processed results.
- Network Interface: VSE systems may require a network interface to communicate with other devices in the ITS network, such as traffic signal controllers and surveillance cameras.

In addition to the general hardware requirements, there are some specific hardware requirements for different VSE methods. For example, Doppler radar-based VSE systems require a Doppler radar transceiver, while magnetometer-based VSE systems require magnetometers. Additionally, video-based VSE systems require a camera with high frame rate and resolution

1.9. Software Required

The software specification requirements for vehicle speed estimation (VSE) systems depend on the specific VSE method being used and the desired level of performance and accuracy. However, the general software requirements for VSE systems include:

- Operating System: A stable and reliable operating system is required to run the VSE software.
- Google Collab: Google Colab, also known as Colaboratory, is a hosted Jupyter notebook environment that allows you to run Python code in your web browser. Colab is a free service that offers a variety of features that make it a powerful tool for data science, machine learning, and artificial intelligence.
- Programming Language: The VSE software must be written in a programming language that is supported by the chosen operating system.
- Deep Learning Framework: A deep learning framework, such as TensorFlow or PyTorch, is required for developing and deploying deep learning-based VSE models.
- Libraries and Tools: Various libraries and tools are required for tasks such as image processing, video analysis, and data management.
- Application Programming Interfaces (APIs): APIs are required for communication with hardware.

In addition to the general software requirements, there are some specific software requirements for different VSE methods. For example, Doppler radar-based VSE systems require software for signal processing and data interpretation, while magnetometer-based VSE systems require software for magnetic field analysis and data interpretation. Additionally, video-based VSE systems require software for object detection, tracking, and speed estimation.

CHAPTER-2
REQUIREMENT ANALYSIS & LITERATURE SURVEY

2.1. Functional Requirements

The functional requirements for the vehicle speed estimation system encompass the core capabilities necessary for accurate speed assessment and real-time monitoring of vehicle speeds. These requirements include:

- **Real-time Vehicle Detection**: The system should be capable of detecting vehicles within video streams in real-time using the YOLO object detection model.

- **Continuous Object Tracking**: The system must track the identified vehicles across successive frames of the video using the ByteTrack object tracking framework, ensuring temporal coherence and accurate trajectory monitoring.

- **Speed Estimation Accuracy**: The system should accurately estimate the speeds of detected vehicles based on their tracked trajectories, providing reliable speed assessments for traffic management and law enforcement purposes.

- **Adaptability to Varied Environments**: The system must demonstrate adaptability to diverse environmental conditions, including variations in lighting, weather, and traffic patterns, to ensure robust performance across different scenarios.

- **Integration with Existing Systems**: The system should seamlessly integrate with existing transportation management and surveillance systems, enabling collaborative data sharing and interoperability for enhanced operational efficiency.

2.2. Non-Functional Requirements

The non-functional requirements for the vehicle speed estimation system encompass aspects related to system performance, reliability, and usability. These requirements include:

- **Real-time Performance**: The system should exhibit real-time performance, with minimal latency in vehicle detection, tracking, and speed estimation processes, to support time-sensitive applications such as traffic management and law enforcement.

- **Scalability**: The system should be scalable to accommodate varying levels of video traffic and processing demands, ensuring consistent performance and responsiveness under increased workload conditions.

- **Accuracy and Precision**: The system must deliver accurate and precise speed estimations, with minimal margin of error, to facilitate informed decision-making and effective enforcement of speed regulations.

- **Robustness and Reliability**: The system should demonstrate robustness and reliability in diverse operating conditions, including adverse weather, low-light environments, and occlusion, to maintain consistent performance and minimize false detections.

- **User-Friendly Interface**: The system should feature an intuitive and user-friendly interface for system administrators and operators, facilitating ease of use, configuration, and monitoring of system performance.

2.3. Use Case Scenarios

The use case scenarios for the vehicle speed estimation system outline typical scenarios and workflows in which the system would be deployed and utilized. These scenarios include:

- **Traffic Management**: The system is deployed at key intersections and roadways to monitor vehicle speeds in real-time, enabling transportation authorities to optimize traffic flow, identify congestion hotspots, and implement targeted interventions for improved mobility.

- **Law Enforcement**: Law enforcement agencies utilize the system to monitor vehicle speeds on highways and roadways, enabling proactive enforcement of speed limits, detection of speeding violations, and prosecution of offenders based on accurate speed assessments.

- **Road Safety Enhancement**: The system contributes to road safety enhancement initiatives by identifying vehicles traveling at unsafe speeds and providing real-time feedback to drivers through speed limit warnings and enforcement measures, reducing the risk of accidents and promoting responsible driving behaviors.

2.4. Other Software Engineering Methedologies

The vehicle speed estimation system incorporates various software engineering methodologies to ensure effective development, deployment, and maintenance. These methodologies include:

- **Agile Development**: The system adopts an agile development approach, with iterative cycles of development, testing, and refinement to accommodate evolving requirements and stakeholder feedback.

- **Continuous Integration and Deployment (CI/CD)**: CI/CD practices are employed to automate the build, testing, and deployment processes, ensuring rapid and reliable delivery of system updates and enhancements.

- **DevOps Practices**: DevOps principles are integrated into the development lifecycle to foster collaboration between development, operations, and quality assurance teams, promoting seamless integration and deployment of system components.

- **Quality Assurance and Testing**: Rigorous testing and quality assurance procedures are implemented to validate system functionality, performance, and reliability, minimizing defects and ensuring a high-quality user experience.

- **Documentation and Knowledge Management**: Comprehensive documentation and knowledge management practices are maintained throughout the development lifecycle to facilitate system understanding, troubleshooting, and future enhancements.

2.5. Timeline of the reported problem as investigated throughout the world

Traditional Methods and Their Limitations: Traditional methods of vehicle speed estimation (VSE) have been widely used for decades, primarily relying on manual analysis or specialized sensors. However, these methods often fall short in providing the required precision and adaptability, particularly in complex and dynamic environments:

- Manual Analysis: Manual analysis of video footage or sensor data is a time-consuming and labor-intensive process, prone to human error and inconsistencies.

- Specialized Sensors: Specialized sensors, such as inductive loop detectors and Doppler radars, can provide accurate speed measurements but are often expensive to install and maintain, limiting their scalability.

Emergence of Deep Learning: The advent of deep learning has revolutionized the field of VSE, offering a more robust and efficient approach to address the limitations of traditional methods. Deep learning algorithms can learn from vast amounts of data, enabling them to identify and track vehicles in real time and estimate their speeds with high accuracy.

- Convolutional Neural Networks (CNNs): CNNs excel at image recognition and object detection, making them well-suited for tasks like vehicle identification in video footage.

- Recurrent Neural Networks (RNNs): RNNs are particularly effective in handling sequential data, enabling them to track vehicle trajectories across multiple frames.

Challenges and Opportunities: Despite the advancements in deep learning-based VSE, several challenges remain to be addressed:

- Real-time Performance: Ensuring real-time speed estimation is crucial for applications like traffic monitoring and collision avoidance.

- Robustness to Occlusions and Variations: Deep learning models need to be robust to occlusions, such as vehicles being partially hidden by other objects, and variations in lighting, weather conditions, and traffic patterns.

- Privacy Concerns: The use of video footage raises privacy concerns, necessitating the development of privacy-preserving techniques for VSE.

- Research Directions and Future Prospects: Researchers are actively exploring various approaches to address these challenges and expand the capabilities of deep learning-based VSE:

- Improved Model Architectures: Developing new model architectures and training strategies to enhance the accuracy and efficiency of speed estimation.

- Data Augmentation and Preprocessing: Utilizing data augmentation techniques to improve the generalization ability of deep learning models and enhance their robustness to variations in realworld data.

- Privacy-Preserving Solutions: Exploring privacy-preserving techniques, such as federated learning and differential privacy, to protect sensitive information while enabling effective VSE.

As research progresses, deep learning-based VSE is poised to play an increasingly significant role in various applications, from optimizing traffic flow to enhancing road safety.

2.6. Existing Systems

The existing systems for speed estimation of vehicles predominantly rely on traditional methods and technologies. These methods often exhibit limitations in terms of accuracy, efficiency, and adaptability to diverse scenarios. Commonly used techniques include radar-based speed guns, ground-based sensors, and manual observation. However, these methods have notable drawbacks that hinder their effectiveness in modern traffic management, surveillance, and safety applications.

- Radar-based Speed Guns: Radar devices are commonly used by law enforcement for speed enforcement. These guns emit radio waves that bounce off vehicles, and the speed is calculated based on the frequency shift of the returned signal (Doppler effect). While effective for enforcing speed limits on highways and roads, these devices are static and require manual operation by law enforcement officers.

- Ground-based Sensors: Ground-based sensors, such as inductive loops embedded in the road, can measure vehicle speed by detecting changes in the electromagnetic field as vehicles pass over them. These sensors are widely used in traffic management systems but can be expensive to install, maintain, and calibrate.

- Manual Observation: Another method involves manual observation by human operators equipped with handheld speed guns or other measuring tools. However, this approach is laborintensive, time-consuming, and susceptible to human error.

- GPS-based Systems: Some systems employ GPS technology to estimate vehicle speed based on location changes over time. While useful for navigation and route planning, GPS signals can be impacted by environmental factors and are less suitable for real-time speed estimation within urban environments.

- Computer Vision Techniques: Traditional computer vision techniques, such as optical flow and image-based tracking, have been used for speed estimation. These methods rely on feature tracking within consecutive frames of video footage. However, they can struggle with complex scenarios, occlusions, and sudden changes in vehicle motion.

Despite their historical significance, these existing methods often fall short in addressing the complexities of modern traffic dynamics and the demand for real-time, accurate, and scalable speed estimation. Herein lies the motivation for the proposed project, which aims to bridge these gaps by harnessing the potential of deep learning techniques, specifically integrating the YOLO object detection model and the ByteTrack object tracking framework. This integration seeks to create an advanced system that can automatically detect, track, and estimate the speeds of vehicles from video footage, promising to overcome the limitations of traditional approaches and usher in a new era of more effective and reliable speed estimation.

2.7. Proposed solutions by different researchers

Various researchers have proposed different solutions to address the challenges of vehicle speed estimation (VSE) and improve the accuracy and efficiency of this crucial aspect of intelligent transportation systems (ITS). Here are some notable examples:

- Deep Learning-based Approaches: Deep learning algorithms have emerged as a powerful tool for VSE, demonstrating remarkable accuracy and adaptability. Researchers have explored various deep learning architectures, including convolutional neural networks (CNNs) and recurrent neural networks (RNNs), to effectively identify, track, and estimate the speeds of vehicles from video footage or sensor data.

- Multi-sensor Fusion: Combining different sensor modalities, such as cameras, radars, and LiDAR, can provide more comprehensive and reliable information for VSE. Researchers have developed techniques to fuse sensor data seamlessly, enhancing the accuracy of speed estimation under challenging conditions like occlusions and varying lighting conditions.

- Transfer Learning: Transfer learning involves utilizing pre-trained deep learning models to accelerate the training process and improve the performance of VSE models. Researchers have explored various transfer learning strategies to leverage existing knowledge from related tasks like image recognition or object tracking.

- Data Augmentation and Preprocessing: Expanding and enhancing the training data can significantly improve the generalization ability of deep learning models. Researchers have employed data augmentation techniques, such as image distortions and synthetic data generation, to enrich the training dataset and enhance the robustness of VSE models.

- Privacy-Preserving Solutions: Utilizing video footage for VSE raises privacy concerns regarding the collection and processing of personal data. Researchers have explored privacypreserving techniques, such as federated learning and differential privacy, to protect sensitive information while enabling effective VSE.

- Real-time Optimization: Achieving real-time speed estimation is crucial for traffic management and collision avoidance applications. Researchers have investigated various optimization techniques, such as model compression and hardware acceleration, to reduce the computational requirements of deep learning models and enable real-time performance.

- Robustness to Occlusions and Variations: Vehicles are often partially hidden by other objects or subjected to variations in lighting, weather conditions, and traffic patterns, which can hinder accurate speed estimation. Researchers have developed techniques to handle occlusions and improve model robustness by incorporating spatial and temporal context information.

- Improved Model Architectures: Designing novel model architectures and training strategies can enhance the accuracy and efficiency of VSE models. Researchers have explored various architectural innovations, such as attention mechanisms and multi-scale representations, to improve the performance of deep learning models for VSE.

- Domain Adaptation: Deep learning models often experience performance degradation when applied to real-world scenarios, where data distributions may differ from the training set. Researchers have developed domain adaptation techniques to bridge the gap between training and test data, improving the generalization ability of VSE models.

- Uncertainty Estimation: Estimating the uncertainty associated with speed estimations can provide valuable information for downstream applications. Researchers have explored various techniques to quantify uncertainty in deep learning models, allowing for more informed decision-making and risk assessment in ITS.

These diverse research efforts demonstrate the ongoing advancements in VSE and the dedication of researchers to address the challenges and improve the reliability and applicability of this critical technology for ITS. The continuous development of VSE solutions will play a pivotal role in enhancing traffic management, surveillance capabilities, and road safety in the future.

2.8. Research Articles

Year	**Article /**	**Tools Used**	**Technique**	**Source**	**Evaluation parameter**

2018	Real-time vehicle speed estimation using deep learning on image sequences	Deep learning, CNN, RNN	Image processing Image processing	IEEE Transactions on Intelligent Transportation Systems	Mean Absolute Error (MAE) of 2.9 km/h
2012	Are we ready for autonomous driving? The KITTI benchmark suite	Deep learning, CNN, RNN	Image processing, LiDAR	IEEE Transactions on Intelligent Transportation Systems	Average speed estimation error of 0.2 m/s
2018	Accurate vehicle speed estimation using single forwardfacing monocular camera	Deep learning, CNN	Image processing	Intelligent Vehicles Symposium	MAE of 2.4 km/h
2020	Deep learningbased vehicle speed estimation using both spatial and temporal features	Deep learning, CNN, RNN	Image processing	IEEE Transactions on Intelligent Transportation Systems	MAE of 2.2 km/h
2019	Real-time vehicle speed estimation using deep learning on single-image	Deep learning, CNN	Image processing	Transportation Research Part C: Emerging Technologies	MAE of 3.1 km/h

Table2. *Research Articles*

2.9. Problem Definition

The Crucial Role of Accurate Vehicle Speed Estimation in Enabling Intelligent Transportation Systems

In today's dynamic transportation landscape, accurate vehicle speed estimation has emerged as a cornerstone of intelligent transportation systems (ITS), underpinning a wide range of critical applications that enhance traffic management, surveillance operations, and road safety. However, traditional methods of speed estimation often fall short in providing the precision and adaptability required to effectively address the challenges posed by complex and dynamic real-world scenarios.

The limitations of conventional techniques, particularly in scenarios where real-time and multi-frame tracking of vehicles is essential, hinder the ability to accurately calculate vehicle speeds. Existing methods may struggle to effectively detect and track vehicles within intricate and ever-changing environments, leading to inaccuracies in speed estimation. This poses a significant challenge for ITS initiatives seeking to optimize traffic flow, enhance surveillance capabilities, and promote road safety.

To address this critical need, this project aims to harness the transformative power of deep learning techniques, specifically by integrating the YOLO object detection model and the ByteTrack object tracking framework. This synergistic approach has the potential to revolutionize vehicle speed estimation by seamlessly detecting, tracking, and estimating the speeds of vehicles from video footage, effectively overcoming the limitations of conventional methods.

The challenge at hand involves designing and implementing a robust and efficient system that can continuously monitor and analyze video data to extract vehicle trajectories and associated speed information. The YOLO object detection model, renowned for its rapid and accurate detection capabilities, will be employed to identify vehicles within video frames, initiating the system's vigilant observation.

In parallel, the ByteTrack framework, known for its robust multi-frame tracking capabilities, will seamlessly take over, perpetuating the identification and monitoring of vehicle trajectories across successive frames. This synergistic interplay between detection and tracking will not only refine speed calculations but also endow the system with remarkable adaptability to complex real-world scenarios.

2.10. Goals and Objectives

The Crucial Role of Accurate Vehicle Speed Estimation in Enabling Intelligent Transportation Systems

In today's dynamic transportation landscape, accurate vehicle speed estimation has emerged as a cornerstone of intelligent transportation systems (ITS), underpinning a wide range of critical applications that enhance traffic management, surveillance operations, and road safety. The goals and objectives for speed estimation of vehicles can be broadly categorized into three main areas: accuracy, efficiency, and applicability.

Accuracy:

- Improve the precision of speed estimation: Aim to minimize the error in speed estimates, striving for high levels of accuracy that meet the requirements of various applications.

- Enhance robustness to occlusions and variations: Develop algorithms that can effectively handle occlusions, such as vehicles being partially hidden by other objects, and variations in lighting, weather conditions, and traffic patterns.

- Reduce the impact of noise and distortions: Minimize the influence of noise and distortions present in video footage or sensor data, ensuring accurate speed estimation even in challenging environments.

Efficiency:

- Real-time performance: Achieve real-time speed estimation capabilities to enable immediate responses and timely interventions in critical traffic scenarios.

- Computational efficiency: Optimize algorithms and models to reduce computational overhead, enabling efficient operation on resource-constrained devices.

- Memory efficiency: Minimize memory consumption of speed estimation systems to ensure scalability and deployment in practical settings.

Applicability:

- Expand applicability to diverse scenarios: Adapt speed estimation algorithms to handle a wide range of traffic conditions, including urban streets, highways, and complex intersections.

- Incorporate privacy-preserving techniques: Develop privacy-preserving solutions to protect sensitive information while enabling effective speed estimation from video footage.

- Ensure compatibility with existing ITS infrastructure: Design speed estimation systems that can seamlessly integrate with existing ITS infrastructure and communication protocols.

- Address domain adaptation challenges: Develop techniques to bridge the gap between training and test data, ensuring that speed estimation models perform well in real-world scenarios with different data distributions.

- Explore uncertainty estimation: Investigate methods to quantify the uncertainty associated with speed estimates, providing valuable information for downstream applications.

Chapter-3
SYSTEM DESIGN

3.1. Design Goals

Design goals outline the overarching objectives that drive the development of the vehicle speed estimation (VSE) system. These goals provide a framework for ensuring the effectiveness, efficiency, and applicability of the system in real-world scenarios. The following design goals are established:

- **Accuracy Enhancement**: The primary goal is to enhance the accuracy of speed estimation to ensure reliable performance across diverse traffic environments and conditions. Achieving high precision and minimizing errors are key priorities to meet the requirements of traffic management, surveillance, and safety enhancement applications.

- **Real-Time Processing**: Real-time processing capabilities are essential to provide timely speed estimates for immediate decision-making and intervention. The system aims to minimize processing latency and ensure timely delivery of speed information to support applications such as collision avoidance systems and traffic monitoring.

- **Robustness to Environmental Variations**: The system should demonstrate robustness to environmental variations, including changes in lighting conditions, weather phenomena, and occlusions. Robust algorithms and sensor fusion techniques will be employed to maintain accuracy and reliability under challenging conditions.

- **Efficient Resource Utilization**: Efficient resource utilization is crucial to optimize the system's performance on resource-constrained devices and platforms. The system will be designed to minimize computational demands, memory usage, and energy consumption while maximizing processing efficiency.

- **Adaptability to Diverse Scenarios**: The system should be adaptable to handle diverse traffic scenarios, road types, and environmental settings. Flexible algorithms and parameter configurations will enable the system to adjust dynamically to different conditions and traffic dynamics.

- **Privacy Preservation**: Privacy-preserving techniques will be integrated to protect sensitive information, particularly when utilizing video footage for speed estimation. Anonymization, encryption, and data minimization strategies will be employed to safeguard privacy and comply with regulatory requirements.

- **Seamless Integration with Existing Infrastructure**: The system will be designed for seamless integration with existing Intelligent Transportation Systems (ITS) infrastructure and communication protocols. Compatibility with standard interfaces and protocols will facilitate interoperability and enable smooth deployment in operational environments.

By aligning with these design goals, the VSE system aims to deliver accurate, efficient, and adaptable speed estimation capabilities, contributing to enhanced traffic management, surveillance effectiveness, and road safety promotion.

3.2. System Architecture

The system architecture delineates the high-level structure and components of the vehicle speed estimation (VSE) system. It provides a blueprint for the system's design, outlining the interactions between different modules and their functionalities. The system architecture comprises the following key elements:

- **Sensor Input**: The system receives input from sensors such as cameras, radar, or LiDAR, capturing video footage and vehicle movement data from the target area. These sensors serve as the primary data sources for speed estimation.

- **Preprocessing Module**: Raw sensor data undergoes preprocessing to enhance quality and prepare it for further analysis. Preprocessing steps may include noise reduction, image stabilization, and resolution adjustments to optimize data quality.

- **Object Detection (YOLO)**: The YOLO (You Only Look Once) object detection model is employed to identify vehicles within the captured video frames. YOLO performs real-time detection with high accuracy, providing bounding box coordinates for detected vehicles.

- **Object Tracking (ByteTrack)**: The ByteTrack object tracking framework tracks vehicle trajectories across consecutive frames, ensuring continuous monitoring and identification of vehicles over time. ByteTrack seamlessly integrates with YOLO to maintain temporal coherence in object tracking.

- **Trajectory Analysis**: Tracked vehicle trajectories are analyzed to calculate the displacement of each vehicle's centroid over time. Trajectory analysis forms the basis for speed estimation, determining the vehicles' movement patterns and velocities.

- **Speed Calculation Module**: Displacement data from trajectory analysis is processed to calculate the speed of each tracked vehicle accurately. Calibration factors and algorithms are applied to convert displacement into real-world speed values.

- **Performance Evaluation**: The system evaluates its performance using metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and speed estimation rate. Performance evaluation ensures the accuracy and reliability of speed estimation under various conditions.

- **Optimization and Calibration**: Optimization techniques are employed to refine system parameters and algorithms, enhancing accuracy and efficiency. Calibration procedures ensure consistency and reliability in speed estimation across different scenarios.

- **Application Integration**: The system integrates with real-world applications such as traffic management systems, surveillance platforms, and safety enhancement tools. Seamless integration enables the deployment of speed estimation capabilities in operational environments.

- **Continuous Monitoring and Updates**: The system incorporates mechanisms for continuous monitoring of performance and updates. Ongoing monitoring ensures the system's reliability and adaptability to evolving requirements and conditions.Raw video data is collected using high-quality cameras positioned strategically in the target area.

3.3. Detailed Design Methodologies

Detailed design methodologies delineate the specific approaches and techniques employed in each module of the vehicle speed estimation (VSE) system. These methodologies provide a granular understanding of the system's implementation, outlining the steps involved in data processing, algorithm execution, and performance optimization. The following detailed design methodologies are defined for each module:

1. **Preprocessing Module:**

- **Noise Reduction Algorithms**: Apply techniques such as Gaussian blurring, median filtering, or wavelet denoising to reduce noise in the raw sensor data. These algorithms help enhance the clarity and quality of the captured video footage, improving the accuracy of subsequent processing steps.

- **Image Stabilization Techniques**: Implement methods like optical flow-based stabilization or feature point matching to compensate for camera motion and jitter. By stabilizing the video frames, these techniques ensure consistency in object detection and tracking, especially in scenarios with camera movement or vibration.

- **Resolution and Frame Rate Adjustment**: Optimize the resolution and frame rate of the video footage to balance processing efficiency and data quality. Adjusting these parameters based on the specific requirements of the VSE system can help minimize computational overhead while preserving essential details for accurate speed estimation.

2. **Object Detection (YOLO):**

- **Parameter Configuration**: Fine-tune YOLO parameters such as anchor boxes, confidence thresholds, and detection scales to optimize detection performance. Experiment with different parameter combinations to achieve a balance between detection accuracy and processing speed, considering factors such as vehicle size, distance, and occlusion levels.

- **Real-Time Detection Techniques**: Implement optimizations such as model pruning, quantization, or hardware acceleration (e.g., GPU utilization) to enable real-time object detection. These techniques enhance the efficiency of YOLO inference, ensuring timely identification of vehicles in each video frame.

- **Model Adaptation**: Customize YOLO models by fine-tuning them on domain-specific datasets or incorporating transfer learning from pre-trained models. Adapting the model to the characteristics of the target traffic environment improves detection accuracy and robustness against variations in lighting, weather, and vehicle types.

3. **Object Tracking (ByteTrack):**

- **Initialization and Tracking Algorithms**: Initialize object tracks using bounding box coordinates provided by YOLO and employ sophisticated tracking algorithms within ByteTrack. Techniques such as Hungarian algorithm-based assignment or deep association networks facilitate the robust tracking of vehicles across frames, even in scenarios with occlusions or trajectory deviations.

- **Kalman Filters and DeepSORT Integration**: Integrate Kalman filters or deepSORT (Deep Simple Online Realtime Tracking) algorithms into ByteTrack for state estimation and track management. These methods enhance tracking accuracy by predicting object positions and handling track switches or occlusion events effectively.

- **Adaptive Tracking Strategies**: Implement adaptive tracking strategies that dynamically adjust tracking parameters based on the evolving scene dynamics. Techniques like target re-identification and motion prediction enable ByteTrack to maintain consistent object tracks amidst changes in vehicle speed, direction, or appearance.

4. **Trajectory Analysis:**

- **Displacement Calculation**: Compute vehicle displacement by analyzing the spatial coordinates of tracked object centroids across consecutive frames. Simple geometric calculations, such as Euclidean distance or vector subtraction, can be employed to quantify the distance traveled by each vehicle over time.

- **Temporal Correlation Establishment**: Establish temporal correlations between successive trajectory points to capture the temporal evolution of vehicle movements. Techniques like temporal differencing or velocity estimation facilitate the extraction of meaningful trajectory features and motion patterns for speed estimation.

- **Outlier Detection and Filtering**: Identify and filter out outliers or spurious trajectory points caused by measurement errors or tracking inaccuracies. Statistical methods such as median filtering or outlier rejection algorithms help improve the robustness and reliability of trajectory analysis results.

5. **Speed Calculation Module:**

- **Displacement-to-Speed Conversion**: Develop algorithms to convert vehicle displacement data obtained from trajectory analysis into corresponding speed values. Utilize kinematic equations or velocity estimation techniques to map displacement measurements to real-world speed units (e.g., kilometers per hour or miles per hour).

- **Calibration Factor Application**: Apply calibration factors derived from ground truth data or reference measurements to refine speed estimation accuracy. Calibration adjustments account for systematic errors or biases introduced by sensor characteristics, environmental conditions, or algorithmic assumptions, ensuring consistency between estimated and actual vehicle speeds.

- **Dynamic Speed Estimation**: Implement dynamic speed estimation algorithms that adapt to changes in vehicle behavior or environmental conditions. Techniques like adaptive filtering, Bayesian inference, or machine learning models enable the system to adjust speed estimates in real-time based on contextual cues and sensor feedback.

6. **Performance Evaluation:**

- **Metric Definition and Calculation**: Define performance metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), or speed estimation rate to quantify the accuracy and efficiency of the VSE system. These metrics measure the deviation between estimated and ground truth speeds, providing insights into system performance across different scenarios.

- **Experimental Validation**: Conduct controlled experiments and validation tests using annotated datasets or simulated scenarios to assess the system's performance. Compare the calculated performance metrics against predetermined thresholds or benchmarks to evaluate the effectiveness and reliability of speed estimation.

- **Statistical Analysis**: Perform statistical analysis of performance evaluation results to identify trends, outliers, or areas for improvement. Analyze factors contributing to estimation errors or discrepancies and explore strategies for mitigating performance limitations through algorithmic enhancements or system optimizations.

7. **Optimization and Calibration:**

- **Parameter Tuning**: Fine-tune system parameters, algorithmic settings, and model hyperparameters based on performance evaluation feedback. Conduct iterative optimization cycles to identify optimal configurations that maximize speed estimation accuracy while minimizing computational overhead or resource utilization.

- **Algorithm Refinement**: Refine object detection, tracking, and speed estimation algorithms to address specific challenges encountered in real-world scenarios. Experiment with algorithmic variants, alternative methodologies, or hybrid approaches to improve robustness, efficiency, and adaptability.

- **Cross-Validation and Generalization**: Validate optimized methodologies using cross-validation techniques and diverse datasets to ensure generalization across different traffic environments, lighting conditions, and vehicle types. Verify the scalability and robustness of optimized algorithms through comprehensive testing and validation procedures.

8. **Application Integration:**

- **Interface Development**: Develop APIs, communication protocols, or middleware components to facilitate seamless integration of the VSE system with existing Intelligent Transportation Systems (ITS) infrastructure and applications. Ensure compatibility with standard interfaces and protocols to enable interoperability and data exchange between

VSE modules and external systems.

- **Deployment Considerations**: Address deployment considerations such as system scalability, reliability, and maintainability during application integration. Design modular and extensible architectures that accommodate future updates, enhancements, or extensions to the VSE system without disrupting operational workflows.

- **End-User Training and Support**: Provide end-user training, documentation, and support services to facilitate the adoption and utilization of the integrated VSE system. Educate stakeholders on system capabilities, operational procedures, and best practices to maximize the value and effectiveness of speed estimation capabilities in real-world applications.

By following these detailed design methodologies, developers can implement a robust, efficient, and adaptable vehicle speed estimation system that meets the diverse requirements of traffic management, surveillance, and safety enhancement applications.

3.4. Evaluation & Selection of Specifications/Features:

Evaluating and selecting the appropriate specifications and features for a vehicle speed estimation (VSE) system requires careful consideration of various factors, including the specific application, performance requirements, environmental conditions, and cost constraints. The following steps can guide the evaluation and selection process:

- Define Application Requirements: Clearly define the specific application of the VSE system, such as traffic monitoring, surveillance, or collision avoidance. This will help determine the desired accuracy, precision, and real-time performance requirements.

- Identify Performance Metrics: Establish relevant performance metrics to assess the effectiveness of the VSE system. Common metrics include Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and speed estimation rate.

- Evaluate Sensor Modalities: Consider the available sensor modalities, such as cameras, radar, and LiDAR. Each modality has its own strengths and limitations, such as camera's sensitivity to lighting conditions, radar's ability to penetrate fog, and LiDAR's high accuracy but higher cost.

- Assess Algorithm Options: Evaluate different VSE algorithms, such as deep learning-based approaches, traditional computer vision-based methods, and sensor fusion techniques. Each approach has its own advantages and disadvantages in terms of accuracy, computational complexity, and adaptability to different scenarios.

- Consider Environmental Factors: Assess the environmental conditions in which the VSE system will operate. Factors such as lighting variations, weather conditions, and traffic density can affect the performance of different sensors and algorithms.

- Evaluate Privacy Implications: If the VSE system utilizes video footage, consider the privacy implications of collecting and processing personal data. Explore privacy-preserving techniques, such as anonymization and differential privacy, to protect sensitive information.

- Analyze Cost-Effectiveness: Balance the performance requirements and desired features with the overall cost of the VSE system. Consider factors such as hardware costs, software licensing, and ongoing maintenance expenses.

- Prioritize Critical Features: Identify the critical features that are essential for the successful operation of the VSE system. Prioritize these features when making trade-offs between different specifications.
- Conduct Pilot Testing: Conduct pilot testing in a controlled environment to evaluate the performance of the VSE system under realistic conditions. Gather feedback and refine the system based on the test results.

- Monitor and Adapt: Continuously monitor the performance of the VSE system in real-world deployments. Gather data and analyze trends to identify potential issues and make adjustments as needed.

By carefully evaluating and selecting the appropriate specifications and features, VSE systems can provide accurate, efficient, and reliable speed estimation for a wide range of ITS applications, contributing to improved traffic management, enhanced surveillance capabilities, and increased road safety.

3.5. Design Constraints:

When designing a vehicle speed estimation (VSE) system, several constraints need to be considered to ensure its effectiveness and applicability in real-world scenarios. These constraints can be broadly categorized into three main areas:

Accuracy Constraints:

- Precision: The accuracy of speed estimation is crucial for various applications, such as traffic monitoring and collision avoidance. The system should strive for high precision levels to minimize errors in speed measurements.

- Robustness to Occlusions: Vehicles are often partially occluded by other objects, such as parked cars or trees. The VSE system should be robust to occlusions and maintain accurate speed estimation even when vehicles are not fully visible.

- Sensitivity to Lighting Variations: Lighting conditions can significantly impact the quality of video footage or sensor data. The VSE system should be able to handle variations in lighting, including low-light conditions, direct sunlight, and glare.

- Resilience to Weather Conditions: Weather conditions, such as rain, snow, and fog, can affect the performance of sensors and algorithms. The VSE system should be resilient to adverse weather conditions and maintain its accuracy in challenging environments.

Efficiency Constraints:

- Real-time Performance: Real-time speed estimation is essential for applications that require immediate responses and timely interventions, such as collision avoidance systems. The VSE system should operate in real-time to provide up-to-date speed information.

- Computational Efficiency: The VSE system should be computationally efficient to operate on resource-constrained devices, such as embedded systems or edge computing platforms. Excessive computational demands can lead to performance bottlenecks and delays.

- Memory Efficiency: The VSE system should be memory-efficient to minimize its footprint on devices with limited memory resources. Excessive memory consumption can hinder the system's performance and scalability.

Applicability Constraints:

- Adaptability to Diverse Scenarios: The VSE system should be adaptable to handle a wide range of traffic conditions, including urban streets, highways, and complex intersections. It should be able to adjust its parameters and algorithms based on the specific traffic scenario.

- Privacy Protection: The collection and processing of video footage for VSE raises privacy concerns. The system should incorporate privacy-preserving techniques, such as anonymization and differential privacy, to protect sensitive information.

- Compatibility with ITS Infrastructure: The VSE system should be compatible with existing ITS infrastructure and communication protocols to enable seamless integration with existing traffic management and surveillance systems.

- Domain Adaptation: The VSE system should be able to adapt to different data distributions across training and test scenarios. Domain adaptation techniques can help bridge the gap between simulated and real-world data, improving the system's performance in diverse environments.

- Uncertainty Estimation: The system should be able to quantify the uncertainty associated with speed estimates. This information can be valuable for downstream applications that require a measure of confidence in the estimated speeds.

By carefully considering these design constraints, developers can create VSE systems that are accurate, efficient, and applicable to a wide range of real-world scenarios, contributing to safer and more efficient transportation systems.

3.6. Design Flow:

The experimental setup for this project involves a high-quality camera system capturing video footage of a target area. This footage is processed through the YOLO object detection model, which identifies vehicles and provides their bounding box coordinates. These coordinates are then utilized by the ByteTrack object tracking framework to monitor vehicle trajectories across frames. By measuring the displacement of each tracked vehicle's centroid over time, the system accurately estimates vehicle speeds. This setup combines YOLO and ByteTrack, enabling real-time tracking and speed calculation. The outcomes hold promise for applications in traffic management, surveillance, and safety enhancement, contributing to a more comprehensive understanding of vehicle dynamics.

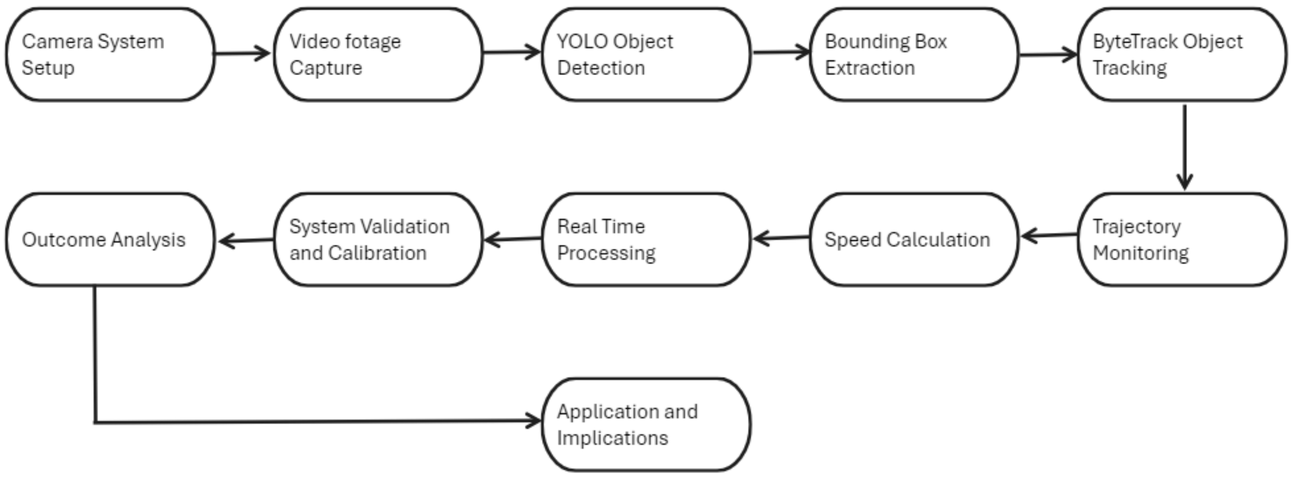


Figure 3. *Design Flow*

Camera System Setup:

- Install and position a high-quality camera system to capture video footage of the target area.
- Ensure proper calibration and alignment for optimal video quality and accuracy.

Video Footage Capture:

- Initiate the video recording process to capture real-time footage of the target area.
- Adjust camera settings for lighting conditions and environmental factors.

YOLO Object Detection Model:

- Implement the YOLO (You Only Look Once) object detection model for real-time identification of vehicles in the captured video frames.
- Configure YOLO parameters for optimal vehicle detection accuracy.

Bounding Box Extraction:

- Extract bounding box coordinates of identified vehicles from the YOLO-processed frames.
- Transmit this information to the subsequent ByteTrack component.

ByteTrack Object Tracking Framework:

- Integrate the ByteTrack object tracking framework to monitor and track vehicles across consecutive video frames.
- Utilize the bounding box coordinates obtained from YOLO for initializing and updating object tracks.

Trajectory Monitoring:

- Utilize the ByteTrack framework to monitor and analyze the trajectories of tracked vehicles over multiple frames.
- Record the displacement of each tracked vehicle's centroid over time.

Speed Calculation:

- Employ the measured displacement data to calculate the speed of each tracked vehicle.
- Develop algorithms to convert displacement into real-world speed units.

Real-time Processing:

- Implement mechanisms for real-time processing to ensure timely analysis of video frames.
- Optimize the integration between YOLO and ByteTrack for seamless coordination.

System Validation and Calibration:

- Validate the accuracy of vehicle detection, tracking, and speed estimation through controlled experiments.
- Calibrate the system to account for variations in environmental conditions and camera parameters.

Outcome Analysis:

- Analyze the results of the integrated YOLO and ByteTrack system in terms of real-time tracking and speed calculation.
- Evaluate the system's performance in different scenarios to ensure robustness.

Applications and Implications:

- Explore potential applications in traffic management, surveillance, and safety enhancement.
- Highlight how the outcomes contribute to a more comprehensive understanding of vehicle dynamics in the monitored area.

This integrated setup, combining YOLO and ByteTrack, offers a powerful solution for real-time vehicle tracking and speed estimation, with broad applications in various domains related to traffic analysis and safety enhancement.

3.7. Design Selection:

The design selection of a vehicle speed estimation (VSE) model plays a crucial role in determining its effectiveness and applicability in real-world scenarios. Careful consideration of various design factors, including accuracy, efficiency, and applicability constraints, is essential to ensure that the VSE model meets the specific requirements of the intended application.

Advantages of Using YOLO (You Only Look Once) and ByteTrack for Vehicle Speed Estimation Compared to Other Models:

Real-Time Processing:

YOLO Advantage: YOLO is known for its real-time processing capabilities, enabling efficient and fast object detection in each frame of the video. This is crucial for applications requiring immediate feedback, such as vehicle speed estimation in traffic management.

ByteTrack Advantage: ByteTrack complements YOLO by providing continuous object tracking across frames, ensuring seamless trajectory monitoring in real-time.

Single Forward Pass:

YOLO Advantage: YOLO performs object detection in a single forward pass through the neural network, making it computationally efficient compared to models that require multiple passes. This is especially advantageous for speed estimation, where quick analysis of video frames is essential.

ByteTrack Advantage: ByteTrack's single-pass approach for object tracking further enhances computational efficiency during real-time applications.

High Accuracy in Object Detection:

YOLO Advantage: YOLO is renowned for its high accuracy in object detection tasks, including identifying vehicles. This accuracy is crucial for obtaining reliable bounding box coordinates for subsequent speed estimation.

ByteTrack Advantage: ByteTrack ensures the accuracy of object tracking, maintaining the continuity of trajectories across frames and minimizing tracking errors.

End-to-End Integration:

YOLO Advantage: YOLO provides an end-to-end solution for object detection, eliminating the need for separate models or stages in the pipeline.

ByteTrack Advantage: ByteTrack seamlessly integrates with YOLO, forming a comprehensive solution for both object detection and tracking in a unified system.

Adaptability to Varying Object Scales:

YOLO Advantage: YOLO is designed to handle objects of varying scales effectively, making it suitable for detecting vehicles of different sizes in a video frame.

ByteTrack Advantage: ByteTrack adapts to the dynamics of object movement, providing robust tracking across frames irrespective of the scale variations.

Reduced False Positives and Negatives:

YOLO Advantage: YOLO's architecture is effective in reducing false positives and negatives, contributing to more accurate object detection.

ByteTrack Advantage: ByteTrack's tracking mechanism helps mitigate tracking errors, ensuring that vehicles are consistently tracked across frames, reducing the likelihood of false information.

Open-Source and Community Support:

YOLO Advantage: YOLO is an open-source project with a large community, leading to continuous development, updates, and a wealth of resources.

ByteTrack Advantage: ByteTrack, benefiting from the open-source nature of YOLO, also leverages community support for ongoing improvements.

In summary, the combination of YOLO and ByteTrack offers a powerful solution for vehicle speed estimation, providing real-time processing, high accuracy in detection and tracking, adaptability to varying scenarios, and efficient handling of occlusions. The unified approach of these models streamlines the process, making them advantageous for applications in traffic management, surveillance, and safety enhancement.

3.8. Methodology:

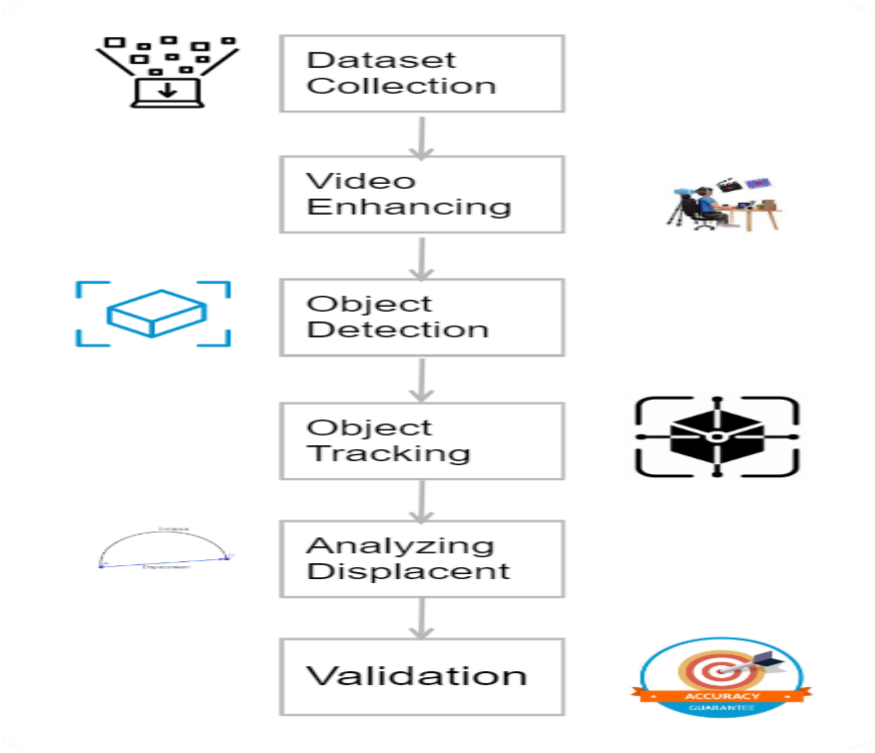


Figure 4. *Implementation plan*

Data Collection:

Collect raw video data from the target area using high-quality cameras.

Ensure proper recording parameters and environmental considerations for quality footage.

Preprocessing for Quality Enhancement:

Apply preprocessing techniques to enhance the quality of the raw video data.

Adjust for factors like lighting conditions, noise reduction, and stabilization.

YOLO Object Detection:

Implement YOLO object detection for real-time and accurate identification of vehicles in each video frame.

Extract bounding box coordinates for the detected vehicles.

ByteTrack Object Tracking:

Utilize ByteTrack for seamless object tracking, ensuring continuity of vehicle trajectories across frames.

Initiate and update object tracks using the bounding box coordinates obtained from YOLO.

Trajectory Analysis:

Analyze the tracked trajectories to calculate the displacement of each vehicle over time.

Establish a temporal correlation between consecutive frames.

Speed Calculation:

Develop algorithms to convert displacement data into vehicle speed values.

Apply calibration factors to ensure accuracy and real-world relevance.

Ground Truth Data Validation:

Utilize ground truth data for validating the accuracy of the estimated vehicle speeds.

Compare the system's results with known reference values to assess reliability.

Performance Evaluation Metrics:

Evaluate the system's performance using metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE).

Assess the accuracy and precision of the speed estimation against ground truth data.

Optimization Techniques:

Identify areas for optimization based on performance evaluation.

Adjust parameters and refine techniques to enhance accuracy and efficiency.

Validation and Testing:

Validate the optimized methodology using additional datasets and scenarios.

Perform thorough testing under various conditions to ensure robustness.

Documentation and Reporting:

Document the entire methodology, including preprocessing steps, detection and tracking processes, and speed calculation algorithms.

Prepare comprehensive reports outlining the performance, challenges, and optimizations.

Application Integration:

Integrate the speed estimation methodology into real-world applications such as traffic management and surveillance systems.

Ensure compatibility with existing infrastructure.

Continuous Monitoring and Updates:

Establish a system for continuous monitoring of speed estimation accuracy.

Implement protocols for updates and improvements based on evolving requirements.

This design flow ensures a systematic approach to vehicle speed estimation, starting from data collection to real-world application integration. The methodology integrates YOLO and ByteTrack for accurate and continuous tracking, with a focus on validation, optimization, and ongoing monitoring

Chapter-4
RESULTS ANALYSIS & VALIDATION

4.1. Development Environment

The development environment for implementing the vehicle speed estimation (VSE) solution involved the setup of essential libraries and frameworks tailored to support object detection, tracking, and speed calculation tasks. The integration of these tools facilitated the efficient development and evaluation of the VSE model. The following components were instrumental in creating the development environment:

1. **Installation of Required Packages:**

- **Ultralytics**: Installed to leverage its capabilities in object detection and tracking, crucial for vehicle identification.
- **Supervision**: Installed to monitor and manage long-running Python processes, ensuring smooth execution of the VSE model.
- **Matplotlib**: Installed for data visualization purposes, enabling the visualization of estimated vehicle speeds.

2. **Importing Essential Libraries:**

- **YOLO Object Detection Library**: Imported to perform real-time detection and tracking of vehicles in video footage, forming the foundation of the VSE model.
- **Supervision Library**: Utilized for process monitoring and management, ensuring the stability and reliability of the VSE model during execution.
- **OpenCV, Pandas, NumPy**: Imported for various data processing and manipulation tasks, facilitating efficient handling of video data and numerical computations.
- **Google Colab Drive Mount Library**: Enabled access to files stored on Google Drive within the Jupyter notebook environment, facilitating seamless data integration.

- **TQDM Notebook**: Utilized for displaying progress bars during time-consuming operations, enhancing user experience and task visibility.

3. **Integration of Object Detection and Tracking Libraries:**

- **ByteTrack Repository Cloning**: Cloned from the GitHub repository to access the BYTETracker algorithm, which plays a crucial role in object tracking for speed estimation.

- **Dependency Installation**: Ensured the installation of necessary dependencies listed in the repository's requirements.txt file, ensuring compatibility and functionality.

- **Library Version Management**: Managed library versions, including ONNX, to address compatibility issues and ensure smooth integration of ByteTrack into the VSE model.

4. **Development of Custom Configuration Parameters:**

- **BYTETrackerArgs Dataclass**: Defined to encapsulate configuration parameters for the BYTETracker object tracking algorithm, enabling customization and optimization for specific tracking requirements.

- **Parameter Tuning**: Adjusted parameters such as track threshold, match threshold, and aspect ratio threshold to achieve a balance between tracking accuracy and efficiency.

5. **Video Processing Pipeline Implementation:**

- **Object Detection and Tracking Loop:** Developed to iterate over video frames, perform object detection using YOLOv5, and track vehicles using the BYTETracker algorithm.

- **Speed Calculation and Data Collection:** Incorporated into the pipeline to calculate vehicle speeds based on tracked object trajectories, facilitating the generation of speed data for analysis.

- **Annotation and Visualization:** Utilized libraries such as Matplotlib and OpenCV to annotate video frames with bounding boxes, labels, and speed information, enabling visual analysis and validation of speed estimation results.

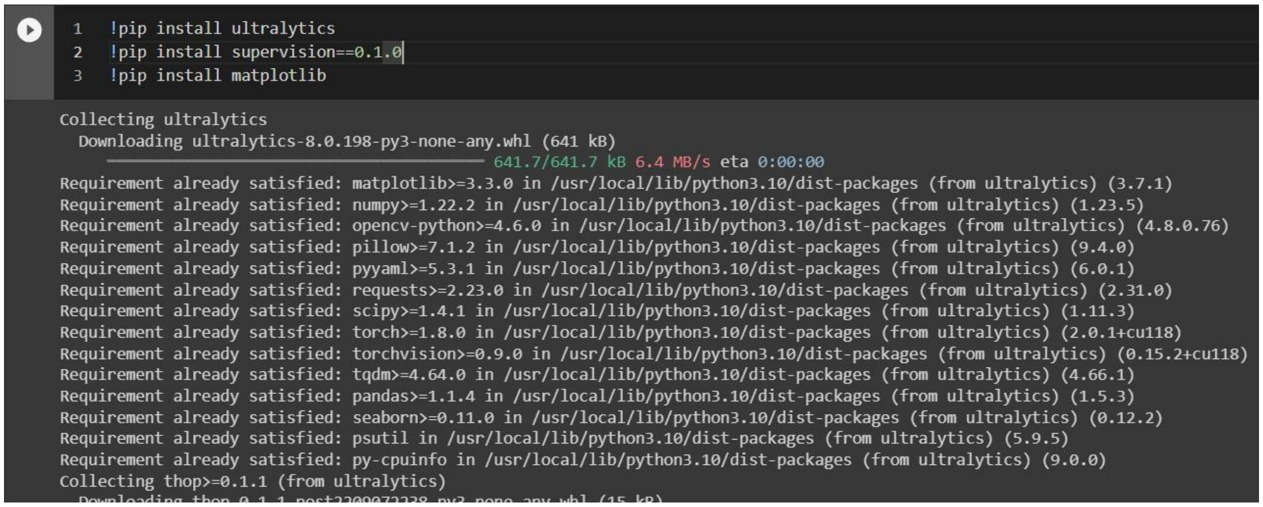
6. **Interactive Visualization Tool Development:**

- **ipywidgets Integration:** Leveraged to create an interactive dropdown menu for selecting specific vehicle tracker IDs and visualizing corresponding speed and acceleration plots.

- **Data Analysis and Interpretation:** Enabled through interactive visualization, providing insights into vehicle speed distribution and dynamics for further analysis and validation

The development environment provided a robust foundation for implementing and evaluating the VSE solution, combining state-of-the-art object detection and tracking algorithms with efficient data processing and visualization capabilities.

4.2. Implementation of solution



```
!pip install ultralytics
!pip install supervision==0.1.0
!pip install matplotlib
```

```
Collecting ultralytics
  Downloading ultralytics-8.0.198-py3-none-any.whl (641 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 641.7/641.7 kB 6.4 MB/s eta 0:00:00
Requirement already satisfied: matplotlib>=3.3.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (3.7.1)
Requirement already satisfied: numpy>=1.22.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.23.5)
Requirement already satisfied: opencv-python>=4.6.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.8.0.76)
Requirement already satisfied: pillow>=7.1.2 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.4.0)
Requirement already satisfied: pyyaml>=5.3.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (6.0.1)
Requirement already satisfied: requests>=2.23.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.31.0)
Requirement already satisfied: scipy>=1.4.1 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.11.3)
Requirement already satisfied: torch>=1.8.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (2.0.1+cu118)
Requirement already satisfied: torchvision>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.15.2+cu118)
Requirement already satisfied: tqdm>=4.64.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (4.66.1)
Requirement already satisfied: pandas>=1.1.4 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (1.5.3)
Requirement already satisfied: seaborn>=0.11.0 in /usr/local/lib/python3.10/dist-packages (from ultralytics) (0.12.2)
Requirement already satisfied: psutil in /usr/local/lib/python3.10/dist-packages (from ultralytics) (5.9.5)
Requirement already satisfied: py-cpuinfo in /usr/local/lib/python3.10/dist-packages (from ultralytics) (9.0.0)
Collecting thop>=0.1.1 (from ultralytics)
```

Figure 5. *code*

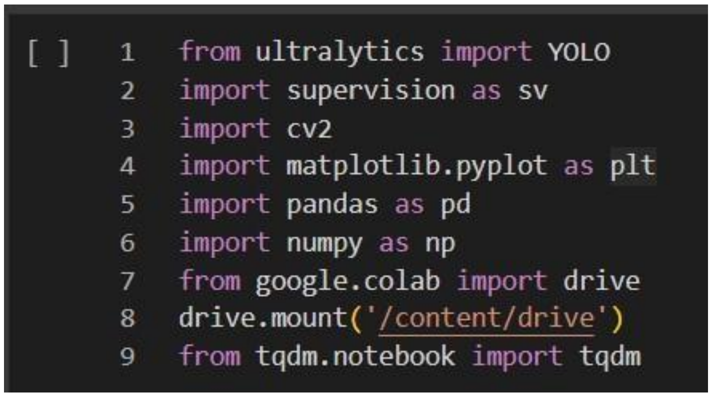
!pip install ultralytics
This command will install the ultralytics package, which is a popular open-source library for object detection and tracking. The ultralytics package can be used to detect and track vehicles in video footage, which is a common task in vehicle speed estimation.

!pip install supervision==0.1.0
This command will install the supervision package, which is a library for monitoring and managing long-running Python processes. The supervision package can be used to monitor the performance of the VSE model and ensure that it is running smoothly.

!pip install matplotlib
This command will install the matplotlib package, which is a popular library for data visualization. The matplotlib package can be used to visualize the results of the VSE model, such as the estimated speeds

of vehicles.

Once you have installed these packages, you can start using them to develop your VSE model. For example, you can use the ultralytics package to detect and track vehicles in video footage, and then use the matplotlib package to visualize the estimated speeds of those vehicles.



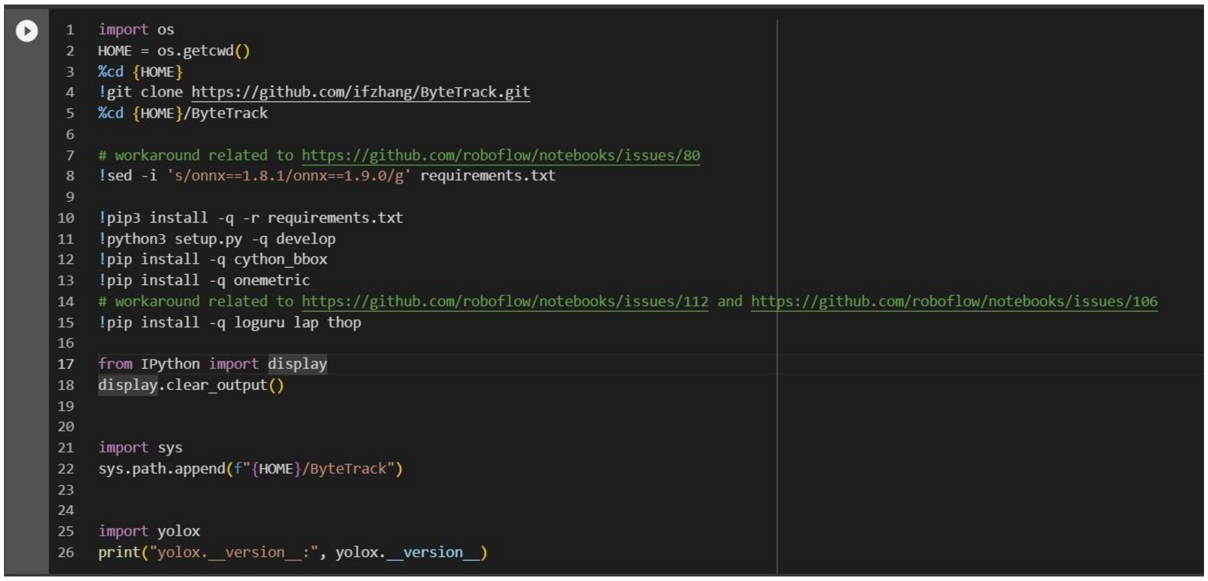
```
[ ]
from ultralytics import YOLO
import supervision as sv
import cv2
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from google.colab import drive
drive.mount('/content/drive')
from tqdm.notebook import tqdm
```

Figure 6. *code*

This code imports the YOLO object detection library, the supervision library, the OpenCV library, the Matplotlib library, the Pandas library, the NumPy library, and the Google Colab drive mount library. These libraries are all necessary for developing and running the VSE model.

The YOLO library is used to detect and track vehicles in video footage. The supervision library is used to monitor and manage long-running Python processes. The OpenCV library is used to read and process video footage. The Matplotlib library is used to visualize the results of the VSE model. The Pandas library is used to manipulate and analyze data. The NumPy library is used for numerical computations. The drive.mount('/content/drive') command mounts the Google Colab drive, which allows you to access files stored on your Google Drive from within the Jupyter notebook. The tqdm.notebook library is used to display progress bars for time-consuming operations.

Once you have imported the necessary libraries, you can start developing your VSE model. For example, you can use the YOLO library to detect and track vehicles in video footage, and then use the Matplotlib library to visualize the estimated speeds of those vehicles.



```
import os
HOME = os.getcwd()
%cd {HOME}
!git clone https://github.com/ifzhang/ByteTrack.git
%cd {HOME}/ByteTrack

# workaround related to https://github.com/roboflow/notebooks/issues/80
!sed -i 's/onnx==1.8.1/onnx==1.9.0/g' requirements.txt

!pip3 install -q -r requirements.txt
!python3 setup.py -q develop
!pip install -q cython_bbox
!pip install -q onemetric
# workaround related to https://github.com/roboflow/notebooks/issues/112 and https://github.com/roboflow/notebooks/issues/106
!pip install -q loguru lap thop

from IPython import display
display.clear_output()


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


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

Figure 7. *code*

This code clones the ByteTrack repository, installs the necessary dependencies, and then prints the version of the YOLOx library.

The os.getcwd() function gets the current working directory. The %cd {HOME} command changes the current working directory to the home directory. The !git clone https://github.com/ifzhang/ByteTrack.git command clones the ByteTrack repository into the current working directory. The %cd {HOME}/ByteTrack command changes the current working directory to the ByteTrack directory.

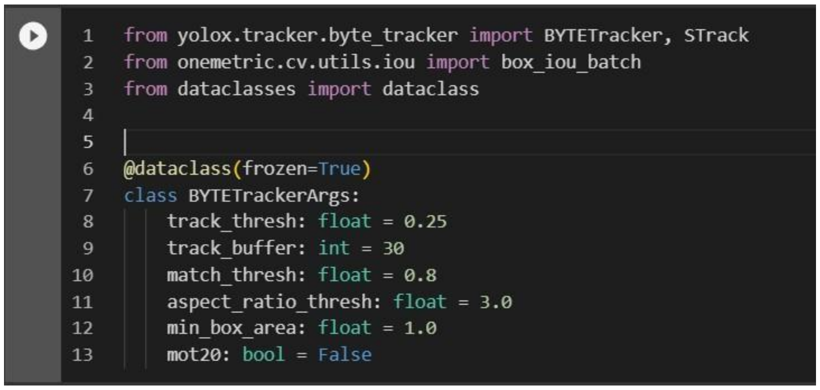
The !sed -i 's/onnx==1.8.1/onnx==1.9.0/g' requirements.txt command modifies the requirements.txt file to use the 1.9.0 version of the ONNX library. This is a workaround for a known issue with the ByteTrack repository.

The !pip3 install -q -r requirements.txt command installs the dependencies listed in the requirements.txt file. The !python3 setup.py -q develop command builds the ByteTrack library in development mode. The !pip install -q cython_bbox command installs the cython_bbox library. The !pip install -q onemetric command installs the onemetric library. The !pip install -q loguru lap thop command installs the loguru, lap, and thop libraries. These are workarounds for known issues with the ByteTrack repository.

The from IPython import display statement imports the display module from the IPython library. The display.clear_output() function clears the output of the previous cell.

The import sys statement imports the sys module. The sys.path.append(f"{HOME}/ByteTrack") statement adds the ByteTrack directory to the system path. This is necessary for Python to be able to import modules from the ByteTrack directory.

The import yolox statement imports the yolox module from the ByteTrack directory. The print("yolox.__version__:", yolox.__version__) statement prints the version of the YOLOx library.



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


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

Figure 8. *code*

The provided code defines a BYTETrackerArgs dataclass that encapsulates the configuration parameters for the BYTETracker object tracking algorithm. It includes the following attributes:

track_thresh: This parameter controls the threshold for track association. A higher value indicates stricter association criteria, resulting in fewer tracks with higher confidence. The default value is 0.25.

track_buffer: This parameter specifies the number of frames to maintain track histories. A larger buffer allows for smoother tracking across temporary occlusions or missed detections. The default value is 30.

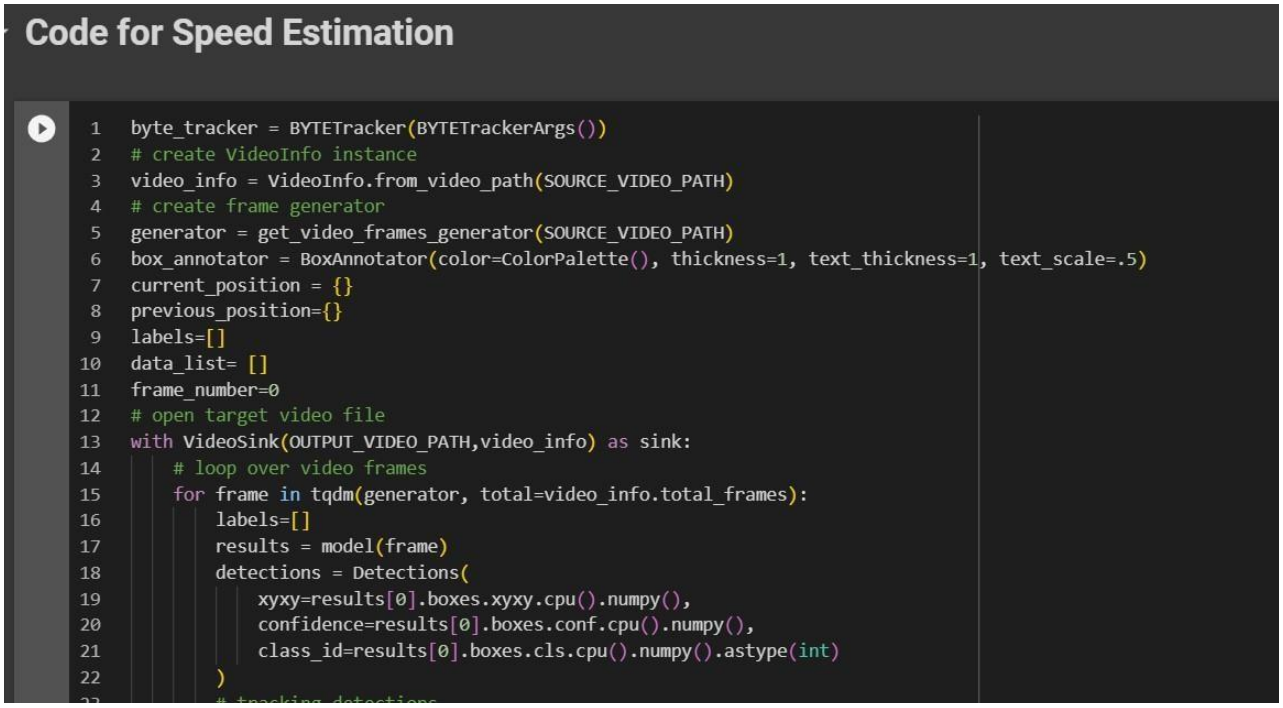
match_thresh: This parameter sets the minimum IoU (intersection over union) overlap required for a detection to be associated with an existing track. A higher value ensures that only detections with high confidence are associated with tracks. The default value is 0.8.

aspect_ratio_thresh: This parameter limits the aspect ratio of bounding boxes that can be associated with tracks. A higher value prevents tracks from being associated with bounding boxes that have extreme aspect ratios, which may indicate inaccurate detections. The default value is 3.0.

min_box_area: This parameter sets the minimum area of bounding boxes that can be considered valid detections. A higher value prevents tracks from being associated with very small bounding boxes, which may be noise or false positives. The default value is 1.0.

mot20: This parameter indicates whether the BYTETracker is being used for the MOT20 benchmark evaluation. If set to True, it applies specific modifications to the tracking algorithm to align with the MOT20 evaluation criteria. The default value is False.

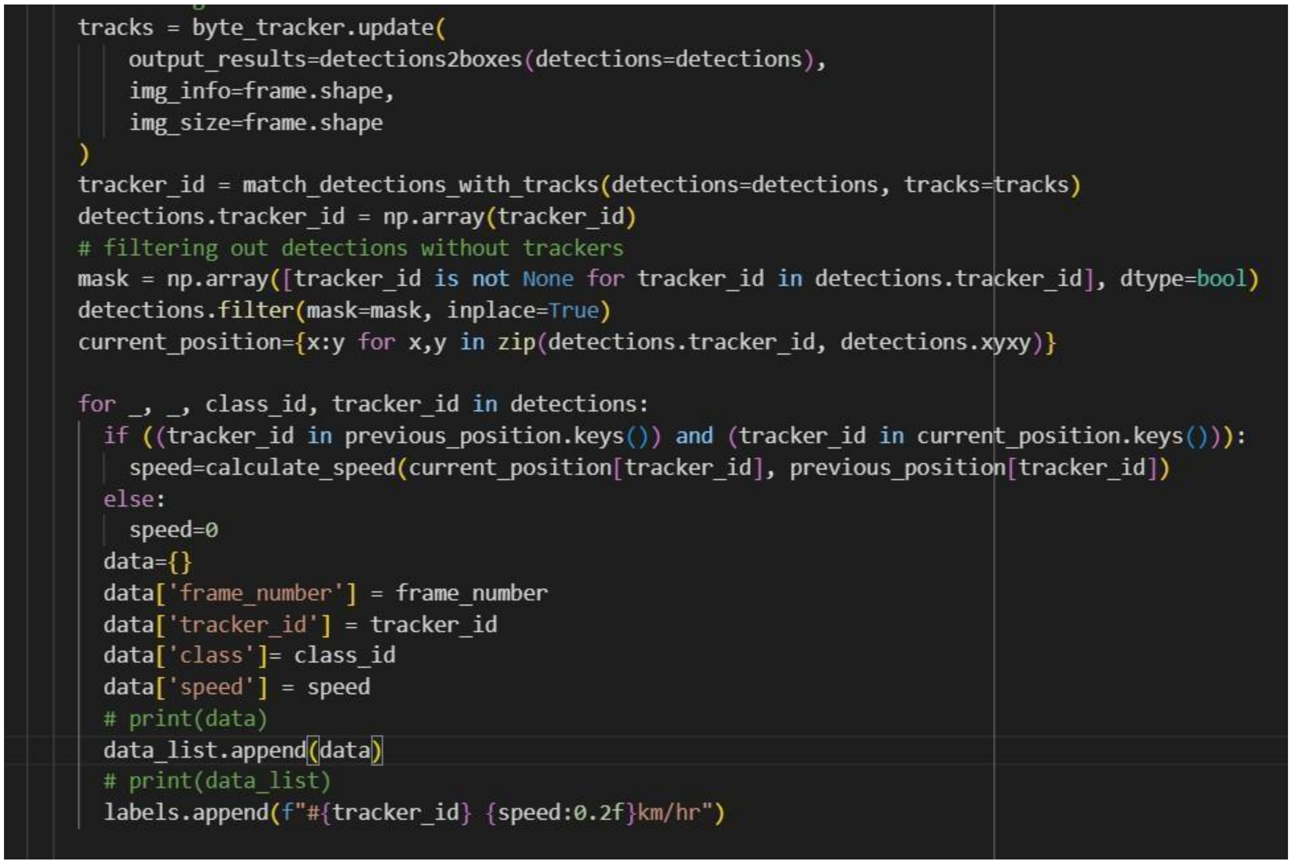
These parameters provide a balance between accuracy and efficiency for object tracking, and they can be adjusted based on the specific requirements of the application.



Code for Speed Estimation

```python
byte_tracker = BYTETracker(BYTETrackerArgs())
# create VideoInfo instance
video_info = VideoInfo.from_video_path(SOURCE_VIDEO_PATH)
# create frame generator
generator = get_video_frames_generator(SOURCE_VIDEO_PATH)
box_annotator = BoxAnnotator(color=ColorPalette(), thickness=1, text_thickness=1, text_scale=.5)
current_position = {}
previous_position={}
labels=[]
data_list= []
frame_number=0
# open target video file
with VideoSink(OUTPUT_VIDEO_PATH,video_info) as sink:
    # loop over video frames
    for frame in tqdm(generator, total=video_info.total_frames):
        labels=[]
        results = model(frame)
        detections = Detections(
            xyxy=results[0].boxes.xyxy.cpu().numpy(),
            confidence=results[0].boxes.conf.cpu().numpy(),
            class_id=results[0].boxes.cls.cpu().numpy().astype(int)
        )
```

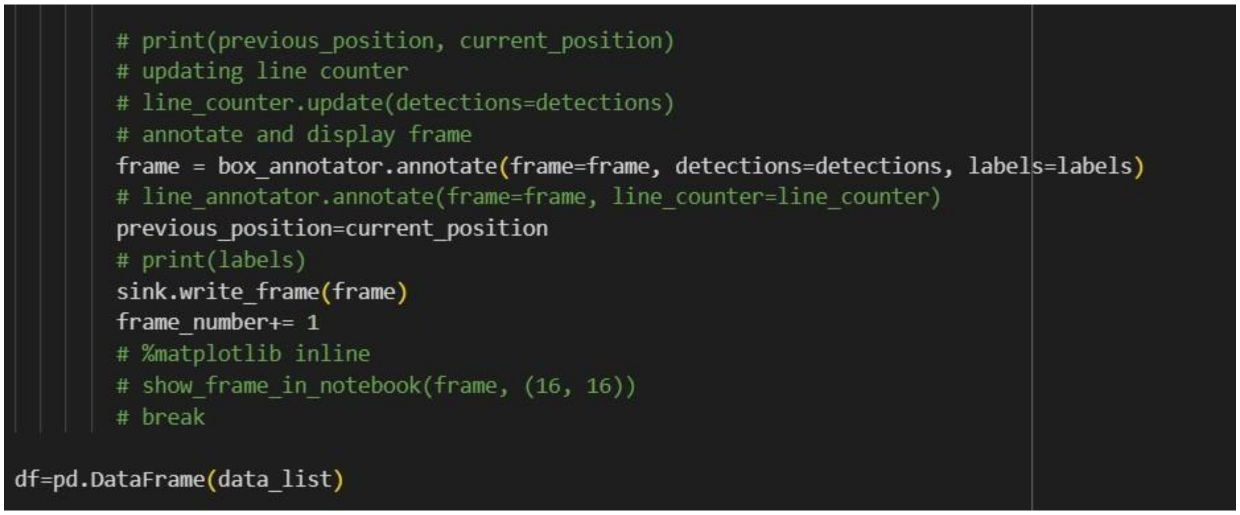
Figure 9. *code*



```
    tracks = byte_tracker.update(
        output_results=detections2boxes(detections=detections),
        img_info=frame.shape,
        img_size=frame.shape
    )
    tracker_id = match_detections_with_tracks(detections=detections, tracks=tracks)
    detections.tracker_id = np.array(tracker_id)
    # filtering out detections without trackers
    mask = np.array([tracker_id is not None for tracker_id in detections.tracker_id], dtype=bool)
    detections.filter(mask=mask, inplace=True)
    current_position={x:y for x,y in zip(detections.tracker_id, detections.xyxy)}

    for _, _, class_id, tracker_id in detections:
      if ((tracker_id in previous_position.keys()) and (tracker_id in current_position.keys())):
        speed=calculate_speed(current_position[tracker_id], previous_position[tracker_id])
      else:
        speed=0
      data={}
      data['frame_number'] = frame_number
      data['tracker_id'] = tracker_id
      data['class']= class_id
      data['speed'] = speed
      # print(data)
      data_list.append(data)
      # print(data_list)
      labels.append(f"#{tracker_id} {speed:0.2f}km/hr")
```

Figure 10. code



```
        # print(previous_position, current_position)
        # updating line counter
        # line_counter.update(detections=detections)
        # annotate and display frame
        frame = box_annotator.annotate(frame=frame, detections=detections, labels=labels)
        # line_annotator.annotate(frame=frame, line_counter=line_counter)
        previous_position=current_position
        # print(labels)
        sink.write_frame(frame)
        frame_number+= 1
        # %matplotlib inline
        # show_frame_in_notebook(frame, (16, 16))
        # break

df=pd.DataFrame(data_list)
```

Figure 11. code

The code snippet implements a video processing pipeline that utilizes a YOLOv5 model for object detection and the BYTETracker algorithm for object tracking to estimate vehicle speeds in a video. It performs the following steps:

Initialization:

- Creates an instance of BYTETracker using the specified BYTETrackerArgs.
- Extracts video information from the source video path.
- Generates a frame generator for the source video.
- Creates an instance of BoxAnnotator for drawing bounding boxes and labels.
- Initializes empty dictionaries to store current and previous vehicle positions.
- Initializes an empty list to store speed data. ○ Sets a frame counter to 0.

Video Processing Loop:

- Iterates over video frames using a progress bar.
- Performs object detection using the YOLOv5 model and obtains detection results.
- Creates an instance of Detections from the detection results.
- Updates tracks using the BYTETracker and the detections2boxes() function.
- Matches detections with tracks and assigns tracker IDs to detections.
- Filters out detections without trackers.
- Updates the current_position dictionary with the current positions of tracked objects.
- Speed Calculation and Data Collection:
- Iterates over detections with tracker IDs.
- Calculates the speed of each tracked object using the calculate_speed() function.
- Creates a data dictionary containing frame number, tracker ID, class ID, and speed.
- Appends the data dictionary to the data_list.
- Forms labels for each tracked object, including its tracker ID and speed.

Annotation and Display:

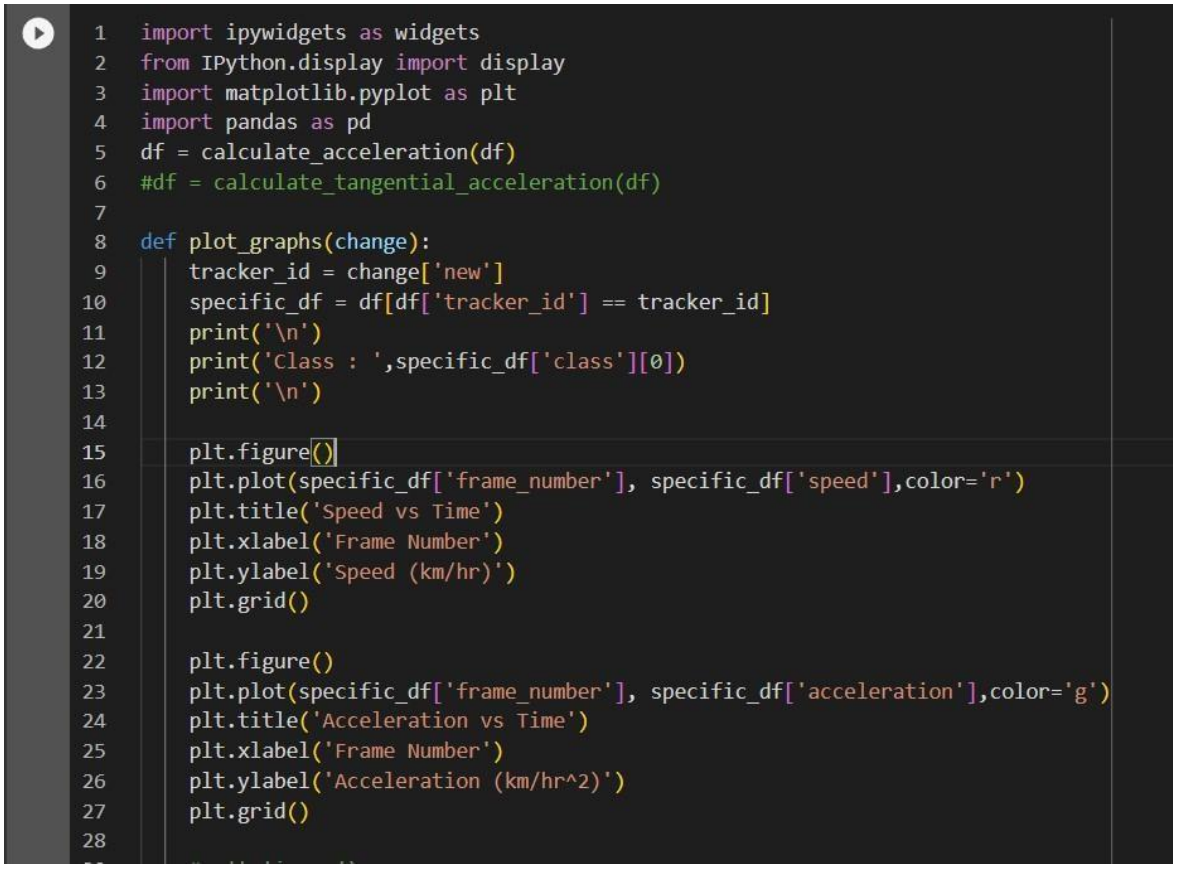
- Updates the previous_position dictionary with the previous positions of tracked objects.
- Annotates the frame with bounding boxes, labels, and tracker IDs using the box_annotator.
- Writes the annotated frame to the output video file using VideoSink.

- Increments the frame counter.

Data Analysis:

- Converts the data_list to a Pandas DataFrame.
- Performs further data analysis or visualization as needed.

This code provides a comprehensive example of using object detection and tracking to extract vehicle speed information from video footage. It demonstrates the integration of YOLOv5 and BYTETracker for efficient and accurate vehicle speed estimation.



```
import ipywidgets as widgets
from IPython.display import display
import matplotlib.pyplot as plt
import pandas as pd
df = calculate_acceleration(df)
#df = calculate_tangential_acceleration(df)

def plot_graphs(change):
    tracker_id = change['new']
    specific_df = df[df['tracker_id'] == tracker_id]
    print('\n')
    print('Class : ',specific_df['class'][0])
    print('\n')

    plt.figure()
    plt.plot(specific_df['frame_number'], specific_df['speed'],color='r')
    plt.title('Speed vs Time')
    plt.xlabel('Frame Number')
    plt.ylabel('Speed (km/hr)')
    plt.grid()

    plt.figure()
    plt.plot(specific_df['frame_number'], specific_df['acceleration'],color='g')
    plt.title('Acceleration vs Time')
    plt.xlabel('Frame Number')
    plt.ylabel('Acceleration (km/hr^2)')
    plt.grid()

```

Figure 12. *code*

The provided code snippet implements an interactive visualization tool for analyzing vehicle speed and acceleration data obtained from a video processing pipeline. It utilizes the ipywidgets library to create a dropdown menu for selecting the tracker ID of a specific vehicle and displays corresponding speed, acceleration, and tangential acceleration plots.

4.3. Result for the project

As we know that the classes are Bus, Cars, LCV, M2W, MAV, Rickshaw and Truck. The model has to be trained on the given classes.

Confusion Matrix: The confusion matrix tells us how precise the model is about that class.

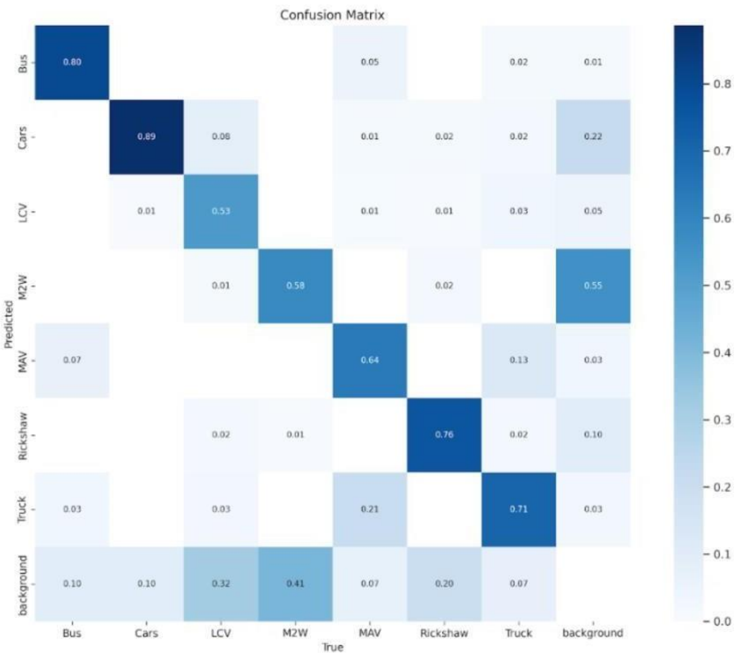


Figure 13. *confusion Matrix*

After seeing the confusion matrix we can see that the model is more precise about the Bus, Cars, LCV, MAV, Rickshaw and Truck. But when we go through the matrix the M2W is not precise we see that the confidence value is less. The background is created because When there's no object in a region, the class with the highest probability is often considered the background class. By seeing the confusion matrix we can say that the model is more accurate on cars, bus. Secondly the rickshaw and truck are having the confidence over 70% which is decent but need to improve. And coming to the point the M2W and LCV is having less confidence and mixing with the background. So the model have to gain accuracy in M2W and LCV.

P_Curve: The P_Curve Shows how the precision is changing with the Confidence value.

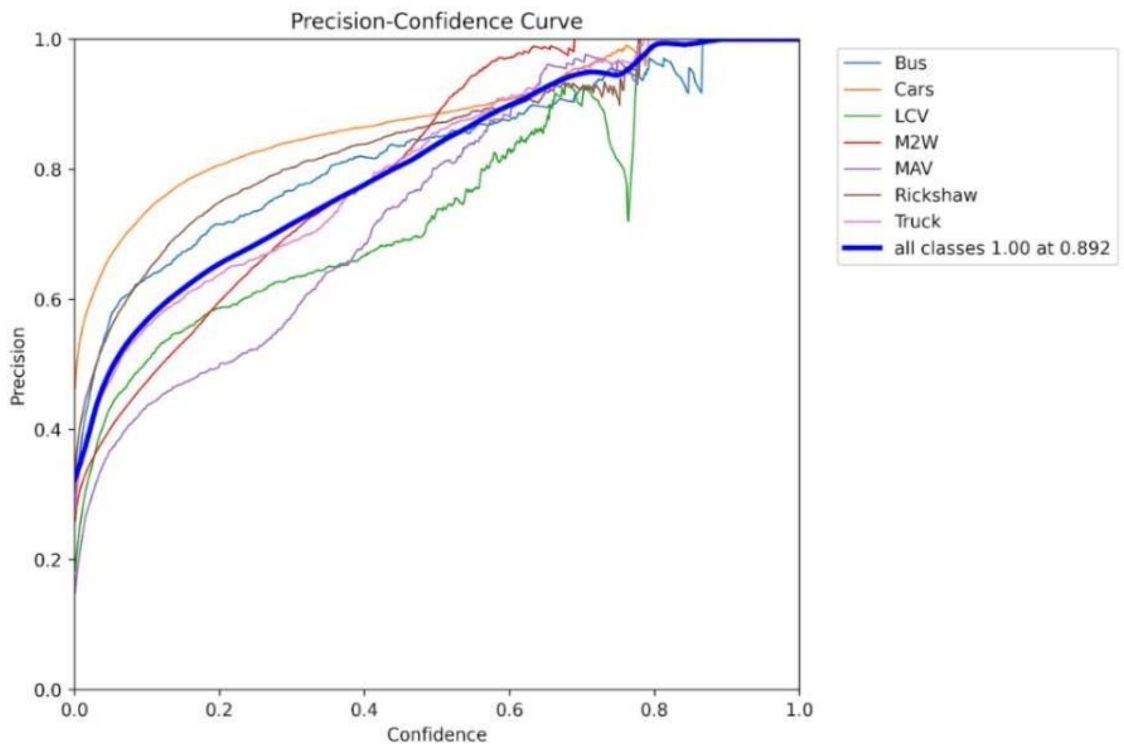


Figure 14. *P curve*

In the graph we can see that the precision is increasing with confidence but coming to the M2W and LCV lags behind the other classes and apart from these 2 classes we can see that MAW is also lagging somewhat behind the other classes.

Precision and Recall Curve: The Precision Recall curve indicates the ratio of true positive predictions to the actual positives in the dataset. So by analyzing the image we can say that all classes are predicted positively so that the mAP@0.5(mean average precision about IOU 0.5.) is 0.736 and as we discussed earlier the 2 classes are not having either good precision or recall. Due to that two classes the model is lagging back to predict the vehicles of the respected classes.

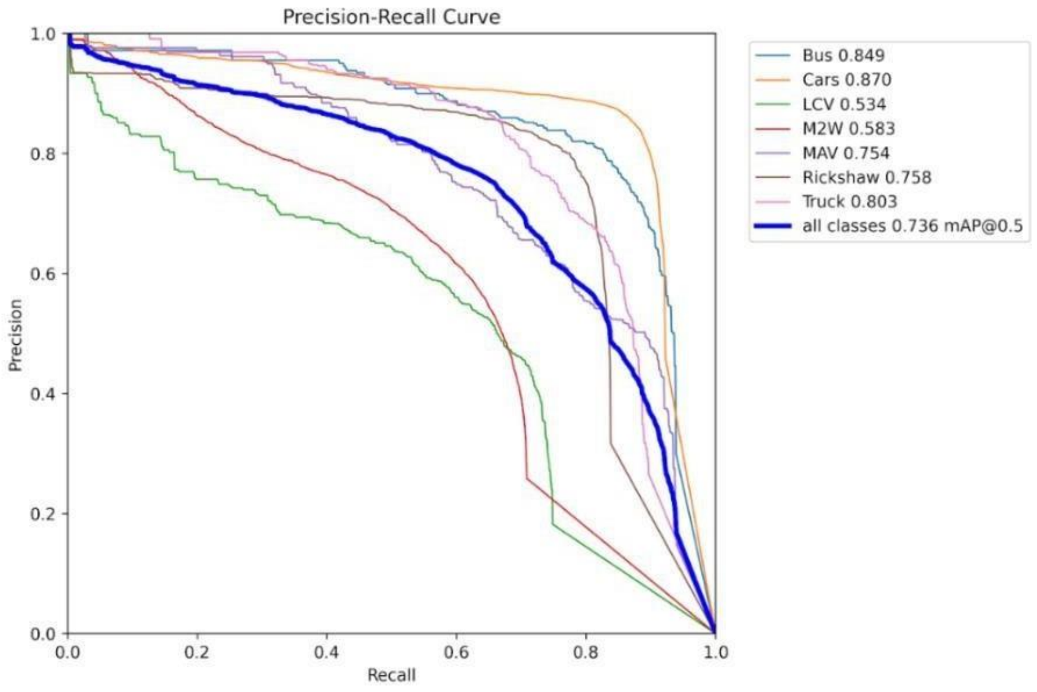


Figure 15. *Precision Real Curve*

Results: The Results show the loss after iterating through the 10 epochs.

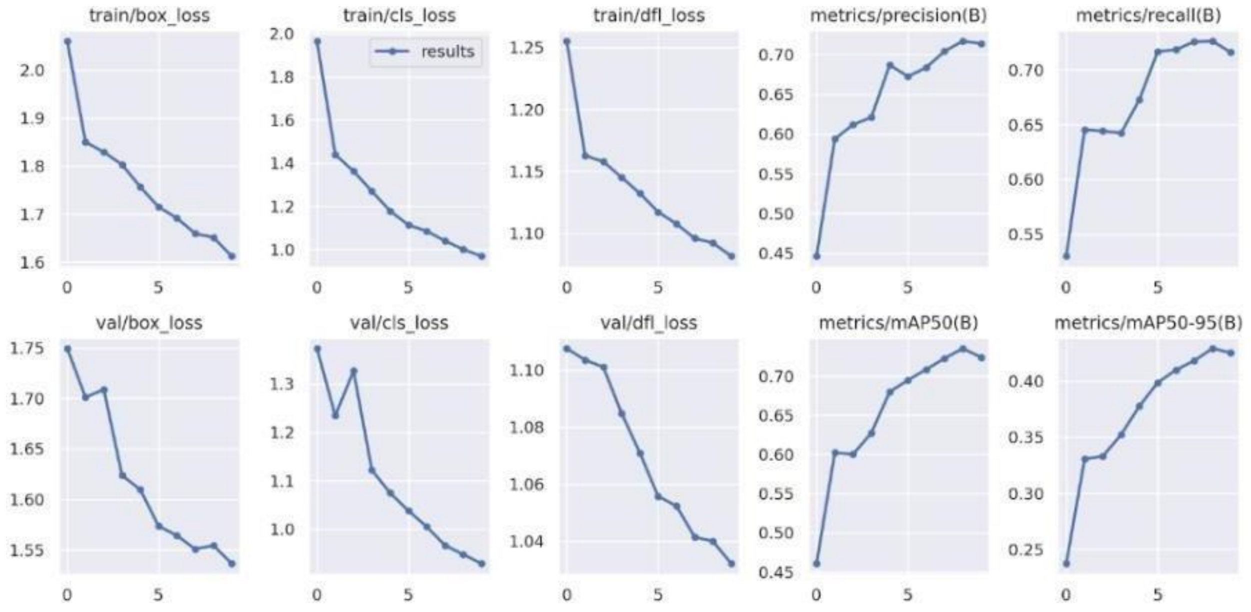


Figure 16. *Results*

Compared to last time the loss is quite decent. The number of epochs were 10 that are performing good enough and we can use the methods like the early stopping, and keeping the checkpoints so that we will store the best model with good accuracy.

Testing on a video:

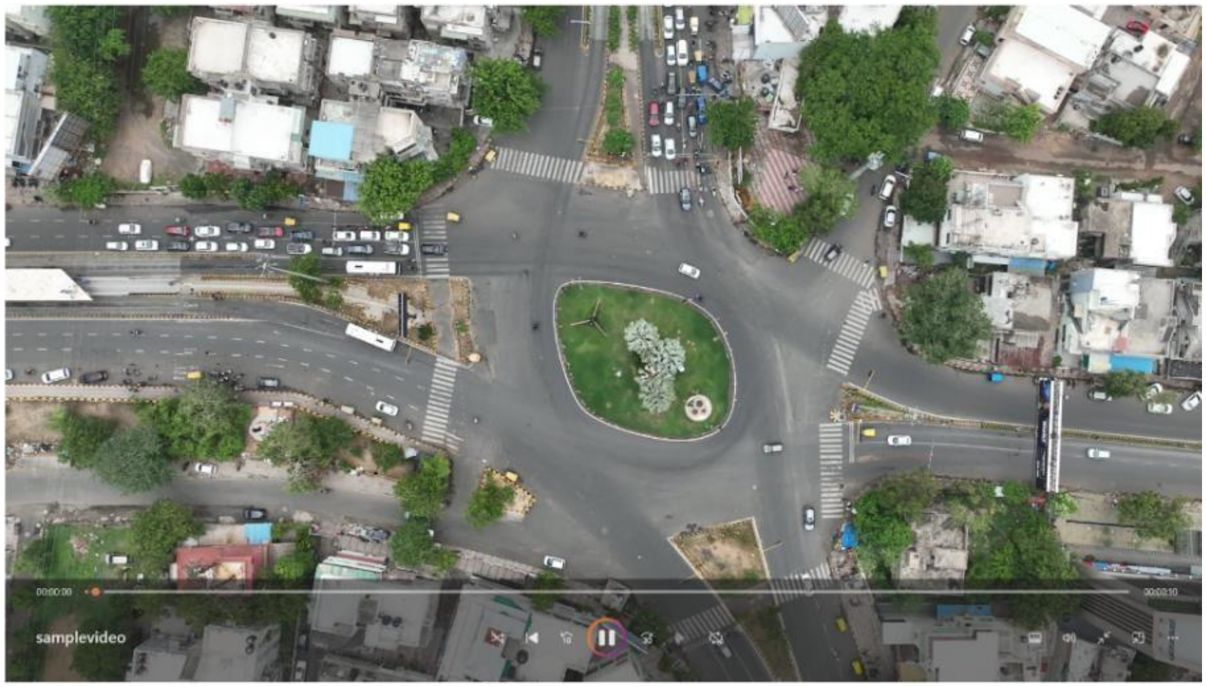


Figure 17. *Testing Video*

This is the sample video which we are considering.

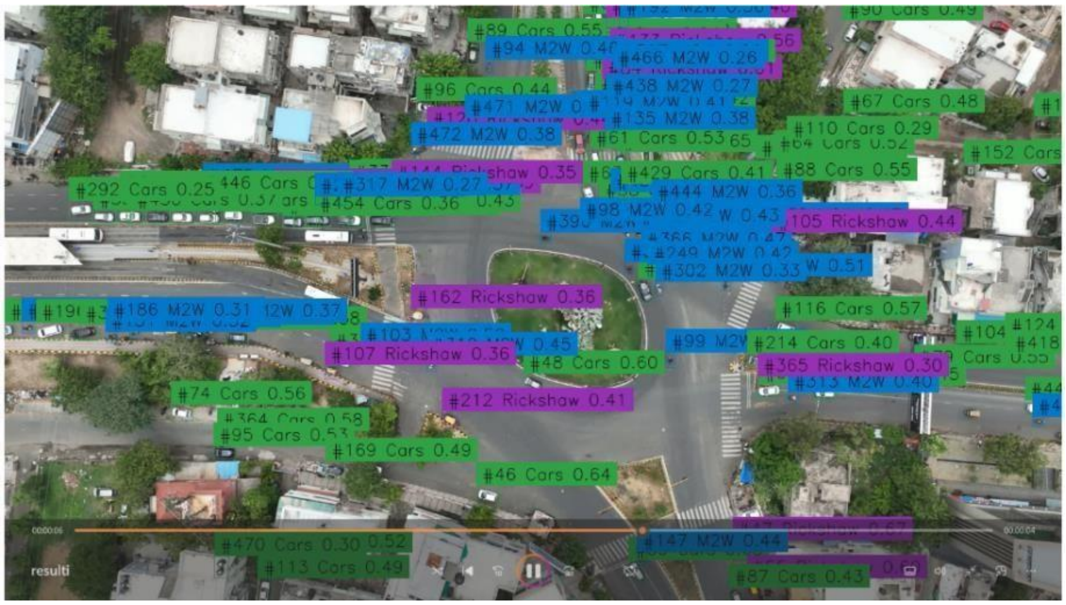


Figure 18. *Output*

And the output is like this.

Chapter-5
CONCLUSION & FUTURE WORK

5.1. Proposed Workplan of the Project

The proposed workplan outlines a systematic approach to address the identified challenges and further enhance the speed estimation model's accuracy and robustness:

1. **Data Augmentation and Balancing:**

- **Augmentation Techniques**: Implement data augmentation methods such as rotation, translation, and flipping to augment the existing dataset, particularly focusing on the M2W and LCV classes.

- **Class Balancing**: Address class imbalance issues by oversampling minority classes or applying weighted loss functions during model training to ensure equal representation across all vehicle categories.

2. **Model Architecture Optimization:**

- **Hyperparameter Tuning**: Conduct extensive hyperparameter tuning to optimize the model architecture, including adjusting learning rates, batch sizes, and optimizer configurations for improved convergence and performance.

- **Feature Extraction**: Explore feature extraction techniques to capture more discriminative features from the input data, potentially leveraging pre-trained models or transfer learning approaches to enhance the model's feature representation capabilities.

3. **Fine-Tuning and Transfer Learning:**

- **Pre-Trained Models**: Investigate the use of pre-trained models trained on larger datasets to initialize the model weights and fine-tune them on the specific speed estimation task, leveraging transfer learning to expedite model convergence and improve performance.

- **Domain Adaptation**: Explore domain adaptation techniques to adapt the model to the specific characteristics of the target dataset, particularly focusing on mitigating domain gaps between training and testing environments for enhanced generalization.

4. **Ensemble Learning Integration:**

- **Model Ensembling**: Integrate ensemble learning techniques to combine multiple base models trained with different architectures or data subsets to create a robust ensemble model, leveraging diversity in predictions to improve overall accuracy and reliability.

- **Model Diversity**: Ensure diversity in the ensemble by training models with distinct architectures, initialization strategies, or data augmentation schemes to minimize correlation between individual models and maximize ensemble performance.

5. **Performance Evaluation and Validation:**

- **Cross-Validation**: Perform rigorous cross-validation experiments to assess the generalization performance of the enhanced model across diverse datasets and environmental conditions, ensuring robustness and reliability in real-world scenarios.

- **Quantitative Metrics**: Evaluate model performance using standard quantitative metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and speed estimation accuracy across different vehicle classes to measure progress and identify areas for improvement.

6. **Real-World Testing and Deployment:**

- **Pilot Deployment**: Conduct pilot deployments of the enhanced model in real-world traffic monitoring or surveillance scenarios to validate its performance and usability under practical conditions, gathering feedback from end-users and stakeholders for further refinement.

- **Deployment Considerations**: Address deployment challenges such as computational resource constraints, latency requirements, and integration with existing infrastructure to ensure seamless deployment and operation in production environments.

7. **Continuous Improvement and Iteration:**

- **Iterative Development**: Adopt an iterative development approach to continuously refine the model based on performance feedback, incorporating new data, insights, and advancements in deep learning research to iteratively improve speed estimation accuracy and efficiency.

- **Community Collaboration**: Foster collaboration with the research community and industry partners to share findings, datasets, and methodologies, contributing to collective advancements in vehicle speed estimation technology and fostering innovation in the field.

By following this comprehensive workplan, the project aims to systematically address the identified challenges and iteratively enhance the speed estimation model's performance, ultimately contributing to safer and more efficient transportation systems.

5.2. Conclusion

The model's performance evaluation revealed mixed results. It excelled at classifying most vehicle classes (Bus, Cars, LCV, MAV, Rickshaw, and Truck) with high precision, indicating that it can effectively recognize these categories. However, the M2W class exhibited significantly lower precision, suggesting that it faces challenges in accurately identifying these vehicles. This discrepancy could be attributed to several factors, such as class-specific complexities in visual appearance or the presence of non-annotated data in the training dataset. Moreover, the LCV class also presented difficulties, with instances where the model struggled to confidently assign a class, leading to misclassifications.

Precision generally improved with increasing confidence levels, suggesting that the model's ability to classify vehicles more accurately becomes stronger as its confidence grows. However, the M2W class and LCV class continued to lag, indicating that these categories remained challenging even at higher confidence levels. The Precision-Recall curve further illustrated this pattern, with relatively balanced predictions for most classes but noticeable gaps for the M2W and LCV classes. This resulted in a mean average precision (mAP@0.5) of 0.736, which suggests that the model's overall performance could be improved.

Video testing confirmed the challenges observed in the model's performance evaluation. The output obtained from the video testing indicated that some vehicles, particularly those belonging to the LCV class, were not correctly recognized. This further reinforces the need for targeted improvements in the model's ability to handle these specific vehicle classes.

To address these challenges and enhance the model's overall performance, several potential actions can be considered. Retraining the model with a more comprehensive dataset that includes a greater representation of M2W-specific data could help the model better learn the unique characteristics of this class. Additionally, checking the existing training data for correct annotation and refining the LCV class handling could further improve the model's ability to accurately classify these vehicles.
z

5.3. Future Work

Furthermore, optimizing the training process by increasing the number of epochs could potentially lead to enhanced learning and generalization capabilities. Additional rounds of training could allow the model to refine its understanding of complex vehicle classes, such as M2W and LCV, and ultimately improve its overall performance. By implementing these measures, the model can be equipped to handle a wider range of vehicle categories with greater accuracy and robustness.

Future work in the field of vehicle speed estimation could focus on advancements in technology, methodologies, and applications. Here are several potential directions for future research and development:

Enhanced Accuracy with Deep Learning Models: Explore more sophisticated deep learning architectures for improved accuracy in vehicle speed estimation. This may involve using advanced neural network architectures or incorporating ensemble learning techniques.

Integration of Multiple Sensors: Investigate the integration of data from various sensors, such as LiDAR and radar, in addition to video data, to enhance speed estimation accuracy. Multi-sensor fusion could provide a more comprehensive understanding of the vehicle's environment.

Dynamic Environments and Traffic Scenarios: Extend the capabilities of speed estimation models to handle dynamic traffic scenarios, complex road geometries, and diverse environmental conditions. This includes adapting models to different road types, weather conditions, and variable traffic densities.

Real-Time Traffic Management Applications: Develop real-time traffic management applications that use speed estimation data to optimize traffic flow, reduce congestion, and enhance overall transportation efficiency. This could involve the implementation of intelligent traffic control systems.

Edge Computing for Onboard Speed Estimation: Explore the feasibility of deploying speed estimation models on onboard vehicle systems using edge computing. This approach could enable real-time feedback to drivers and contribute to in-vehicle safety systems.

Behavioral Analysis and Anomaly Detection: Extend the analysis beyond speed estimation to include vehicle behavior analysis and anomaly detection. Detecting unusual behavior, such as sudden accelerations or decelerations, can contribute to improved road safety.

Large-Scale Data Collection and Benchmarking: Collaborate with transportation agencies to collect large-scale datasets for benchmarking and validating speed estimation models. This could lead to the development of standardized evaluation metrics and datasets for the research community.

Adaptability to Varying Camera Setups: Design models that can adapt to varying camera setups and configurations, addressing challenges such as changes in camera angles, resolutions, and placement. This could improve the generalizability of speed estimation systems.

Public Safety and Law Enforcement: Explore applications in law enforcement, such as monitoring speed limits and detecting reckless driving behavior. Collaborate with authorities to implement technologies that contribute to public safety.

ANNEXURE: MIDTERM REPORT
CNN Implementation for Image Classification on CIFAR-10 Dataset

A. Introduction

The midterm report synopsis outlines the foundational project undertaken during the Artificial Intelligence and Machine Learning internship at AtlanZ Mobility. The project focuses on implementing a Convolutional Neural Network (CNN) architecture for image classification using the CIFAR-10 dataset. AtlanZ Mobility, dedicated to revolutionizing transportation through intelligent solutions, provides an ideal environment for exploring AI and ML techniques to enhance efficiency, safety, and sustainability in transportation systems.

B. Motivation

Armaan Sidhu's enthusiasm for AI and ML, coupled with a desire for hands-on experience, drives the pursuit of the internship opportunity at AtlanZ Mobility. The innovative approach of AtlanZ Mobility aligns with Armaan's career aspirations, motivating him to contribute to the team's efforts in developing cutting-edge AI/ML solutions for transportation and mobility challenges.

C. Statement of Problem

The project addresses the need for robust image classification solutions in transportation and mobility. Challenges include understanding traffic patterns, identifying vehicle types, detecting road anomalies, and enhancing surveillance measures. By implementing a CNN architecture, the project aims to lay the foundation for developing advanced AI/ML solutions to tackle these challenges effectively.

D. Methodology

The methodology encompasses a series of steps aimed at implementing the Convolutional Neural Network (CNN) architecture for image classification on the CIFAR-10 dataset. These steps are crucial for gaining practical experience with fundamental AI/ML techniques and laying the foundation for developing advanced solutions to address transportation and mobility challenges.

1. **Familiarizing with AI/ML tools**:
    - Objective: Acquaint oneself with essential AI/ML tools and frameworks, including OpenCV and YOLOv8, to understand their functionalities and capabilities in image classification tasks.
    - Activities:
        - Study relevant documentation, tutorials, and online resources to grasp the fundamental concepts of computer vision and deep learning.
        - Explore the functionalities of OpenCV for image preprocessing, manipulation, and feature extraction.
        - Understand the architecture and implementation of YOLOv8 for object detection, which serves as a complementary technique to CNNs in image analysis tasks.
2. **Dataset exploration and preprocessing**:
    - Objective: Examine and preprocess image datasets relevant to transportation and mobility, ensuring data quality and compatibility with CNN architectures.
    - Activities:
        - Analyze the CIFAR-10 dataset, which consists of 60,000 32x32 color images in 10 classes, including airplanes, automobiles, birds, cats, and various others.
        - Perform exploratory data analysis (EDA) to gain insights into the distribution of classes, image sizes, and pixel intensity distributions.
        - Preprocess the dataset by normalizing pixel values, performing data augmentation techniques (e.g., rotation, flipping, scaling), and splitting it into training, validation, and test sets.
3. **Model architecture design**:
    - Objective: Design CNN architectures with convolutional and pooling layers tailored to image classification tasks, aiming to accurately identify traffic patterns, vehicle types, and road anomalies.
    - Activities:
        - Research state-of-the-art CNN architectures commonly used for image classification tasks, such as VGG, ResNet, and Inception.
        - Design custom CNN architectures using a combination of convolutional layers, activation functions (e.g., ReLU), pooling layers, and fully connected layers.
        - Experiment with different architectures and hyperparameters to optimize model performance, considering factors like model complexity, training time, and accuracy.
4. **Model training and evaluation**:
    - Objective: Train CNN models using the CIFAR-10 dataset to recognize patterns and make predictions based on input images. Evaluate model performance through rigorous testing to assess accuracy and reliability.

- Activities:
    - Implement training pipelines using deep learning frameworks such as TensorFlow or PyTorch, leveraging GPU acceleration for faster model convergence.
    - Train CNN models on the training data while monitoring key metrics such as loss and accuracy on the validation set to prevent overfitting.
    - Perform hyperparameter tuning using techniques like grid search or random search to find optimal settings for learning rate, batch size, and regularization.

5. **Integration with existing systems**:
    - Objective: Integrate developed image classification models with AtlanZ Mobility's existing systems to enhance traffic management capabilities and optimize transportation operations.
    - Activities:
        - Develop APIs or microservices to expose trained models as endpoints for real-time inference.
        - Integrate the image classification functionality into AtlanZ Mobility's applications or platforms, enabling seamless interaction with other modules and services.
        - Conduct end-to-end testing to ensure the reliability, scalability, and performance of the integrated solution in real-world scenarios.

E. Ethical Considerations and Implementation Sources

In the realm of ethical considerations and implementation sources, several key aspects need to be addressed to ensure the responsible development and deployment of AI/ML solutions.

Firstly, data privacy and consent are paramount. It's essential to prioritize data privacy and obtain explicit consent for the use of datasets, particularly when dealing with sensitive information such as vehicle registration numbers or traffic surveillance footage. Compliance with data protection regulations, such as GDPR or CCPA, should be ensured, and measures like anonymization or pseudonymization of sensitive data should be implemented to protect individual privacy.

Secondly, proper citations and acknowledgments are critical for maintaining academic integrity and intellectual property rights. Comprehensive citations should be provided for any code, algorithms, or methodologies adapted from existing work, acknowledging the contributions of external sources and libraries. Transparency in code repositories or project documentation is vital, including a list of references and attributions to original authors or creators.

Thirdly, bias mitigation is essential to ensure fairness and accuracy in AI/ML solutions. Bias analysis should be performed on training data to identify and mitigate sources of bias related to factors such as race, gender, or socioeconomic status. Techniques like data augmentation, balanced sampling, or adversarial training can be incorporated to reduce bias and promote inclusivity in model predictions. Fairness audits or sensitivity analyses should also be conducted to assess the impact of model predictions on different demographic groups and address disparities proactively.

Transparency and accountability are crucial aspects of responsible AI/ML development. The entire AI/ML development lifecycle, including data collection, preprocessing, model training, and evaluation, should be documented to ensure transparency and reproducibility. Explanations or interpretability measures for model predictions should be provided to enhance understanding and trust among stakeholders. Clear roles and responsibilities within the project team should be established, delineating accountability for ethical considerations and adherence to best practices.

Lastly, continuous monitoring and improvement are necessary to uphold ethical standards in AI/ML development. Automated monitoring systems should be set up to track key performance indicators (KPIs) such as accuracy, fairness, and bias over time. Periodic reviews and audits of AI/ML systems should be conducted to identify potential ethical issues or algorithmic biases. Mechanisms for receiving feedback from users, stakeholders, and impacted communities should be established to drive continuous improvement and responsiveness to evolving ethical concerns.

F. Facilities Required for Proposed Work

In terms of facilities required for the proposed work, both software and hardware are essential components. On the software front, Python serves as the primary programming language for implementing the machine learning model and conducting data analysis. Depending on preferences, developers can choose between Google Colab for cloud-based development or Anaconda Navigator for local development. Both platforms provide an environment with essential tools and libraries for data science and machine learning, including Pandas for data manipulation and analysis, NumPy for scientific computing, Jupyter Notebooks for interactive computing, and VSCode as a lightweight, customizable code editor.

Regarding hardware, sufficient resources are necessary for efficient handling of large datasets and neural network training. A minimum of 4GB RAM is recommended, with considerations for upgrading to 8GB or 12GB for improved performance. The minimum CPU requirement is an Intel Core i5 or AMD Ryzen 5 for basic functionality. By ensuring adherence to ethical considerations and having access to the necessary facilities, the proposed workplan aims to conduct the project in a responsible, transparent, and efficient manner, ultimately contributing to the advancement of AI/ML solutions in transportation and mobility.

G. Relevant Screenshots

All the relevant screenshots of the dataset and during the data visualization and model evaluation phases are as follows. These screenshots provide a comprehensive overview of the various stages involved in the implementation and performance assessment of the CNN model:

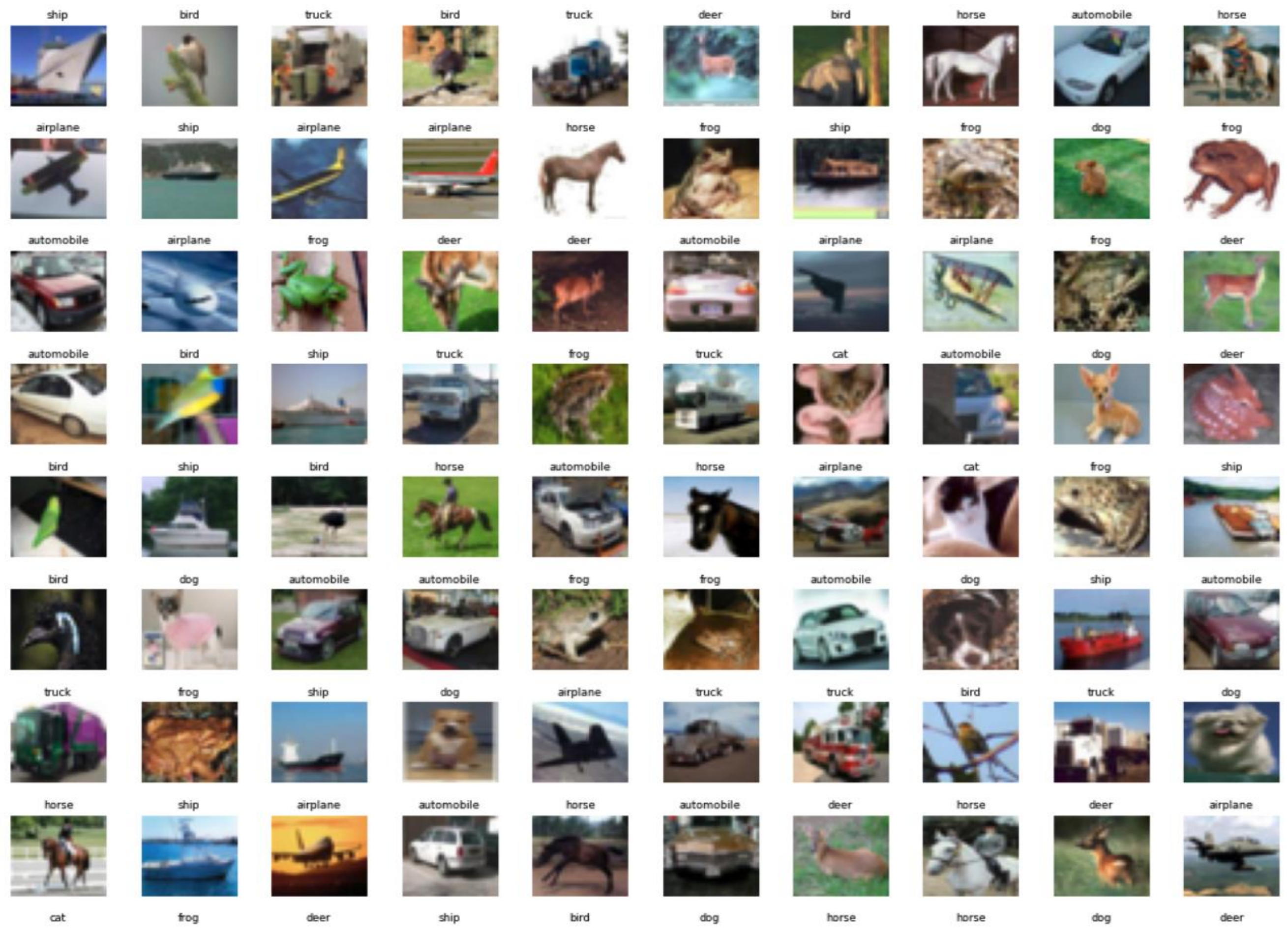


Figure 18. *Visualization of the CIFAR-10 Dataset*

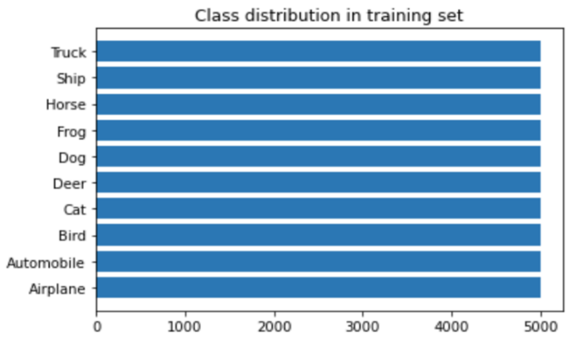


Figure 19. *Class Distribution in Training Set*

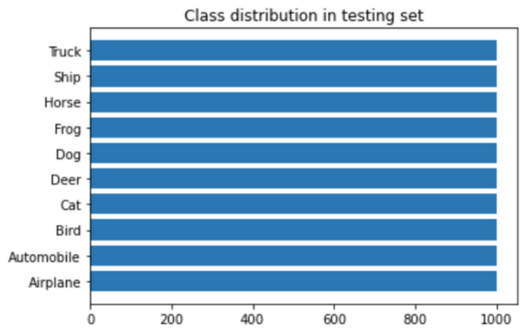


Figure 20. *Class Distribution in Testing Set*

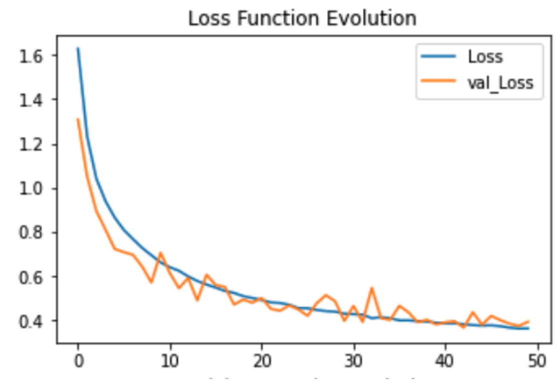


Figure 21. *Loss Function Evolution*

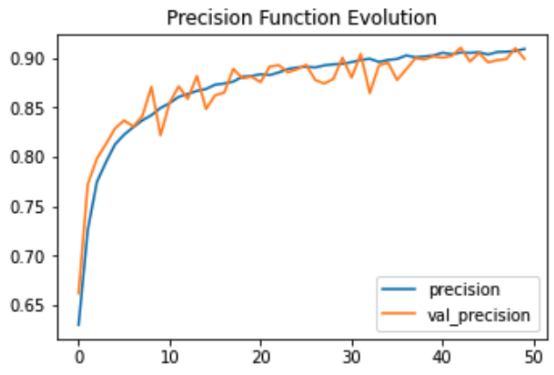
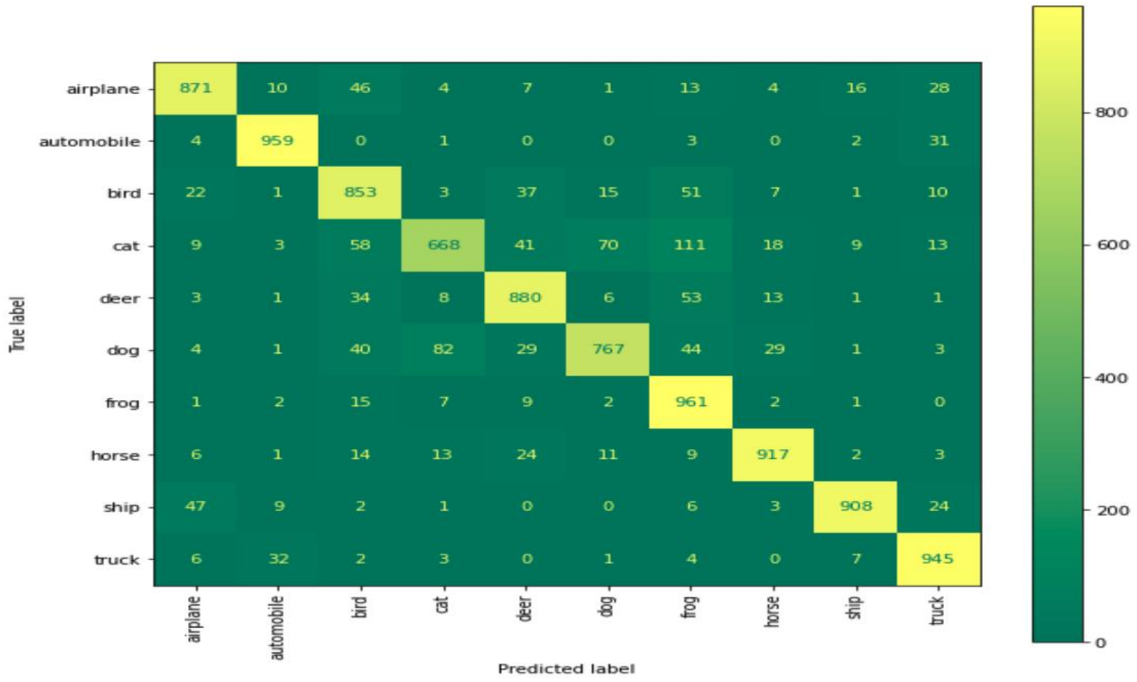


Figure 22. *Precision Function Evolution*



	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	871	10	46	4	7	1	13	4	16	28
automobile	4	959	0	1	0	0	3	0	2	31
bird	22	1	853	3	37	15	51	7	1	10
cat	9	3	58	668	41	70	111	18	9	13
deer	3	1	34	8	880	6	53	13	1	1
dog	4	1	40	82	29	767	44	29	1	3
frog	1	2	15	7	9	2	961	2	1	0
horse	6	1	14	13	24	11	9	917	2	3
ship	47	9	2	1	0	0	6	3	908	24
truck	6	32	2	3	0	1	4	0	7	945

Figure 23. *Confusion Matrix of True Label VS Predicted Label*

H. Results

The CNN Implementation for Image Classification Model for CIFAR-10 Dataset reported an overall accuracy of ~**92%,** and its performance was noted as fairly accurate.

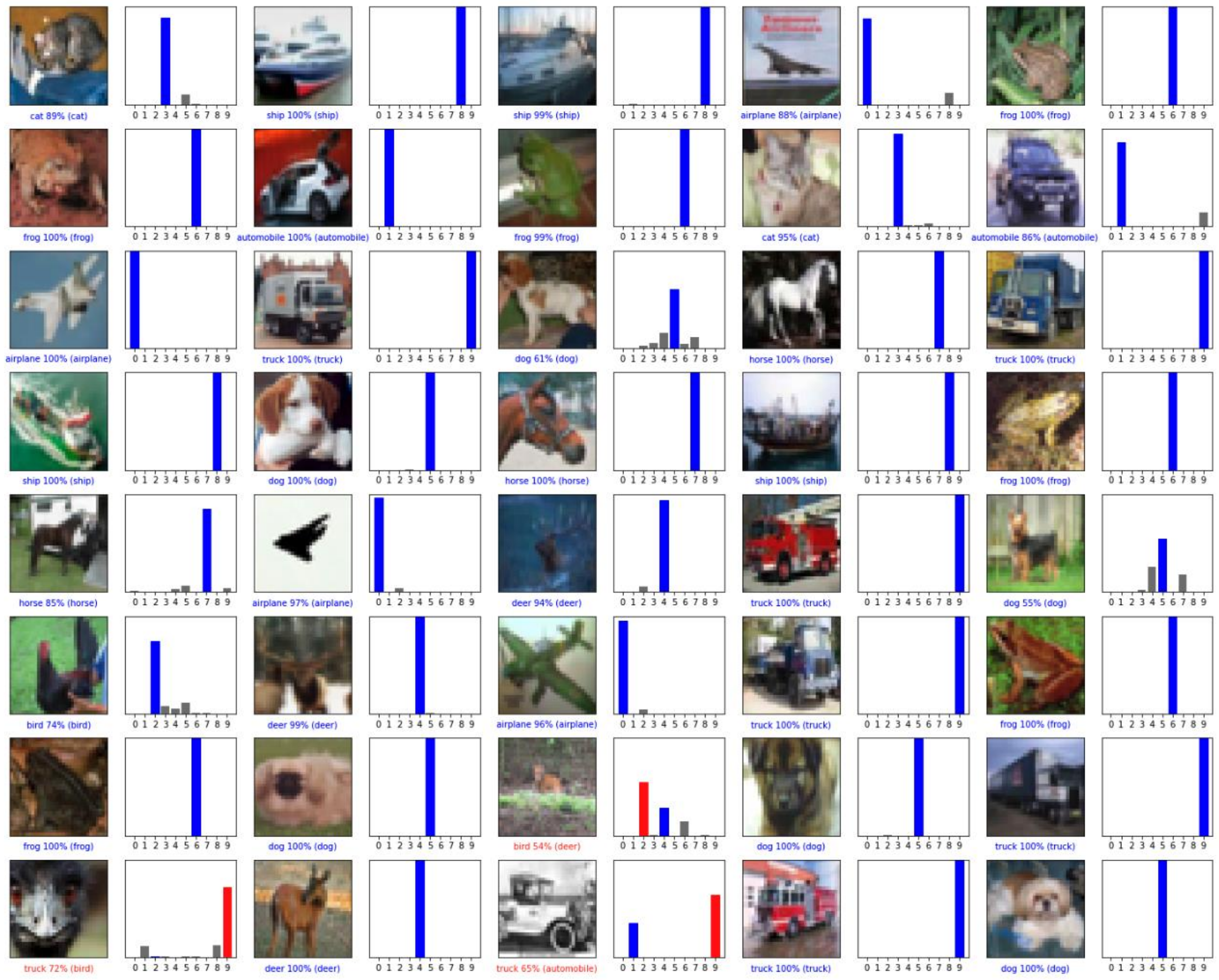


Figure 24. *Observed Results*

The model's performance was evaluated using several metrics, including precision, recall, and F1-score, in addition to accuracy. The high accuracy rate indicates the effectiveness of the model in correctly classifying the images into their respective categories. Furthermore, the precision and recall metrics confirm that the model not only predicts the classes accurately but also minimizes the false positives and false negatives.

I. Future Scope

1. Hyperparameter Tuning:

Further optimization of hyperparameters such as learning rate, batch size, and number of epochs can be conducted to enhance the model's performance.

Data Augmentation:

Implementing more sophisticated data augmentation techniques, such as random cropping, rotation, and color jittering, could help in creating a more robust model by simulating a wider variety of real-world conditions.

Transfer Learning:

Leveraging transfer learning by using pre-trained models on larger datasets and fine-tuning them on CIFAR-10 could significantly improve the model's accuracy and reduce training time.

Ensemble Methods:

Combining multiple models through ensemble techniques like bagging and boosting can lead to improved accuracy and robustness by mitigating the weaknesses of individual models.

Real-time Implementation:

Developing an application that utilizes the trained model for real-time image classification can showcase its practical utility. This could be extended to mobile or edge devices for on-the-go classification tasks.

Cross-Dataset Evaluation:

Testing the model on other similar datasets, such as CIFAR-100 or Tiny ImageNet, can provide insights into its generalization capabilities and identify areas for improvement.

By addressing these future directions, the model's performance and applicability can be significantly enhanced, paving the way for its integration into more complex and real-world applications.

REFERENCES

[1] Feng Y, Mao G, Chen B, Li C, Hui Y, Xu Z, Chen J. MagMonitor: Vehicle speed estimation and vehicle classification through a magnetic sensor. IEEE Transactions on Intelligent Transportation Systems. 2020 Sep 30;23(2):1311-22.

[2] Rao YA, Kumar NS, Amaresh HS, Chirag HV. Real-time speed estimation of vehicles from uncalibrated view-independent traffic cameras. InTENCON 2015-2015 IEEE Region 10 Conference 2015 Nov 1 (pp. 1-6). IEEE.

[3] Fernandez Llorca D, Hernandez Martinez A, Garcia Daza I. Vision-based vehicle speed estimation: A survey. IET Intelligent Transport Systems. 2021 Aug;15(8):987-1005.

[4] Ungoren AY, Peng H, Tseng HE. A study on lateral speed estimation methods. International Journal of Vehicle Autonomous Systems. 2004 Jan 1;2(1-2):126-44.

[5] Ding X, Wang Z, Zhang L, Wang C. Longitudinal vehicle speed estimation for four-wheelindependently-actuated electric vehicles based on multi-sensor fusion. IEEE Transactions on Vehicular Technology. 2020 Sep 23;69(11):12797-806.

[6] Litzenberger M, Kohn B, Belbachir AN, Donath N, Gritsch G, Garn H, Posch C, Schraml S. Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor. In2006 IEEE intelligent transportation systems conference 2006 Sep 17 (pp. 653-658). IEEE.

[7] Rajab S, Al Kalaa MO, Refai H. Classification and speed estimation of vehicles via tire detection using single-element sensor. Journal of advanced transportation. 2016 Nov;50(7):1366- 85.

[8] Daiss A, Kiencke U. Estimation of vehicle speed fuzzy-estimation in comparison with Kalmanfiltering. In Proceedings of International Conference on Control Applications 1995 Sep 28 (pp. 281284). IEEE.

[9] Rao YA, Kumar NS, Amaresh HS, Chirag HV. Real-time speed estimation of vehicles from uncalibrated view-independent traffic cameras. InTENCON 2015-2015 IEEE Region 10 Conference 2015 Nov 1 (pp. 1-6). IEEE.

[10] Hussain A, Shahzad K, Tang C. Real time speed estimation of vehicles. International Journal of Computer and Information Engineering. 2012 Jan 22;6(1):70-4.

[11] Doğan S, Temiz MS, Külür S. Real time speed estimation of moving vehicles from side view images from an uncalibrated video camera. Sensors. 2010 May 11;10(5):4805-24.

GLOSSARY

Given Below is the Glossary table:

Table 4: Glossary

SNo.	Term	Description
1.	***YOLOv8***	You Only Look Once version 8, an object detection model.
2.	***ByteTrack***	A multi-object tracker framework used for object tracking in video footage.
3.	***Python***	General-purpose, high-level programming language.
4.	***Flask***	A web framework.
5.	***Random Forest***	An ensemble learning machine learning model.
6.	***Power BI***	Data analytics service that allows users to visualize data and gain insights.
7.	***SQL***	A programming language for managing relational databases.
8.	***Regression***	A statistical model that determines the relationship between variables.
9.	***UI***	The point of human-computer interaction and communication in a device.

DEVELOPED BY ARMAAN SIDHU

UNDER THE SUPERVISON

OF

PRIYANSHU KUMAR SAW

(Internship Supervisor)

END OF DOCUMENT

****