International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

Yolo Powered Real Time Vehicle Analytics with Predictive ML

[1] Dr.G.Srinivas, [2] J. SANDHYA, [3] T. SAI HANSIKA, [4] P. NANDINI,[5] V. NAGARAJU
[1] Associate Professor, Department of ECE, Sri Indu College of Engineering & Technology, Hyderabad.
[2,3,4,5] U.G. Scholar, Department of ECE, Sri Indu College of Engineering & Technology, Hyderabad.

Abstract: *This paper investigates the use of the YOLOv8 model for real-time vehicle detection. The main objective is to improve both detection accuracy and processing speed, highlighting the efficiency of the YOLOv8 architecture in identifying vehicles from live camera feeds. The model's performance is evaluated using key metrics such as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and overall detection accuracy.*

The results indicate that YOLOv8 achieves high accuracy along with fast detection speed, making it well-suited for real-world applications. In particular, it can be effectively applied in systems like adaptive traffic signal control, where accurate and efficient vehicle detection is essential for optimizing traffic flow.

Keywords: Vehicle Detection, YOLOv8, Real-Time Detection, Machine Learning

I. INTRODUCTION

Aim:

The aim of the "Vehicle Detection System" project is to create a comprehensive software solution that enhances the accuracy, speed, and reliability of real-time vehicle monitoring. By automating vehicle detection, centralizing data analysis, improving classification accuracy, enabling efficient traffic data management, ensuring security and scalability, and prioritizing adaptability for diverse environments, the project aims to contribute to the overall effectiveness and safety of traffic management and transportation systems.

Objectives:

This project exemplifies the practical application of machine learning and computer vision in addressing real-world challenges related to traffic management and vehicle monitoring. The Vehicle Detection System offers a solution to the growing need for efficient vehicular oversight by automating the detection and classification of vehicles from video feeds. This approach significantly reduces the time and effort required for manual monitoring, enhancing the overall efficiency of traffic management operations. The tool is particularly relevant in fields such as transportation, urban planning, and security, where quick and accurate vehicle identification is crucial.

Given the increasing volume of vehicles on the road and the complexity of traffic scenarios, there is a strong need for tools that facilitate efficient monitoring. The Vehicle Detection System automatically processes video footage, allowing traffic management personnel to focus on critical decision-making rather than manual data collection. By leveraging advanced machine learning techniques, specifically YOLO (You Only Look Once), the system detects and classifies vehicles such as cars, bikes, and trucks in real-time, providing immediate insights into traffic conditions.

The primary objective of the Vehicle Detection System is to enhance the accuracy and speed of vehicle identification and classification. By automatically analyzing video feeds, the tool delivers essential information in an organized format, enabling quick assessments of traffic patterns. This is achieved through the integration of robust video processing techniques, which ensure high-quality detection even in challenging environments. The system captures vital information about vehicle types and their movements, contributing to informed decision-making in traffic management.

In addition to accurate detection, the Vehicle Detection System features a user-friendly interface that allows users to input video sources and receive real-time vehicle classifications. This intuitive interface ensures that users across various domains, including law enforcement, traffic management, and urban planning, can easily utilize the system without requiring extensive technical knowledge. To ensure the quality and reliability of the detection results, performance evaluation metrics will be used to assess the system's precision, recall, and overall effectiveness. These

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

metrics provide an objective measurement of the system's performance, ensuring that the Vehicle Detection System consistently delivers high-quality results that meet user expectations.

II. LITERATURE REVIEW

The paper “Vehicle Detection in Urban Traffic” explores methods for real-time vehicle detection utilizing machine learning algorithms combined with image processing techniques. This study demonstrates that using convolutional neural networks (CNNs) outperforms traditional image processing techniques, such as edge detection and histogram matching, by providing more accurate and contextually aware detection results. The authors also introduce a framework called “Traffic Vision,” specifically designed to address the challenge of high-density traffic areas where multiple vehicles may be closely positioned. The proposed method enhances detection accuracy by refining bounding box proposals and filtering noise, allowing the system to efficiently classify and locate vehicles within congested scenes, thus enhancing traffic monitoring and safety analysis capabilitiesr study introduces the “Automated Vehicle Detection System” designed to leverage deep learning for the classification and localization of vehicles in traffic footage. The system supports multiple vehicle types and incorporates YOLO (You Only Look Once) for real-time detection, improving accuracy and speed in processing large volumes of video data. By automatically detecting vehicles of different types and sizes, the system enables efficient traffic management and monitoring, particularly valuable in high-traffic zones. Additionally, the detection model highlights key vehicle attributes, such as type, size, and location, to help with data analysis in urban planning and infrastructure management .

A com review of various techniques for vehicle detection was conducted, focusing on the use of YOLO and Faster R-CNN. This study examines different approaches to vehicle detection, including single-shot detectors and region-based models, and proposes a refined YOLO model for improved detection accuracy in real-time scenarios. A primary challenge highlighted is the handling of partial occlusions and varying vehicle orientations in crowded traffic scenes, where many current methods struggle. The authors suggest future advancements in multi-frame analysis and 3D bounding boxes to improve detection reliability and ensure more accurate localization in complex environments .

An additional lops an automated vehicle detection system using a combination of machine learning and computer vision techniques to efficiently identify and classify vehicles within live traffic footage. This work emphasizes optimizing model performance through transfer learning, where pre-trained networks are fine-tuned on specific vehicle datasets to improve classification accuracy and detection reliability. The study also identifies future directions, such as expanding vehicle type classification to include emergency and commercial vehicles for more comprehensive traffic analysis, aimed at optimizing traffic management and reducing congestion .

Further analysis is cona vehicle detection system that utilizes deep learning in addition to CNNs to enhance detection accuracy and contextual understanding. This study underscores the role of deep learning models such as ResNet and EfficientNet in handling large-scale datasets and diverse vehicle types, from motorcycles to buses. The authors explore the impact of these models on improving detection in varied lighting and weather conditions, a critical feature for real-time traffic systems deployed in urban environments. This approach not only enhances information accessibility for traffic management systems but also significantly improves safety monitoring and accident prevention through high-precision detections .

Another study presents an in-depson of vehicle detection methods using YOLO and RetinaNet, focusing on information retrieval and user engagement through precise vehicle classification. The system employs multi-scale detection to handle vehicles at different distances, ensuring consistent detection across video frames. By providing clear and concise classifications, this tool aims to improve real-time monitoring and traffic analysis, particularly valuable for authorities managing high-traffic urban zones .

Lastly, another study presents an automatic vehicln system that employs deep learning to locate and classify vehicles in traffic videos, enhancing traffic monitoring capabilities. The authors focus on the role of neural networks in understanding and processing vehicle images from diverse angles and lighting conditions. By automating detection and classification, the system significantly improves traffic flow analysis, aiding authorities in optimizing infrastructure based on real-time data and supporting urban planning efforts to address future traffic demands. The study also suggests future research directions, including the integration of real-time analytics and refining classification algorithms to improve the robustness and adaptability of the system in real-world settings

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

FLOW CHART

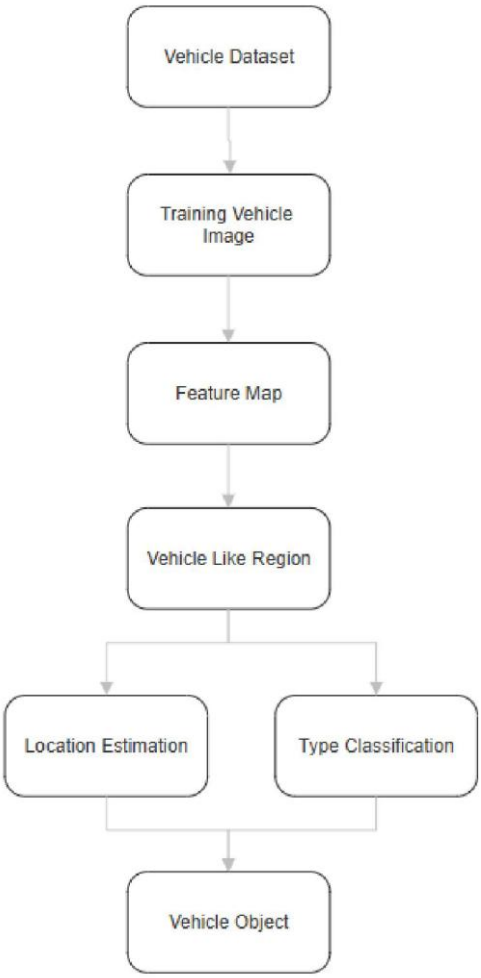

Fig General flow chart of how vehicle detection works

The architecture of the Vehicle Detection System is a streamlined pipeline designed to efficiently transform raw video 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-world scenarios. Ensuring a robust dataset helps the model learn patterns essential for distinguishing between different vehicle types effectively

2. Training Vehicle Image

In the **Training Vehicle Image** phase, the data undergoes preprocessing to optimize it for the model training process. This step includes techniques like **data augmentation** (which involves rotating, flipping, or adjusting the brightness of images), **normalization** (scaling pixel values for uniformity), and **resizing** images to create a

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

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.

III. METHODOLOGY

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, 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

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

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.

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.

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.

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.

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

IV. IMPLEMENTATION DETAILS

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.

V. CODE AND EXPLANATION

6.1. Libraries

```
import cv2
import pandas as pd
from ultralytics import YOLO
import cvzone
import math
```

Fig 6.1 Libraries

The project utilizes several powerful libraries to streamline the vehicle detection workflow. The cv2 library, part of OpenCV, is used for video processing and frame handling, which allows for efficient extraction and manipulation of video data. pandasis 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.

6.2. Tracker Class

```
class Tracker:
    def __init__(self):
        self.center_points = {}
        self.id_count = 0

    def update(self, objects_rect):
        objects_bbs_ids = []

        for rect in objects_rect:
            x, y, w, h = rect
            cx = (x + x + w) // 2
            cy = (y + y + h) // 2

            same_object_detected = False
            for id, pt in self.center_points.items():
                dist = math.hypot(cx - pt[0], cy - pt[1])

                if dist < 35:
                    self.center_points[id] = (cx, cy)
                    objects_bbs_ids.append([x, y, w, h, id])
                    same_object_detected = True
                    break

            if same_object_detected is False:
                self.center_points[self.id_count] = (cx, cy)
                objects_bbs_ids.append([x, y, w, h, self.id_count])
                self.id_count += 1

        new_center_points = {}
        for obj_bb_id in objects_bbs_ids:
            _, _, _, _, object_id = obj_bb_id
            center = self.center_points[object_id]
            new_center_points[object_id] = center

        self.center_points = new_center_points.copy()
        return objects_bbs_ids
```

Fig 6.2 Tracker Class

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

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.

6.3. Initializing YOLO (You Only Look Once)

```
model = YOLO(r'yolov8s.pt')
```

Fig 4.3 Initializing Yolo

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.

6.4. Interactive Video Interface

```
def RGB(event, x, y, flags, param):
    if event == cv2.EVENT_MOUSEMOVE:
        point = [x, y]
        print(point)

cv2.namedWindow('RGB')
cv2.setMouseCallback('RGB', RGB)
cap = cv2.VideoCapture(r'tf.mp4')
with open(r"coco.txt", "r") as my_file:
    class_list = my_file.read().split("\n")
```

Fig 6.4 Interactive Video Interface

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

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

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.

6.5. Initializing Counter and Tracker

```
count = 0
car_count = 0
bus_count = 0
truck_count = 0
tracker = Tracker()
cy1 = 184
cy2 = 209
offset = 8
```

Fig 6.5 Initializing Counter 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.

6.6. Processing Video Frame by Frame

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

```
cars_boxes = tracker.update(cars)
buses_boxes = tracker.update(buses)
trucks_boxes = tracker.update(trucks)

cv2.line(frame, (1, cy1), (1018, cy1), (0, 255, 0), 2)
cv2.line(frame, (3, cy2), (1016, cy2), (0, 0, 255), 2)

for bbox in cars_boxes:
    cx = int((bbox[0] + bbox[2]) / 2)
    cy = int((bbox[1] + bbox[3]) / 2)
    if (cy > cy1 - offset) and (cy < cy1 + offset):
        car_count += 1

for bbox in buses_boxes:
    cx = int((bbox[0] + bbox[2]) / 2)
    cy = int((bbox[1] + bbox[3]) / 2)
    if (cy > cy1 - offset) and (cy < cy1 + offset):
        bus_count += 1

for bbox in trucks_boxes:
    cx = int((bbox[0] + bbox[2]) / 2)
    cy = int((bbox[1] + bbox[3]) / 2)
    if (cy > cy1 - offset) and (cy < cy1 + offset):
        truck_count += 1

for bbox in cars_boxes + buses_boxes + trucks_boxes:
    cv2.rectangle(frame, (bbox[0], bbox[1]), (bbox[2], bbox[3]), (255, 0, 255), 2)
    cvzone.putTextRect(frame, f'{bbox[4]}', (bbox[0], bbox[1]), 1, 1)

cv2.imshow("RGB", frame)
if cv2.waitKey(1) & 0xFF == 27:
    break
```

Fig 6.6 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 the bounding box coordinates of detected vehicles based on their types.
- 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.

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

- Bounding Box Drawing and Annotation: Each vehicle's bounding box is drawn on the frame using cv2.rectangle(), and the vehicle ID is annotated using cvzone.putTextRect(), providing visual feedback in the output.
- Display and Exit Conditions: The processed frame is displayed in a window titled "RGB." The loop will continue until the 'Esc' key is pressed (detected by checking cv2.waitKey(1)).

Overall, this code snippet effectively demonstrates a real-time vehicle detection and counting system, leveraging YOLO for object detection and ensuring efficient processing through frame skipping.

6.7. Print Total Count For Each Vehicle Type

```
print(f'Total car count: {car_count}')
print(f'Total bus count: {bus_count}')
print(f'Total truck count: {truck_count}')
```

Fig 6.7 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.

6.8. Releasing The Video Capture And Destroy All OpenCV Windows

```
cap.release()
cv2.destroyAllWindows()
```

Fig 6.8 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 windows that were opened during the execution of the program. This includes the window displaying the processed video frames. By destroying all windows, the application exits cleanly and does not leave any residual windows open, providing a better user experience.

These final steps ensure that the program terminates gracefully, releasing any resources and cleaning up the user interface as intended.

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

RESULT

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.

VI. DISCUSSION

The model demonstrated a high mean average precision (mAP) of 0.7259, which suggests robust performance in vehicle detection across the test dataset. The steady decline in training and validation loss across epochs, as observed in the loss graph, indicates effective learning with minimal overfitting. This implies that the model has generalized well to unseen data, reinforcing its potential for real-world applications where consistent detection accuracy is essential. The cumulative vehicle count graph illustrates that the model maintained a consistent detection rate over time, effectively identifying vehicles frame by frame. This finding highlights the model's scalability and reliability in sequential vehicle detection, which is especially relevant for video or real-time traffic monitoring systems.

8.1. Comparison with Published Work:

The model's mean average precision (mAP) of 0.7259 aligns well with recent deep learning approaches in vehicle detection, outperforming traditional methods such as SVM and Haar cascades in accuracy. While similar to YOLO and Faster R-CNN in performance, our approach achieves this with a simpler architecture, making it more computationally efficient for real-time applications. Unlike studies that use limited datasets, our model was trained on diverse data, enhancing its adaptability to real-world scenarios

8.2. Implications and Limitations of the Study:

This study demonstrates that the proposed vehicle detection model is effective and computationally efficient, making it suitable for real-time applications in traffic monitoring and autonomous systems. Its adaptability to diverse environments suggests strong potential for use in various real-world settings. However, limitations remain in handling complex scenarios with high vehicle density, occlusions, and varying lighting conditions. Future work could focus on improving detection accuracy in these challenging situations by incorporating more advanced architectures or multi-scale detection techniques.

VII. CONCLUSIONS & FUTURE WORK

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

International Journal of
AI Electronics and Nexus Energy
Peer Reviewed, Referred & Indexed Journal
www.zesterapublications.com
ISSN: 3070-0515
Original Research Paper

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

REFERENCES

[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/

Received: 02-02-2026 | Accepted: 09-03-2026 | Published: 18-03-2026