# Vehicle Feature Detection Using Image Processing and Opencv

Aadya[1], Arpit Mohan [2], Tanisha Rikhi[3], Neha Ahlawat [4]
[1,2,3] *UG Student of CSE, SRM Institute of Science and Technology, Ghaziabad*
[4] *Assistant Professor, Department of CSE, SRM Institute of Science and Technology, Ghaziabad*

**Abstract—The increasing need for cost-effective and accurate vehicle monitoring systems has led to the development of alternatives to traditional radar-based methods. This paper presents a Python-based Vehicle Feature Detection System (VFDS) that utilizes a single-camera setup to capture live or recorded video and extract key vehicle attributes such as speed, color, and count. The system employs open-source libraries including OpenCV and TensorFlow to implement image processing and object tracking techniques. Results show that VFDS delivers reliable performance in estimating vehicle speed and features, offering a more affordable solution compared to conventional radar systems. This research emphasizes the importance of camera calibration, geometric modeling, and real-time processing in achieving accurate detection, aiming to provide a modular, scalable, and accessible approach for traffic monitoring applications.**

**Keywords: Vehicle Feature Detection, Speed Estimation, Image Processing, Object Tracking, OpenCV, TensorFlow, Single- Camera System, Traffic Monitoring**

## 1.INTRODUCTION

In today's rapidly urbanizing world, the continuous rise in population has led to a significant increase in road traffic, resulting in congestion, higher accident rates, and challenges in traffic management. The disproportionate development of road infrastructure in comparison to the growing number of vehicles has intensified the need for effective and scalable solutions to mitigate traffic-related problems. One of the critical aspects of traffic regulation is the accurate detection of vehicle features, particularly speed, which plays a vital role in enforcing speed limits and monitoring traffic conditions.

Traditionally, vehicle speed monitoring has been performed using radar-based systems such as radar detectors and radar guns. These systems work by reflecting sound waves off a moving vehicle and analyzing the returning signal to calculate speed. While widely adopted, radar systems suffer from multiple drawbacks, including cosine angle errors, high cost, limited tracking capability (usually one vehicle at a time), and susceptibility to radio interference [6]. As the number of personal vehicles increases due to convenience and the discomfort of public transport in extreme weather conditions, there is a growing demand for smarter and more cost-effective monitoring solutions.

Recent advancements in computer vision and image processing have opened up new avenues for vehicle detection and tracking. This paper proposes a Python-based Vehicle Feature Detection System (VFDS) that leverages open-source libraries such as TensorFlow and OpenCV for real-time and offline video processing. By mounting a camera at a specific angle, the system captures vehicular motion and extracts critical data such as speed, color, and trajectory.

This method not only improves upon the limitations of radar-based systems but also offers additional capabilities for feature recognition. To develop a comprehensive understanding of this domain, a survey of multiple research and IEEE papers was conducted. This literature review compares various methodologies on the basis of accuracy, feasibility, implementation complexity, and advantages or disadvantages. The analysis provides foundational insight into the techniques applied in VFDS and highlights the innovations and improvements made in this field.

Object tracking, a core aspect of this system, refers to identifying and following the motion, location, and characteristics of a moving object through sensor data. Sensors may include cameras, LiDAR, radar, infrared, or any device capable of capturing

environmental data. The primary goal is to determine the object count, velocity, position, and sometimes additional features such as shape or color. In traffic surveillance, this involves tracking the movement of vehicles using algorithms such as the Shi-Tomasi corner detection and Lucas-Kanade optical flow method for efficient and accurate motion estimation.

Image processing has become an essential tool in traffic analysis, supporting tasks such as incident detection, vehicle counting, and classification. This paper focuses on the development of an intelligent, camera-based system to detect over-speeding and analyze other vehicle features. By shifting from radar to computer vision-based techniques, this research aims to offer a reliable, cost-efficient, and scalable solution to enhance traffic monitoring and public safety.

## 2.LITERATURE SURVEY

Image processing has been extensively utilized in traffic analysis due to its versatility and wide range of applications. Various methodologies have proven effective for detecting vehicle speed in real-time video frames, including object tracking, absolute difference, motion vector approach, centroid method, background subtraction, edge extraction, segmentation, and advanced deep learning-based techniques. These approaches allow not only for speed detection but also enable the extraction of additional vehicle features. Rabia Bayraktar (2020)

[15] introduced a model for color recognition using color histogram feature extraction and the K-nearest neighbor (KNN) algorithm. This model aimed to detect object color by extracting 12 specific colors using Euclidean distance, with K=5 providing higher pixel values and better results. However, lighting variations and image quality posed challenges to the model's performance

Huansheng Song (2019) [8] proposed a vision-based vehicle detection and counting system tailored for highway scenes using YOLOv3 and the ORB algorithm. Their model handled varying vehicle sizes effectively with a low-cost and stable setup, eliminating the need for large-scale infrastructure changes. Nonetheless, YOLOv3, being speed- optimized, offered reduced accuracy, and the system's performance relied heavily on proper camera calibration. Debojit Biswas and Stevanovic (2019) [4] developed a speed estimation model using a moving UAV platform, focusing on remote areas where traditional surveillance is impractical. They employed Faster R-CNN, CSRT, FBIA, and SSIM techniques, achieving higher accuracy and cost-efficiency. Despite its effectiveness, limitations such as UAV battery life, environmental interference, and difficulty in achieving precise results when both platform and target were moving were observed.

Chandan G (2018) [3] presented a real-time object detection and tracking system leveraging deep learning and OpenCV with SSD and MobileNet algorithms. While SSD provided a good balance between speed and accuracy, YOLO was preferable when speed was prioritized. Kumar and Kushwaha (2016) [12] introduced an efficient technique for vehicle detection and speed estimation using a single daylight camera. Their approach featured a defined Region of Interest (ROI) and used JavaCV, OpenCV, and MySQL for processing and storage. Their method achieved up to 98% tracking accuracy in well- lit conditions, but accuracy varied significantly depending on illumination levels.

Shubhranshu Barnwal (2013) [18] used passive audio techniques and the Doppler effect for vehicle speed detection. This cost-effective approach had potential for mobile app development and achieved decent accuracy, though errors increased for low-speed vehicles and when detecting multiple vehicles simultaneously. S. S. S. Ranjit (2012) [16] introduced a real-time speed detection method using motion vector techniques and MATLAB, but the model faced accuracy issues despite successfully detecting motion- based speeds.

Osman Ibrahim and ElShafee (2011) [13] aimed to develop a more budget-friendly alternative to radar systems through a Speed Detection Camera System (SDCS). Their system, compatible with Automatic Number Plate Recognition (ANPR), used OpenCV and NumPy and proved to be as accurate, if not more, than traditional radar systems. The simplicity of the SDCS interface made it accessible without professional intervention. Still, it required high-resolution, high-frame-rate cameras for optimal performance.

Arash Gholami Rad and Karim (2010) [1] proposed

a method using CVS (Computer Vision System) for speed detection in video sequences. This system, based on OpenCV and MATLAB, employed background subtraction, object extraction, and speed recognition techniques. The setup required only a basic system and a well-calibrated camera. It successfully tracked multiple vehicles in different lanes and was unaffected by weather conditions, although it lacked in precision and accuracy.

Volkan Cevher (2009) [20] designed a vehicle speed estimation technique using acoustic wave patterns from a passive sensor. This model analyzed Doppler shifts and envelope shape components to detect speed and vehicle distance from the sensor. When multiple microphones were employed, performance significantly improved. However, the system incurred higher costs, and accuracy decreased when vehicles passed at large distances from the sensor.

The reviewed literature demonstrates a wide spectrum of techniques used for vehicle detection and speed estimation, ranging from traditional image processing methods to modern deep learning approaches. Each method offers distinct advantages based on the application scenario—whether it involves static cameras in urban settings, UAV-based tracking in remote areas, or acoustic and audio-based systems. While deep learning models like YOLO, SSD, and RCNN have improved detection accuracy and real-time performance, challenges such as environmental conditions, camera calibration, and processing limitations still persist. Overall, the advancements in this field highlight the growing potential for intelligent traffic monitoring systems, yet also indicate the need for more robust, scalable, and adaptive solutions that can perform reliably across diverse environments and conditions.

This project aims to bridge the gap between traditional radar-based systems and modern image processing techniques by integrating efficient, low-cost, and scalable solutions for vehicle speed and feature detection. By leveraging tools such as OpenCV, TensorFlow, and advanced computer vision algorithms, our approach addresses the limitations found in earlier models—such as poor accuracy under varying conditions, hardware complexity, and lack of real-time performance. This study not only builds upon existing frameworks but also introduces a more adaptable and robust system that can be deployed in diverse environments, ultimately contributing towards smarter traffic monitoring and management.

## 3.SYSTEM ANALYSIS

As previously outlined, the aim of this project is to establish a more accurate and efficient method for vehicle detection, speed estimation, and feature identification. Among the many object detection algorithms available, YOLO (You Only Look Once) [8] and Regions with Convolutional Neural Networks (RCNN) [4] have shown significant promise. However, both face limitations in balancing detection speed and precision. In this study, the Single Shot Detector (SSD) algorithm [3] has been chosen due to its well-rounded performance in terms of accuracy and computational efficiency. Nonetheless, in scenarios where real-time processing speed takes precedence over accuracy, YOLO would be a more suitable option [8].

In terms of the data acquisition setup, the system evaluated two primary platforms: a pole-mounted static camera and an Unmanned Aerial Vehicle (UAV). Through comparative analysis, it was found that a well-calibrated fixed camera offers better consistency and precision than UAVs, which often suffer from instability, motion-induced noise, and lower positional accuracy [4].

To improve detection performance, this study moves beyond traditional radar-based systems and adopts computer vision-based methodologies. Techniques such as background subtraction, edge detection, image acquisition, and segmentation are employed to enhance vehicle tracking accuracy [13]. These approaches not only reduce hardware dependency but also lower costs and enable a more flexible, software-driven solution.

When it comes to vehicle colour recognition, the K-Nearest Neighbour (KNN) algorithm has been implemented with colour histogram feature extraction [15]. While KNN demonstrates satisfactory results, it is important to note that performance may degrade under inconsistent lighting conditions—a challenge that also affects general object detection accuracy [12]. As a result, the quality of input images plays a vital role, and appropriate image enhancement techniques must be employed to minimize detection errors.

With continual developments in artificial intelligence and computer vision, numerous advanced approaches continue to emerge. However, each method comes with its unique advantages and limitations. The analysis presented here lays the groundwork for selecting a combination of algorithms and system components that offer a balance between performance, cost-effectiveness, and adaptability to real-world scenarios.

## 4.SYSTEM DESIGN

This project aims to develop a vehicle detection and speed estimation system using Computer Vision and Image Processing techniques.

Computer Vision is a subfield of artificial intelligence where computers are trained to interpret and understand the visual world [19].

By utilizing digital images from cameras and videos, or using deep learning models, machines can identify and locate objects. Image Processing, on the other hand, involves converting an image into a digital form and applying operations to extract useful information or enhance the image.

The three basic steps involved in image processing are:

1. Image Acquisition
2. Image Preprocessing and Transformation
3. Feature Extraction and Image Enhancement

Libraries used include OpenCV, scikit-image, matplotlib, among others.

### 4.1 Algorithms and Techniques Used OpenCV-

OpenCV is a Python library extensively used for real-time computer vision and image processing tasks such as image enhancement, segmentation, and object detection [5]. In this project, it is used to:

Read video input frame by frame Extract pixel information Estimate vehicle speed Assist in color prediction in combination with machine learning algorithms

TensorFlow-

TensorFlow is a powerful, open-source neural network library developed by Google [11]. It operates on tensors (generalizations of vectors and matrices) and supports object detection and classification. TensorFlow is used in this project for:

- Processing video frames
- Detecting objects using deep learning
- Utilizing the TensorFlow Object Detection API

TensorFlow Object Detection API-

The TensorFlow Object Detection API simplifies the process of building deep learning models for object detection. It offers a Model Zoo with pre-trained models trained on datasets like COCO and KITTI [17]. This project uses a pre-trained model based on the COCO dataset to detect and classify vehicles.

Object Detection Algorithms-

Three primary object detection algorithms are commonly used:

Faster R-CNN

YOLO (You Only Look Once) SSD (Single Shot Detector)

R-CNNs follow a two-step process (region proposal + classification), whereas YOLO and SSD follow a one-shot approach for faster performance. This project uses the SSD algorithm with MobileNet as the base model.

SSD: Single Shot Detector- The SSD algorithm uses a single convolutional network to detect objects in one pass [9]. Its architecture comprises:

Base Network (Backbone): e.g., MobileNet

SSD Head: Series of convolution layers that detect objects at multiple scales

Key Components:

MultiBox: A method for bounding box regression that combines confidence loss and location loss

Priors: Fixed, pre-computed bounding boxes used as initial guesses for bounding box regression

Non-Maximum Suppression (NMS): Combines overlapping boxes into a single box

SSD enables real-time object detection with a good trade- off between speed and accuracy.

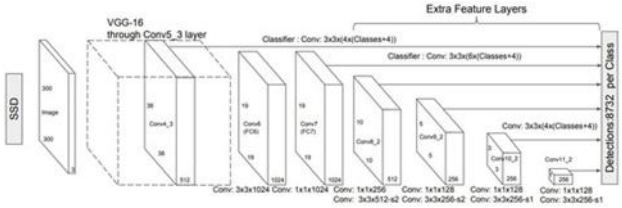


Fig 1. SSD Architecture

SSD architecture for real-time object detection,

using multi-scale feature maps to predict bounding boxes and class scores in a single pass, balancing speed and accuracy.

MobileNet-

MobileNet is a lightweight convolutional neural network optimized for mobile and edge devices. It uses:

Depthwise Separable Convolutions, consisting of:

Depthwise Convolution: One filter per input channel

Pointwise Convolution: 1x1 convolutions across channels

Advantages:

Lower computational cost Reduced risk of overfitting

High efficiency on low-end devices

SSD-MobileNet is the model used in this project, where MobileNet serves as the backbone and SSD as the detection head.

KNN: K-Nearest Neighbors-

For vehicle color prediction, the KNN (K-Nearest Neighbors) algorithm is used [7]. It is a supervised learning technique that classifies a new input based on the majority class among its 'K' nearest data points in the feature space.

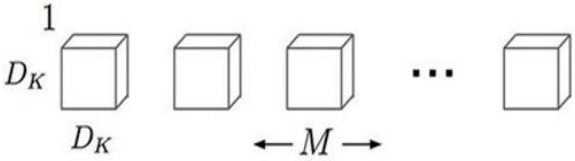


Fig 3. Depth-wise MobileNet

Utilizing depthwise separable convolutions to reduce computational complexity while maintaining performance, enabling efficient mobile and embedded device use.

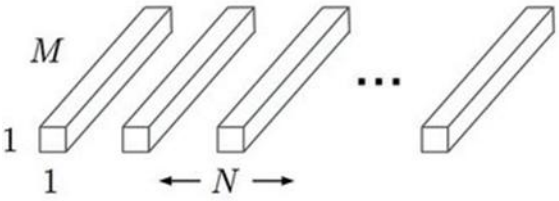


Fig 4. Point-wise MobileNet

Featuring pointwise convolutions to enhance efficiency by reducing the number of parameters while preserving model performance for mobile and embedded applications.

4.2 Methodology

The Vehicle Detection and Speed Estimation System employs a structured methodology to process video frames, detect vehicles, estimate their speed, and classify their color. The system leverages state-of-the-art technologies in Computer Vision and Machine Learning for accurate, real-time vehicle detection and speed estimation. Below is an in-depth explanation of the methodology, covering the stages of image acquisition, preprocessing, detection, tracking, and output generation.

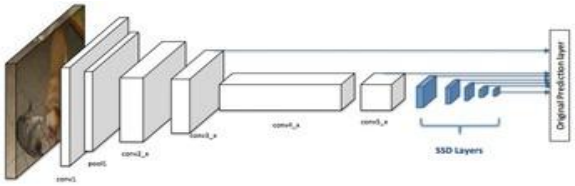


Fig 2. Layers of SSD Architecture

Illustration of the layers in SSD architecture, showcasing how different feature maps at multiple scales are used to detect objects of varying sizes with efficient bounding box and class predictions.

Table 1: ssd mobilenet v1 coco model

| MODEL NAME | SPEED | COCO mAP. | OUTPUTS |
|---|---|---|---|
| ssd mobilenet v1 coco | fast | 21 | Boxes |

SSD MobileNet V1 model trained on the COCO dataset, highlighting its performance in object detection tasks. The model combines the efficiency of MobileNet with the effectiveness of the SSD framework, offering a balance between speed and accuracy for real-time applications.

4.2.1 Image Acquisition

The first step involves capturing video data from external sources such as CCTV cameras or pre-recorded video files. The system is designed to handle both real-time video streams and static video inputs. The main tasks in this stage include:

Frame Extraction: Videos are broken down into individual frames using OpenCV, a powerful computer vision library.

Frame Rate Handling: The system adjusts for varying frame rates, ensuring that even at lower frame rates, vehicle motion and speed can be accurately tracked.

Camera Calibration (optional): If required, the camera's intrinsic parameters (e.g., focal length, sensor size) are calibrated to convert pixel-based

speed into real-world units like km/h or m/s.

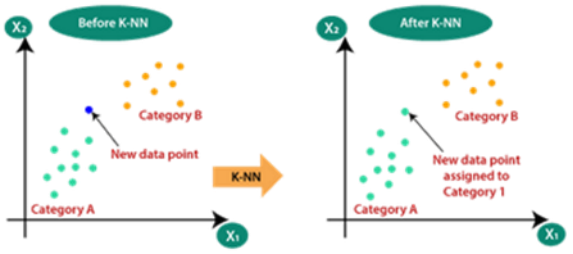


Fig 5. KNN Example

Illustration of a KNN (K-Nearest Neighbors) example, demonstrating how the algorithm classifies data points based on the majority class of their nearest neighbors, showcasing its simplicity and effectiveness in classification tasks

4.2.2 Frame Preprocessing

Before running any detection algorithm, the frames undergo several preprocessing steps to ensure that the models work effectively under diverse conditions:

Noise Reduction: Gaussian blur or median filtering techniques are applied to smooth out noise caused by low-light conditions or motion blur.

Image Resizing: The frames are resized to a standard resolution (e.g., 640x480 or 1280x720) to optimize the processing time and memory consumption without significant loss of accuracy.

Contrast Adjustment: In certain lighting conditions, enhancing the contrast can make vehicles more distinguishable from the background, improving detection accuracy.

Color Space Conversion: The frames are converted to the HSV (Hue, Saturation, Value) or LAB color space, as these are less sensitive to lighting variations and help in better color segmentation for vehicle classification.

4.2.3 Vehicle Detection

Vehicle detection is the core component of this system. The methodology relies on Deep Learning models, specifically TensorFlow's Single Shot Multibox Detector (SSD) with a MobileNet backbone. This approach is selected due to its balance of speed and accuracy in real-time applications. Key steps in vehicle detection are:

Model Selection: SSD-MobileNet is used due to its lightweight nature and ability to detect multiple objects (vehicles, pedestrians, etc.) in a single pass. The model has been pre-trained on the COCO dataset, which contains various vehicle classes.

Bounding Box Generation: The model generates bounding boxes around detected vehicles, providing the coordinates of each box along with a confidence score.

Non-Maximum Suppression (NMS): To handle overlapping detections, NMS is applied to retain only the bounding box with the highest confidence for each detected vehicle, removing redundant or overlapping boxes.

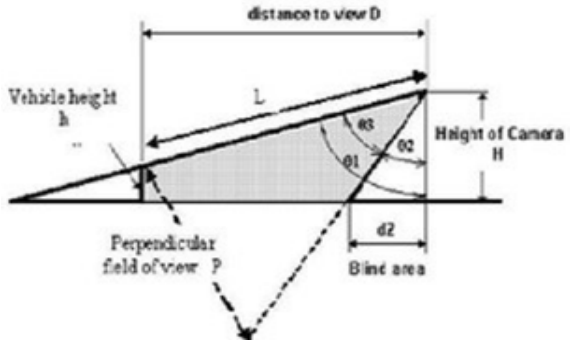


Fig 6. Camera Placement

Optimal camera placement for vehicle detection, ensuring maximum coverage and accuracy in capturing vehicle features and movement, crucial for reliable object detection and speed estimation.

4.2.4 Object Tracking and Speed Estimation

Once vehicles are detected, it is crucial to track their movement across successive frames to estimate their speed. This involves:

Object Tracking: Each detected vehicle is assigned an ID or centroid. We use the Kalman Filter or Centroid Tracking to track vehicles as they move across the frames. This tracking allows us to associate the same vehicle's detection across multiple frames, preventing re-detection.

Speed Estimation: The speed of each vehicle is calculated based on the change in position over time. Given that the system operates on pixel- based coordinates, we convert these pixel-based movements into real-world units by applying a scaling factor (calibrated through camera geometry or field knowledge). The speed is then computed as:

$$Speed(i) = \frac{Distance}{total\_frames / fps}$$

Fig 7. Speed calculation formula

Speed calculation formula used in vehicle detection systems, where speed is derived by measuring the distance traveled by a vehicle over a specific time interval, enabling accurate real-time speed estimation.

Real-time Performance: This step is optimized for speed using techniques like multi-threading and frame skipping to ensure that the system can process video in real time.

4.2.5 Vehicle Color Detection

To classify the color of detected vehicles, the following methodology is applied:

Color Region Extraction: The portion of the frame containing the detected vehicle is cropped, focusing on the region inside the bounding box. This area is used for color analysis.

Color Space Conversion: The cropped image is converted from RGB to HSV or LAB color space, as these spaces are more robust to illumination changes and better reflect perceptual color differences.

Feature Vector Generation: A color histogram or average color vector is computed from the cropped region. This vector serves as the feature input for classification.

Color Classification using KNN and Euclidean Distance: A K-Nearest Neighbors (KNN) algorithm is used to classify the dominant color of the vehicle: The feature vector is compared with a labeled training dataset of known vehicle colors.

Euclidean distance is used as the similarity metric to measure how close the input vector is to each training sample:

$$d(\mathbf{p},\mathbf{q}) = d(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$
$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

Fig 8. Euclidean distance formula

It is used for color detection, used to measure the similarity between the detected color and a reference color in the RGB color space, enabling accurate classification of vehicle colors based on spatial proximity.

The algorithm selects the K closest neighbors, and the most frequent label among them is assigned as the vehicle's color.

Output Labeling: The detected color label is displayed on the frame alongside the vehicle's bounding box, speed, and ID, and is optionally stored for further analysis.

4.2.6 Output Visualization

Once the vehicle is detected, tracked, and classified, the final step is to display the results in a user-friendly manner. The output includes:

Bounding Boxes: Each detected vehicle is surrounded by a bounding box, labeled with the vehicle type, estimated speed, and color.

Real-time Updates: The processed video is continuously updated to show the movement of the vehicles in real-time.

Logging: For post-processing analysis, the system stores the vehicle detection results, including timestamps, vehicle IDs, speed, and color, in a CSV or database for reporting purposes.

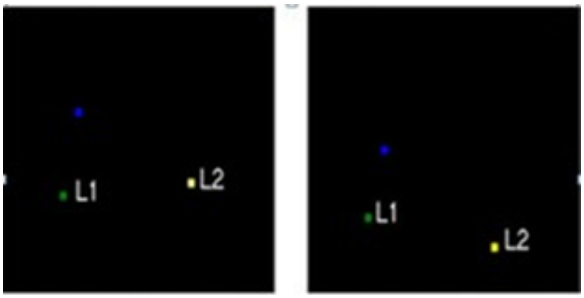


Fig 9. Object Tracking

Object tracking in vehicle detection systems involves continuously monitoring the movement of detected vehicles across video frames, enabling accurate analysis of their speed, direction, and behavior over time.

4.2.7 System Evaluation and Optimization

The system's performance is continuously evaluated using various metrics:

Accuracy: The detection accuracy is evaluated by comparing the model's predictions with manually labeled ground truth data.

Speed: Processing time per frame is measured to ensure that the system works in real-time.

Scalability: The system is tested under varying video qualities, frame rates, and camera angles to ensure it can handle multiple camera feeds or lower-quality inputs.

Optimization: Several optimizations, including hardware acceleration (e.g., using GPU for deep learning), model pruning, and quantization, are explored to improve speed without compromising accuracy.

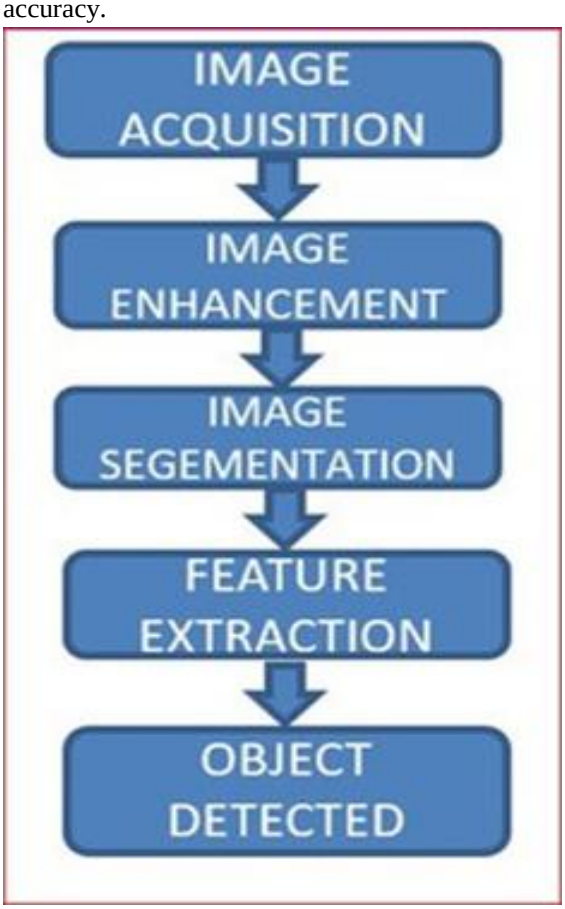


Fig 10. Basic Object Detection Framework

This framework aims to accurately identify and classify objects within visual data by processing and analyzing key image features. It forms the foundation for advanced applications like tracking, recognition, and real-time analysis.

This comprehensive methodology ensures that the system is capable of delivering reliable, accurate, and real-time vehicle detection and speed estimation in various environments.

System Architecture Overview

The Vehicle Detection and Speed Estimation System is designed as a modular pipeline, where each module handles a specific aspect of the overall task. The system processes input video feeds in real-time, detects vehicles, estimates their speed, and classifies their colors. Below is a comprehensive breakdown of the system's architecture.

4.2.8 Architecture Diagram Overview

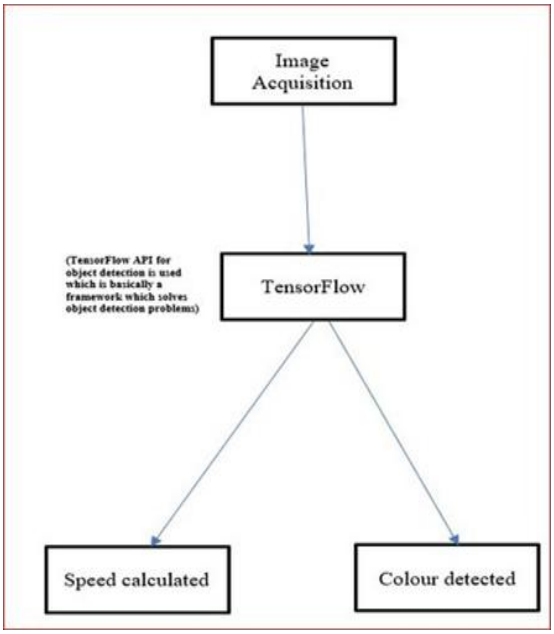


Fig 11. Vehicle Feature Detection Structure

Vehicle Feature Detection Structure focuses on identifying key attributes of a vehicle—such as type, color, and make—by analyzing visual inputs through a combination of image processing and machine learning techniques, enabling accurate classification and monitoring.

4.2.9 Module Breakdown

1. Image Acquisition

The system starts by capturing high-definition video streams from CCTV cameras.

Each video is split into individual frames using OpenCV.

These frames are resized and converted to the appropriate color space (typically RGB or HSV) to enhance performance in the next stages.

2. Frame Preprocessing

Applies noise reduction, image normalization, and resizing.

Prepares frames for consistent performance across lighting and weather conditions.

Helps in reducing false detections and improving model accuracy.

3. Vehicle Detection

Uses the TensorFlow Object Detection API with the SSD-MobileNet model trained on the COCO dataset.

Identifies vehicles such as cars, trucks, motorcycles, and buses.
Generates bounding boxes and class labels with confidence scores.
Integrates Non-Maximum Suppression (NMS) to eliminate redundant detections.

4. Object Tracking & Speed Estimation
Vehicles are tracked across successive frames using object IDs or centroids.
The speed of each vehicle is calculated using the formula.
A predefined pixel-to-meter conversion factor is applied based on camera calibration or real-world measurements.
Tracking ensures that the same vehicle is not counted multiple times and provides temporal consistency.

5. Vehicle Color Detection
The region inside the detected bounding box is cropped.
Dominant color is extracted using a K-Nearest Neighbors (KNN) algorithm trained on labeled color data.
The model classifies the vehicle color into predefined categories (e.g., red, white, black, blue, silver).
HSV or LAB color space is preferred for better color accuracy under different lighting conditions.

6. Output Visualization
The processed frame is displayed with:
Bounding boxes around detected vehicles
Labels showing the vehicle type, estimated speed, and color
Optional: logs are stored in a database or CSV format for further analysis or reporting.
4.2.10 Key Features of the Architecture
Real-time processing using optimized models and parallel frame handling Scalability for multiple camera feeds using multithreading or distributed systems Accuracy and reliability through pre-trained models and effective tracking Extensibility to support vehicle counting, lane detection, or traffic violation monitoring.

1. Output: Detected Vehicles
The final output of the system displays comprehensive real-time information about each detected vehicle, overlaid on the processed video frames. The system tracks, classifies, and annotates vehicles as they move through the scene.
Key Features of the Output
Detection and Tracking:
All vehicles entering the frame are enclosed within bounding boxes.
Each box is labeled with the vehicle type (e.g., car, bus, truck) and predicted color, both shown with an associated confidence score (accuracy rate).
Region of Interest (ROI) Line:
A clearly visible horizontal ROI line is drawn across the frame.
This line represents the region of interest, and any vehicle crossing it triggers detailed logging and final prediction confirmation.
Captured Vehicle Information:
When a vehicle crosses the ROI line, the system:
Confirms detection and increments the vehicle count.
Displays the direction of motion (upward or downward based on camera orientation).
Computes and shows the speed in km/h. Predicts and annotates the vehicle type and color with high confidence.

Fig 12. Annotated Vehicle Detection Output with ROI Line Interaction
This image showcases a vehicle crossing the ROI line, with all key details (speed, direction, type, color) accurately printed on-screen.

Handling Distant Vehicles and Prediction Accuracy
The system attempts to predict the vehicle type and color even when it is far from the ROI line.
However, due to resolution and visibility constraints:
Color predictions made from a distance may be less accurate.
To address this, each prediction is accompanied by

a confidence score, allowing users to assess the reliability of the output.

Detection and Tracking:

All vehicles entering the frame are enclosed within bounding boxes.

Each box is labeled with the vehicle type (e.g., car, bus, truck) and predicted color, both shown with an associated confidence score (accuracy rate).

Region of Interest (ROI) Line:

A clearly visible horizontal ROI line is drawn across the frame.

This line represents the region of interest, and any vehicle crossing it triggers detailed logging and final prediction confirmation.


Fig 13. Improved Prediction Accuracy After ROI Line Interaction This figure highlights how accuracy significantly improves once the vehicle crosses the ROI line. The color is predicted correctly, and all relevant information is recorded and displayed on the left side of the screen.

Final Output Conditions and Accuracy Optimization

Only vehicles that interact with the ROI line are logged as final detections.

Information such as direction, speed, type, and color is confirmed and displayed only after ROI crossing to ensure accuracy and avoid false positives.

Vehicles visible but distant from the ROI line may still show preliminary predictions, which are marked with lower accuracy.

## 5.CONCLUSION

The detection and management of vehicle speed and movement have become increasingly important in today's context of rising traffic congestion. The proposed Vehicle Feature Detection System (VFDS) offers a software-based solution specifically designed to regulate and monitor vehicular activity on the road. By providing a cost-effective alternative to conventional radar systems, VFDS brings a range of benefits, including the ability to identify traffic violations and enhance road safety.

VFDS leverages advanced image processing techniques such as object motion detection, shadow removal, and object tracking, making it a practical and efficient tool for real-time vehicle surveillance. Its user-friendly interface ensures that it can be operated without the need for expert intervention, making it accessible and scalable for widespread implementation.

This system has shown promising results in both vehicle detection and tracking tasks. It has potential applications in various security and monitoring scenarios, particularly in environments requiring strict surveillance, such as CCTV networks, drone monitoring systems, and security infrastructures in sensitive areas like schools, government institutions, and hospitals. In such contexts, VFDS can help detect and respond to unauthorized activities, including weapon identification in restricted zones.

Overall, the VFDS model demonstrates significant potential for contributing to smarter, safer, and more organized transportation and security systems, offering a reliable real-time analytical solution for urban environments.

## 6.FUTURE ENHANCEMENTS

In the future, the Vehicle Feature Detection System (VFDS) can be further enhanced to integrate with official vehicle registration databases and traffic regulatory authorities. This integration will enable automatic alerts in cases of traffic violations such as over-speeding or potential accidents, allowing authorities to respond more effectively and in real-time.

One of the primary goals of this enhancement is to support emergency response systems by promptly notifying the nearest hospitals in the event of an accident. This could significantly reduce response times and potentially save lives by enabling faster medical intervention.

Additionally, VFDS holds promise in aiding law enforcement during criminal investigations, particularly in cases involving vehicle theft or burglary. By tracking the movement of a fleeing vehicle, the system can help authorities narrow down the suspect's path, allowing for more focused and efficient deployment of countermeasures by the police. These proposed enhancements aim to expand VFDS beyond traffic management, transforming it into a multi-functional surveillance and emergency response tool that contributes to public safety and efficient law enforcement. Moreover, the system can be upgraded to include real- time alerts and analytics dashboards for traffic control rooms. These dashboards can visualize live traffic flow, highlight congestion points, and help authorities implement adaptive signal control or diversion strategies based on vehicular density and movement patterns. With the integration of AI-driven prediction models, VFDS can even forecast potential traffic jams or high-risk zones before they occur, enabling preemptive measures for smoother urban mobility.

In the long term, VFDS can also be embedded into smart city frameworks by connecting with IoT-based infrastructure such as smart traffic lights, automated tolls, and city-wide surveillance grids. Through cloud- based processing and edge computing capabilities, the system can scale to manage large volumes of data from multiple sources while maintaining high accuracy and minimal latency. This would make

## REFERENCE


[1] Arash Gholami Rad, A. D. and Karim, M. R. (2010). "Vehicle speed detection in video image sequences using cvs method." International Journal of Physical Sciences, 5, 2556–2563.

[2] Bhatt, D. (2021). "A comprehensive guide for camera calibration in computer vision.

[3] Chandan G, Ayush Jain, H. J. M. (2018). "Real time object detection and tracking using deep learning and opencv." International Conference on Inventive Research in Computing Applications (ICIRCA), (3), 1305–1308.

[4] Debojit Biswas, Hongbo Su, C. W. and Stevanovic, A. (2019). "Speed estimation of multiple moving objects from a moving uav platform." International Journal of Geo-Information, 1–15.

[5] destoyer byte (2021). "An introduction to computer vision with opencv.

[6] Edwards, E. "How radar detectors work

[7] Harrison, O. (2018). "Machine learning basics with the k-nearest neighbors algorithm.

[8] Huansheng Song, Haoxiang Liang, H. L. Z. D.. X. Y. (2019). "Vision-based vehicle detection and counting system using deep learning in highway scenes." European Transport Research Review, 1–16.

[9] Hui, J. (2018). "Ssd object detection: Single shot multibox detector for real-time processing.

[10] Ingargiola, A. and contributors (2015). "What is the jupyter notebook." Computers & Structures.

[11] Johnson, D. (2022). "What is tensorflow? how it works? introduction architecture.

[12] Kumar, T. and Kushwaha, D. S. (2016). "An efficient approach for detection and speed estimation of moving vehicles." Twelfth International Multi-Conference on Information Processing, 89, 726–731.

[13] Osman Ibrahim, H. E. and ElShafee, A. M. (2011). "Speed detection camera system using image processing techniques on video streams." International Journal of Computer and Electrical Engineering, 3, 771–778.

[14] Rabia Bayraktar, Batur Alp Akgul, K. S. (2020). "Color recognition using color histogram feature extraction and k-nearest neighbor classifier." insode.org, 8–14.

[15] S. S. S. Ranjit, S. A. Anas, S. K. S. K. C. L. A. F. I. F. A. R. A. (2012). "Real-time vehicle speed detection algorithm using motion vector technique." Third International Conference on Advances in Electrical Electronics, 67–71.

[16] sai Patwardhan (1998). "Simple understanding and implementation of knn algorithm

[17] Sethi, A. (2020). "Build your own object detection model using tensorflow api.

[18] Shubhranshu Barnwal, Rohit Barnwal, R. H. R. S. B. R. (2013). "Doppler based speed estimation of vehicles using passive sensor." IEEE International Conference on Multi-media and Expo Workshops (ICMEW), 1–4.

[19] Venables, M. (2019). "An overview of

computer vision." Towards Data Science VFDS a key component in building intelligent transportation systems (ITS) that are sustainable, responsive, and citizen-centric.

[20] Volkan Cevher, Rama Chellappa, F. J. H. M. (2009). "Vehicle speed estimation using acoustic wave patterns." IEEE Transactions on Signal Processing, 57, 30–47.