Real-Time Vehicle Detection Model Using YOLOv8

Vaibhav Davhale, Sanket Bagal, Tushar Jadhav, Aditya Ingawale
Department of Computer Engineering,
Genba Sopanrao Moze College of Engineering, Balewadi, Pune, India

ABSTRACT: With the rapid advancement of intelligent transportation systems, real-time vehicle detection has become a critical component in traffic monitoring, autonomous driving, and road safety applications. This research explores the application of the YOLO (You Only Look Once) deep learning model for fast and accurate vehicle detection. By leveraging a convolutional neural network-based architecture, the proposed system efficiently detects and classifies vehicles in real-world traffic scenarios. The model is trained on a diverse dataset, incorporating various environmental conditions to enhance robustness. Experimental results demonstrate that the proposed YOLO-based approach achieves high detection accuracy while maintaining low computational cost, making it suitable for real-time deployment in smart traffic management systems. To further enhance detection accuracy, this study integrates advanced data augmentation techniques and optimization strategies, including transfer learning and anchor box refinement. The findings suggest that YOLO-based vehicle detection can significantly improve traffic flow analysis, automated surveillance, and accident prevention systems. Future research will focus on integrating sensor fusion techniques and improving small-object detection for more comprehensive vehicle recognition.

Keywords: Vehicle Detection, YOLOv8, Real-Time Detection, Machine Learning, Traffic Monitoring, Autonomous Driving.

1. INTRODUCTION

The rapid advancement of intelligent transportation systems (ITS) has led to increased demand for efficient and accurate vehicle detection methods. Real-time vehicle detection plays a crucial role in applications such as traffic monitoring, autonomous driving, accident prevention, and smart city management. Traditional computer vision-based techniques for vehicle detection, such as edge detection and background subtraction, often struggle with complex environmental conditions, including varying lighting, occlusions, and diverse vehicle types. To overcome these limitations, deep learning-based object detection models have gained significant attention due to their superior accuracy and robustness. Among various deep learning models, You Only Look Once (YOLO) has emerged as one of the most efficient object detection algorithms, offering real-time performance without compromising detection accuracy. Unlike region-based methods such as R-CNN and Faster R-CNN, YOLO performs object detection in a single pass through the neural network, making it highly suitable for real-time applications. The YOLO framework divides an image into grids and predicts bounding boxes and class probabilities simultaneously, enabling fast and precise vehicle detection. This research focuses on implementing and optimizing YOLO for real-time vehicle detection in complex traffic environments. By leveraging advanced deep learning techniques, the study aims to improve detection accuracy, reduce computational cost, and ensure robustness across diverse traffic conditions. The proposed model is evaluated on benchmark datasets, and its performance is analyzed in terms of accuracy, inference speed, and adaptability to real-world scenarios. Additionally, potential enhancements such as anchor box refinement, transfer learning, and sensor fusion are explored to further optimize the system. The remainder of this paper is structured as follows: Section 2 discusses related work in the field of vehicle detection, highlighting existing methodologies and their limitations. Section 3 details the proposed YOLO-based model, including dataset selection, model architecture, and training methodology. Section 4 presents the experimental results and performance evaluation. Finally, Section 5 concludes the study and outlines future research directions.

2. LITERATURE SURVEY

1. Intelligent Site Detection Based on Improved YOLO Algorithm
Authors: Shidan Sun1, Shijia Zhao and Jiachun Zheng Jimei
Description: This paper studies the improvement of algorithm based on YOLO to detect the wearing of helmet and reflective clothing at the same time. The algorithm optimizes the K-means clustering algorithm to select a more appropriate aiming frame to adapt to the data set; optimizes the internal structure of the Darknet network and some network fine-tuning, and extracts the features of small targets many times to enhance the effect of feature extraction; the experimental analysis results show that the algorithm can meet the requirements of the real scene safety helmet and reflective clothing wear detection at the same time, and can be used for reference.

2. Based on the improved YOLOv8 pedestrian and vehicle detection under low-visibility conditions
Authors: Xiuchun Wu1Mechanics Institute, Shanghai Dianji
Description: The paper proposes an improved algorithm to address the challenges of automatic driving in low visibility conditions. It introduces DCNv2 (Deformable Convolution v2) to replace the convolution in the C2F module. The DCNv2 convolution allows the network to learn spatial deformations in the convolutional layers, enabling more flexible and precise feature extraction, the improved backbone network can better capture the variations in object shapes and sizes, especially for small and fine-grained objects in complex scenes. Incorporates a self-attention mechanism called Biform.

3. Method research on ship detection in remote sensing image based on Yolo algorithm
Authors: Xi Li, Kaiyu Cai
Aiming at the difficult problem of ship identification in remote sensing image, this paper proposes the YOLO V3 based identification method. The algorithm was optimized through training sample pre-screening, Anchor box recombination class, feature pyramid network optimization, sliding segmentation of input large image window and other methods.

4. Research On Multi- Target Vehicle Detection and Tracking Based on YOLO
Authors: Tchanchou Ngatouo, Sanxi Jiang
Description: In this paper, they have constructed auto asset collection by the camara position proposed an entity recognition and shadowing algorithm to lane videos sequence captured by phones. The Lane roadside area baseline yielded yet another acceptable ROI area. On the basis of the collection of anno-tatted lane vehicle objects, the YOLOv5 object identification algorithm created a whole chain of lane engine detection brands.

5. Vehicle Overtaking Detection Using Computer Vision Techniques
Authors: Chin Sin Ong, Tee Connie, Michael Kah Ong Goh
Description: They proposed point-line distance method shows a high success rate in detecting illegal vehicle overtaking on road. When comparing the three object detection models including YOLOv7, YOLOv8n, and YOLOv8x algorithms, YOLOv8x model yields to the best object detection result thus being selected as the object detector in this paper. Deep SORT is demonstrated to successfully tracked vehicles with stability even when perspective changes occur within the videos. By combining the canny edge detection and Hough transform methods, overtaking lane marking has been successfully identified within frames, allowing automated lane detection for the specified area.

3. FLOWCHART

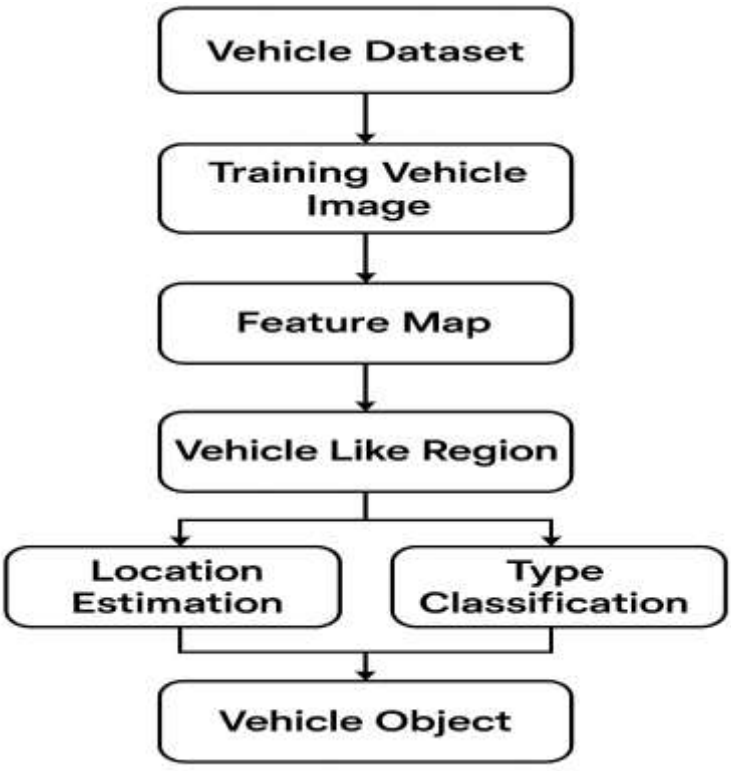

Fig 1 General flow chart of how vehicle detection works

The architecture of the Vehicle Detection System is a streamlined pipeline designed to efficiently transform raw data into actionable insights, specifically for vehicle identification and classification. The system integrates a series of stages, each with a specific purpose, ensuring an organized and effective workflow that leverages advanced deep learning techniques. This flowchart provides a comprehensive look at each stage of the system's process:

1. Vehicle Dataset The process begins with the Vehicle Dataset, which serves as the cornerstone of the Vehicle Detection System's architecture. This dataset comprises labeled images of various vehicle types (e.g., cars, bikes, trucks) captured under a range of conditions. The diversity and quality of this dataset are crucial, as they directly affect the system's ability to generalize and accurately identify vehicles in diverse real-time scenarios. Ensuring a robust dataset helps the model learn patterns essential for distinguishing between different vehicle types effectively.

2. In the Training Vehicle Image phase, the data undergoes preprocessing to optimize it for the model training process. This step includes techniques like data augmented (which involves rotating, flipping, or adjusting the brightness of images), normalization (scaling pixel values for uniformity), and resizing images to create a consistent dataset. During training the Vehicle Detection System utilizes the YOLO (You Only Look Once) algorithm, a powerful framework known for real-time object detection capabilities. YOLO processes the preprocessed data, analyzing annotated images to learn patterns and features that differentiate one vehicle type from another.

3. Feature Map Following the training phase, the system generates a Feature Map. This feature map encapsulates the essential characteristics of the images and is vital for subsequent detection tasks. Key features captured include vehicle shape, size, color, and texture, all of which enable the system to detect vehicles within an image accurately. YOLO architecture creates this feature map by applying a series of convolutional layers that reduce the spatial dimensions while enhancing the depth of feature representation. By focusing on unique visual features, the feature map becomes a foundation for further stages.

4. Vehicle-Like Region The next stage, Vehicle-Like Region, analyzes the feature map to identify areas that are likely to contain vehicles. To accomplish this, the system uses anchor boxes—predefined bounding boxes that propose areas where vehicles are probably located. By employing techniques such as non-maximum suppression (which eliminates redundant bounding boxes), the system refines its detection, retaining only the most relevant bounding boxes. This optimization step is essential, as it minimizes computational load and narrows down the regions in the images that require further analysis, boosting both speed and accuracy.

5. Location Estimation and Type Classification in the Location Estimation and Type Classification stage, the system uses the refined bounding boxes to estimate the precise locations of vehicles within each frame while classifying each detected object. YOLO excels in this dual function by predicting class probabilities and bounding box coordinates simultaneously. This means that the model not only identifies a vehicle's position but also determines its category (car, bike, truck, etc.). The combination of accurate location and reliable classification provides a well-rounded output for practical applications.

6. Vehicle Object The final stage, Vehicle Object consolidates all previous outputs to deliver a robust detection result. Here, the system identifies the presence, location, and type of each vehicle, enabling real-time insights into vehicular landscapes. The result allows for applications such as traffic analysis, vehicle counting, and monitoring, which can be utilized in traffic management and safety solutions. By optimizing for high performance, this stage ensures that the system remains efficient even in environments with high vehicle volumes. The Vehicle Detection System's architecture demonstrates a well-defined, systematic framework that seamlessly integrates each of these stages. By advancing through data collection, preprocessing, feature extraction, region selection, and classification, this architecture transforms raw video data into meaningful outputs. The structured pipeline illustrates the power of machine learning and computer vision to deliver effective solutions in vehicular monitoring and traffic management, catering to growing needs for efficient and real-time vehicular insights.

4. METHODLOGY

The methodology for developing the Vehicle Detection System involves a systematic approach that integrates several stages to ensure efficient and accurate detection and classification of vehicles from video feeds. The first step is data collection, achieved by leveraging video sources from traffic cameras or user-uploaded footage. This data collection ensures a diverse dataset that includes various vehicle types and traffic conditions, 1. 1. providing a solid foundation for training the detection model. After data collection, the next phase is data preprocessing, which cleans and prepares the video frames for analysis. This involves techniques such as frame extraction, where individual frames are taken from the

video stream for processing. Key steps in this phase include resizing the images to a consistent dimension suitable for the YOLO model, normalizing pixel values, and augmenting the data through transformations like rotation and flipping to enhance the model's robustness. Additionally, any unnecessary noise or irrelevant elements in the frames are filtered out to focus solely on the vehicles. The core of the methodology is the vehicle detection process, which employs the YOLO (You Only Look Once) algorithm for real-time detection and classification. YOLO utilizes a single neural network to predict bounding boxes and class probabilities directly from full images, allowing for fast and accurate identification of multiple vehicle types, such as cars, bikes, and trucks, within the same frame. The model is trained on annotated datasets that include labelled images of various vehicles, enabling it to learn distinctive features and characteristics for effective classification. The system is implemented using Python, integrating various libraries such as OpenCV for video processing and TensorFlow or PyTorch for model training and inference. The tool features a user-friendly interface that allows users to input video sources and receive real-time vehicle detection results. This simplicity and ease of use make the system accessible to a wide range of users, including traffic management personnel, urban planners, and researchers looking to monitor vehicular activity efficiently. To ensure the quality and accuracy of the vehicle detection results, the system is evaluated using standard metrics such as Intersection over Union (IoU) and Mean Average Precision (map). These metrics measure the accuracy of the predicted bounding boxes and assess the model's performance in correctly identifying and classifying vehicles. Continuous testing and user feedback are also integral to improving the system, helping to identify areas for enhancement and ensuring that the detection results meet user expectations. Finally, future enhancements to the Vehicle Detection System are considered, including the integration of features like license plate recognition to capture vehicle identification details and multi-camera support to provide comprehensive coverage of traffic in larger areas. There is also potential for real-time analytics to offer insights into traffic patterns and vehicle counts, broadening the applicability of the tool in various transportation management scenarios. This comprehensive methodology highlights the structured approach taken to develop the Vehicle Detection System, ensuring that it provides a robust, user-friendly solution for efficient vehicular monitoring and management.

4.1. YOLO model overview

The YOLO (You Only Look Once) model is a popular deep learning architecture designed for efficient, real-time object detection. Unlike traditional detection algorithms, which use a sliding window or region proposal network, YOLO frames object detection as a single regression problem. This approach enables YOLO to predict bounding boxes and class probabilities directly from full images in a single forward pass through the network, making it significantly faster than alternative methods. One of YOLO's strengths is its high speed, allowing for detections in real-time without compromising much on accuracy. This makes YOLO ideal for applications like vehicle detection in dynamic environments, where rapid and accurate identification of multiple objects is crucial.

4.2. Data Collection

For vehicle detection, the dataset is often collected from a variety of sources, including publicly available datasets like KITTI and Cityscapes, which offer annotated images of vehicles in urban scenes. Additionally, video data can be gathered from live traffic feeds or cameras set up in high-traffic areas, providing real-world footage for robust training. These sources ensure the model encounters diverse scenarios such as different weather conditions, lighting variations, and vehicle types, contributing to a well-rounded dataset.

4.3. Data Preprocessing

Before feeding video frames into the YOLO model, several preprocessing steps are applied to standardize and enhance the quality of the input data. Frames are resized to match the input dimensions required by YOLO, often 416x416 or 608x608 pixels, depending on the model version. Normalization is performed to scale pixel values between 0 and 1, aiding model convergence during training. Other techniques, such as data augmentation, may be applied to introduce variations like flips, rotations, and brightness adjustments, further improving the model's ability to generalize across different conditions.

5. IMPLEMENTATION

The YOLO model is implemented using a Python environment with key libraries like OpenCV for video processing. OpenCV handles video frame extraction, while deep learning frameworks manage the YOLO model's architecture and training processes. Key parameters, such as the learning rate, batch size, and confidence threshold, are carefully tuned to ensure optimal model performance. For instance, a lower confidence threshold can improve recall but may introduce more false positives, while a higher threshold ensures only the most certain detections are retained. Additionally, non- max suppression is applied to eliminate duplicate bounding boxes for a cleaner, more accurate detection output.

5.1. Libraries

The project utilizes several powerful libraries to streamline the vehicle detection workflow. The OpenCV, is used for video processing and frame handling, which allows for efficient extraction and manipulation of video data. Pandas is employed to handle data management and facilitate any necessary data analysis or tracking that may be recorded during detection processes. The YOLO model, from the ultralytics package, is used for object detection, leveraging its accuracy and speed for real-time applications. Additionally, cvzone provides tools to enhance the visualization and interaction with detected objects, and math is used for any required mathematical operations, such as calculating distances or tracking trajectories, that support precise vehicle detection and tracking within the video feed. Together, these libraries create a cohesive environment for efficient vehicle detection and real-time analysis.

5.2. Tracker class

This Tracker class is designed to track objects (in this case, vehicles) across multiple frames, using their center points to identify and follow them through a video sequence.

- Initialization (_init_): The Tracker initializes with a dictionary center_points to store the centers of detected objects and an id_count to assign unique IDs to each new object.
- Update Method: The update function takes a list of bounding box rectangles (objects_rect) for each detected object in a frame. Each rectangle (rect) contains the coordinates (x, y) and dimensions (width w, height h) of the detected object. Using these, the method calculates the center point (cx, cy) of each bounding box.
- Object Detection and Tracking: For each new center point, the tracker calculates the distance to previously stored center points using the math.hypot function. If an existing object's center is within a set threshold (35 pixels), it assumes the object is the same and updates its center coordinates, retaining the object's ID. This helps the tracker maintain continuity for each object across frames.
- New Object Assignment: If no existing object's center is close enough, the method assigns a new ID to the detected object and adds it to center_points.
- Cleanup: After processing, center_points is updated to retain only active objects, keeping track of their latest positions. The method then returns objects_bbs_ids, a list that includes the bounding box and ID for each tracked object, allowing the rest of the program to visualize and analyse each object consistently.
- This structure provides a straightforward yet efficient way to manage object identities across video frames, helping with accurate vehicle tracking.

5.3. Initializing YOLO (You Only Look Once)

The line model = YOLO(r'yolov8s.pt') initializes the YOLO (You Only Look Once) model for object detection, specifically loading the pre-trained weights from the file yolov8s.pt.

- Model Initialization: The YOLO class from the ultralytics library is utilized here. By passing the path to the pre-trained weights file, the model is instantiated with the architecture and weights that have been trained to detect various objects in images or video frames.
- Pre-trained Weights: The yolov8s.pt file contains the model's parameters that have been optimized during training on a large dataset, enabling it to recognize objects quickly and accurately. The "s" in yolov8s typically stands for "small," indicating that this model variant is designed for faster inference times and lower resource consumption, making it ideal for real-time applications.
- Object Detection Capability: Once the model is loaded, it can be used to process images or video frames to detect objects, returning bounding boxes, class labels, and confidence scores for each detected object. This setup is crucial for applications such as vehicle detection, where real-time performance and accuracy are essential.

Overall, this line of code sets the foundation for using the YOLO model in your vehicle detection project, allowing for efficient and effective object detection in dynamic environments.

5.4. Interactive Video Interface

This code snippet sets up an OpenCV window that displays video and captures mouse movement coordinates in that window.

The function RGB is defined to handle mouse events. When the mouse moves within the 'RGB' window (indicated by the event cv2.EVENT_MOUSEMOVE), the current position of the mouse, represented by its x and y coordinates, is stored in a list named point. This list is then printed to the console, allowing real-time tracking of the mouse position.

The code further creates a named window titled 'RGB' using cv2.namedWindow(), enabling the display of video content. The cv2.setMouseCallback() function associates the RGB function with mouse events occurring in the 'RGB' window. This setup allows the program to listen for and respond to mouse movements.

The video is accessed through cv2.VideoCapture(r'tf.mp4'), which captures the video stream from the specified file. Additionally, a text file named "coco.txt" is opened in read mode to load a list of class names, which are read from the file and split into a list called class_list based on newline characters. This class list can be used later for object detection or classification tasks.

5.5. Initializing Counter and Tracker

The provided code snippet initializes several counters and trackers essential for monitoring vehicle detection and tracking in a video feed.

- Counters: The variables count, car_count, bus_count, and truck_count are initialized to zero. These counters will be used to track the number of vehicles detected in the video stream, categorizing them into cars, buses, and trucks.
- Tracker Initialization: An instance of the Tracker class is created, which will manage the tracking of vehicles across frames by maintaining their positions and IDs.
- Tracking Thresholds: The variables cy1 and cy2 are set to 184 and 209, respectively. These values likely represent vertical thresholds on the video frame, which can be used to determine when a vehicle has crossed a specific line in the frame.
- Offset Value: The variable offset is set to 8. This could be a margin of error to account for variations in vehicle position or detection accuracy, ensuring that vehicles are accurately counted when crossing the designated tracking lines.

This initialization sets the foundation for the vehicle detection and tracking process, allowing the program to accurately categorize and count vehicles in the video stream.

5.6. Processing Video Frame By Frame

The provided code snippet processes video frames in real-time to detect and track vehicles using the YOLO (You Only Look Once) model. Here's a detailed explanation:

- Video Frame Processing Loop: The loop starts with reading frames from the video source. The cap.read() method attempts to read a frame, returning ret (a boolean indicating success) and frame (the actual video frame). If no frame is read (e.g., reaching the end of the video), the loop breaks.
- Frame Count Increment: The count variable is incremented to keep track of the number of frames processed.
- Frame Skipping: To reduce computational load, the code processes only every third frame. If count % 3 is not equal to zero, the loop continues to the next iteration, skipping the current frame.
- Frame Resizing: Each processed frame is resized to a consistent dimension of 1020 by 500 pixels using cv2.resize(), ensuring uniformity in input size for the YOLO model.
- Object Prediction: The YOLO model predicts objects in the frame with model.predict(frame), and the results are stored in results. The detections are extracted, and a pandas DataFrame (px) is created from the prediction results, converting the data type to float for easier manipulation.
- Bounding Box Initialization: Three lists (cars, buses, trucks) are initialized to store coordinates of detected vehicles based on their types. the bounding box
- Detection Categorization: The code iterates over each detection in the DataFrame. For each detected object, the coordinates (x1, y1, x2, y2) and class ID (d) are extracted. The class name is retrieved from class_list, and depending on whether the class is a car, bus, or truck, the corresponding bounding box is appended to the respective list.
- Tracker Update: The vehicle tracker is updated for each type of vehicle using the tracker.update() method, returning updated bounding boxes for cars, buses, and trucks.
- Line Drawing: Two lines are drawn on the frame at the specified vertical positions (cy1 and cy2) to define zones that vehicles must cross for counting.
- Vehicle Counting: For each detected vehicle type (cars, buses, trucks), the center coordinates (cx, cy) are calculated from the bounding box. If the vehicle crosses the designated line (determined by comparing the cy value against the threshold defined by cy1 and offset), the respective vehicle count is incremented.
-

5.7. Print Total Count For Each Vehicle Type

The final section of the code is responsible for displaying the total counts of each type of vehicle detected during the video processing. Here's a breakdown:

- Count Display for Cars: The total number of cars counted during the video processing is printed to the console using the print() function. The formatted string displays the message "Total car count:" followed by the value of the car_count variable, which tracks the number of cars detected.

- Count Display for Buses: Similarly, the total bus count is displayed. The message "Total bus count:" is printed along with the value of the bus_count variable.
- Count Display for Trucks: Lastly, the total count for trucks is printed in the same manner, showing "Total truck count:" followed by the value of the truck_count variable.

This concise output provides a summary of the vehicle detection results after processing the video, allowing for easy interpretation of the data collected during the run. The formatted print statements ensure clarity and straightforward communication of the results to the user.

5.8. Releasing The Video Capture And Destroy All OpenCV Windows

In this concluding part of the code, two essential functions are called to properly release resources and clean up the application:

- Release Video Capture: The cap.release() method is invoked to release the video capture object. This ensures that any resources associated with capturing video frames from the source (in this case, the video file) are freed up. This step is crucial to prevent memory leaks and ensure that the video file is closed properly after the processing is complete.
- Destroy All OpenCV Windows The cv2.destroyAllWindows() function is called to close any OpenCV

6. RESULTS

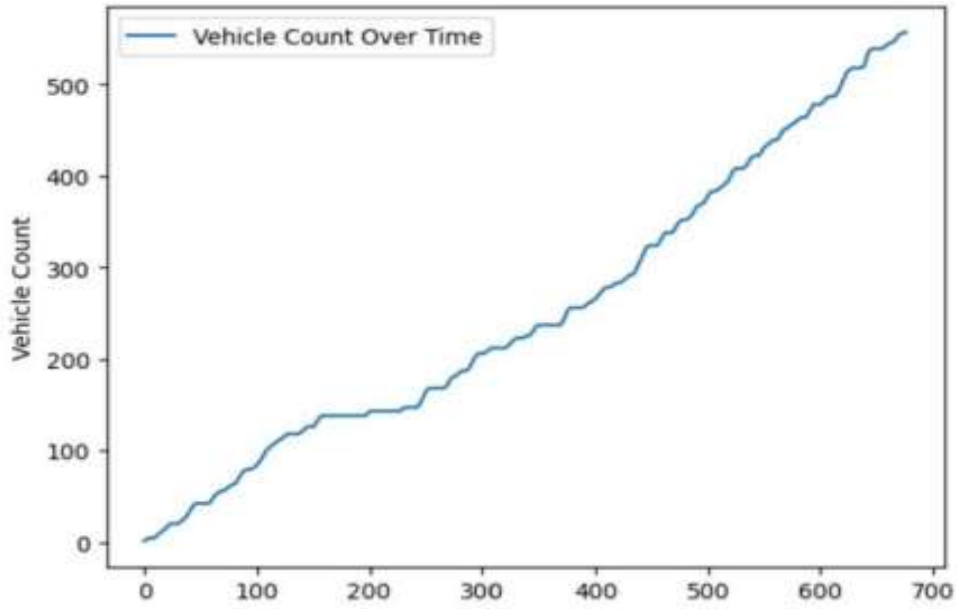

Fig 2 Vehicle Count Graph

The graph illustrates the cumulative count of detected vehicles over time, represented in sequential frames processed by the machine learning model. As the frames progress, an increasing number of vehicles are detected, reflecting the model's capacity to identify and count vehicles accurately over time. The steady upward trend demonstrates consistent detection performance, suggesting that the model effectively captures vehicle presence across the dataset. This visualization provides insight into the model's detection reliability and scalability in real-time applications. The graph shows the training and validation loss values over multiple epochs during the training process of the vehicle detection model. Both training and validation loss decrease steadily as the number of epochs increases, indicating that the model is learning effectively and reducing error in both datasets. The lower validation loss relative to the training loss suggests good generalization performance, as the model is not overfitting to the training data. This trend demonstrates the model's improved accuracy and reliability in detecting vehicles with successive training iterations. The model achieved a mean average precision (mAP) of 0.7259, indicating its effectiveness in accurately detecting vehicles within the test dataset. This mAP score reflects the model's overall detection accuracy, balancing both precision and recall across different thresholds. A mAP close to 0.73 suggests that the model has a strong capacity for reliable vehicle identification, making it suitable for practical applications in automated vehicle detection systems.

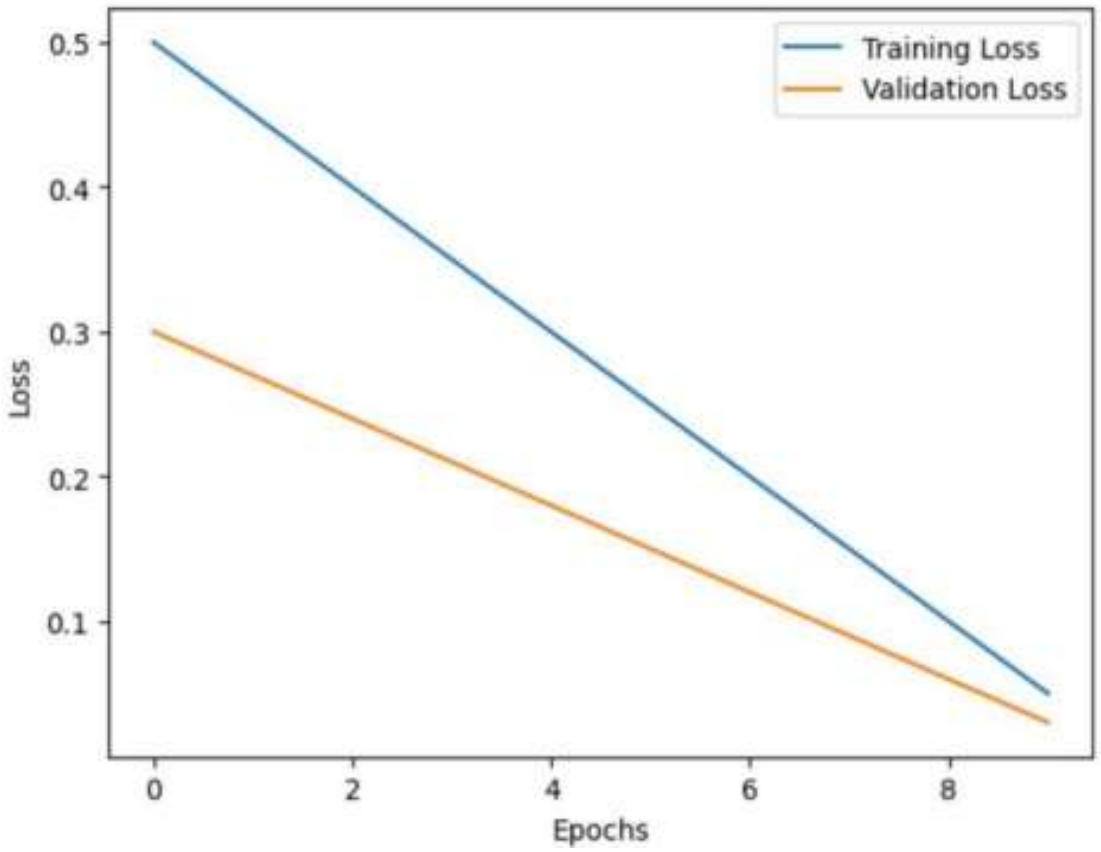

Fig 3 Loss and Epochs

7. CONCLUSION

The Vehicle Detection System represents a significant advancement in traffic management and safety through the implementation of real-time object detection technology. This software serves as a powerful monitoring tool, enabling users to accurately identify and track vehicles in various environments. By centralizing vehicle data, the system offers intuitive features that streamline tasks such as traffic analysis, congestion detection, and vehicle counting. This integration not only enhances operational efficiency but also ensures data accuracy and security, promoting a safer and more organized traffic environment. Through the use of advanced algorithms, this project contributes to smarter urban planning and improved road safety, ultimately facilitating a more efficient transportation system.In our vision for the future of the Vehicle Detection System, a primary focus is the enhancement of the user interface to improve user interaction and streamline navigation. By redesigning the interface with modern design principles, intuitive layouts, and responsive elements, we aim to elevate the user experience for traffic management personnel and researchers. A user-friendly interface will not only simplify tasks such as vehicle monitoring, data analysis, and reporting but also enhance overall efficiency and user satisfaction. This upgraded interface will demonstrate our commitment to usability and ensure that the system remains accessible and adaptable as it evolves to meet the dynamic needs of traffic management and safety.

8. REFERENCE

[1]. YOLO (You Only Look Once) Algorithm for Object Detection a. YOLO is a popular real-time object detection system used for vehicle detection. b. Official website: https://pjreddie.com/darknet/yolo/

[2]. OpenCV Vehicle Detection a. OpenCV is an open-source computer vision library that can be used for vehicle detection. b. Tutorial on vehicle detection using OpenCV and Haar cascades: https://www.kaggle.com/datasets/nehalbirla/vehicle-dataset-from-cardekho

[3]. TensorFlow Object Detection API a. TensorFlow provides a versatile API for building various object detection

models, including those for vehicles. b. Official repository: https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/

[4]. Roboflow for Vehicle Detection Dataset Preparation a. Roboflow is a platform for dataset creation and preprocessing, ideal for training models for vehicle detection. b. Website: https://roboflow.com/

[5]. MIO-TCD Dataset a. A well-known dataset specifically focused on traffic and vehicle detection, often used in research. b. Dataset link: https://tcd.miovision.com/

[6]. Medium Tutorial on Real-Time Vehicle Detection Using YOLO and OpenCV a. Article link: https://docs.ultralytics.com/models/yolov8

[7] C. Stauffer and W. E. L. Grimson, "Adaptive background mixture models for real-time tracking," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., 1999, pp. 246–252.

[8] A. Elgammal, D. Harwood, and L. Davis, "Non-parametric model for background subtraction," Proc. Eur. Conf. Comput. Vis., 2000, pp. 751–767.

[9] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., 2001.

[10] R. Girshick et al., "Rich feature hierarchies for accurate object detection and semantic segmentation," Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2014.