Bulletin of Electrical Engineering and Informatics
Vol. 14, No. 6, December 2025, pp. 4758~4768
ISSN: 2302-9285, DOI: 10.11591/eei.v14i6.10554

Comparative analysis of Haar Cascade, OpenCV, and you only look once algorithms for vehicle detection

Gagandeep Kaur[1], Shital Pawar[2], Rutuja Rajendra Patil[3], Amol Vijay Patil[4], Anuradha V. Yenkikar[4], Nikita Bhandari[5], Kalyani Dhananjay Kadam[6]
[1]Symbiosis Institute of Technology, Nagpur Campus, Symbiosis International (Deemed University), Pune, India
[2]Department of Computer Engineering, Bharati Vidyapeeth's College of Engineering for Women, Pune, India
[3]Department of Computer Engineering, MIT, Academy of Engineering, Alandi, Pune, India
[4]Department of CSE - Artificial Intelligence, Vishwakarma Institute of Technology, Pune, India
[5]Balaji Institute of Technology & Management, Sri Balaji University, Pune, India
[6]Department of Computer Engineering, Vishwakarma University, Pune, India

Article Info

Article history:

Received Apr 24, 2024
Revised Sep 28, 2025
Accepted Oct 14, 2025

Keywords:

Computer vision
Haar cascade classifier
Object detection
OpenCV
Vehicle identification
You only look once

ABSTRACT

Object detection is one of the substantial tasks in computer vision and has a wide range of applications ranging from autonomous driving to monitoring systems. This study presents a comparative analysis of vehicle detection approaches, contrasting traditional methods (OpenCV contour analysis and Haar Cascade) with modern deep learning-based you only look once version 8 (YOLOv8) and its variants. Vehicles were identified and localized within video frames using bounding boxes, with performance assessed through accuracy, F1-score, mean average precision (mAP), and inference speed. YOLOv8 consistently achieved superior accuracy (up to 98% in specific scenarios) and real-time processing speeds (155 FPS), confirming its suitability for safety-critical applications such as intelligent transport systems and autonomous navigation. However, its higher computational and memory demands highlight deployment trade-offs, where lighter variants like YOLOv8s remain feasible for embedded or low-power devices. In contrast, Haar Cascade and contour analysis offered faster execution and smaller memory footprints but lacked robustness under complex environmental conditions. The study also acknowledges limitations such as dataset bias, adverse weather effects, and scalability challenges, which may impact generalization in real-world deployments. By analyzing these trade-offs, the work provides essential insights to guide practitioners in selecting suitable vehicle detection solutions across diverse application environments.

Corresponding Author:

Gagandeep Kaur
Symbiosis Institute of Technology, Nagpur Campus, Symbiosis International (Deemed University)
Pune, India
Email: gagandeep.kaur@sitnagpur.siu.edu.in

1. INTRODUCTION

Over the years, object detection has advanced significantly and has contributed a lot to areas such as autonomous vehicles, industrial robotics, surveillance, and augmented reality among others [1]. Despite these advancements, real-time detection and localization remain challenging, particularly in vehicle detection, where both accuracy and efficiency are critical. Highly optimized models are needed for identifying and locating objects in dynamic environments [2]. To address these challenges, researchers have devised various detection models to improve accuracy, efficiency, and robustness.

Object detection techniques based on the Haar Cascade classifier and OpenCV have been in practice for quite some time because of their simplicity and speed. OpenCV is a mainstream computer vision library that has efficient implementations of Haar Cascade and histograms of oriented gradients (HOG) and can be successfully applied in autonomous driving, robotics, and surveillance systems. Specifically, Haar Cascades make use of a series of simple rectangular features that employ a cascade of weak classifiers to speedily locate the objects of interest [3]. The another major class of algorithms for object detection-with the advent of deep learning-is you only look once (YOLO), which treats detection as one consolidated regression problem [4]. YOLO processes the whole image in one go, which allows for real-time detection with great accuracy and spatial coherence. All versions, particularly YOLOv5, YOLOv7, and YOLOv8 have shown enhanced detection precision, speed, and robustness as compared to classical algorithms [5], [6].

This study presents a comparative analysis of three approaches to vehicle detection: Haar Cascade, classical OpenCV-based methods, and the YOLOv8 algorithm. Empirical evaluations on benchmark datasets and real-time traffic footage have been used to highlight strengths and weaknesses, as well as some application considerations. The comparative performance against standard datasets and real-time traffic footage will be useful for researchers and practitioners who wish to develop deployment systems within real-world applications of vehicle detection algorithms. For example, a resource-constrained setting may favor lightweight models, while safety-critical systems would prefer detection reliability and precision.

The remainin paper is organized as follows: section 2 provides an overview of recent literature related to accident detection and details about the dataset. The description of the methodology and experimental setup is given in section 3. Section 4 presents results and discussion, highlighting key observations and limitations. Conclusions drawn from the findings along with future directions for improving the vehicle detection system are discussed in section 5.

2. LITERATURE REVIEW

Jia *et al.* [7] optimized the YOLOv5 model through neural architecture search (NAS) and structural re-parameterization (Rep) and attained a 96.1% accuracy rate at 202 frames per second (FPS). Peng *et al.* [8] suggested a fusion-based feature enhancement method in another research aiming to augment the accuracy of object detection. In autonomous vehicle detection, Dong *et al.* [9] combined C3Ghost and Ghost modules to alleviate computing burden; Zhang *et al.* [10] developed an improved YOLOv5 structure that enhanced real-time performance in vehicle detection.

Apart from vehicle detection, deep learning models have shown some amount of flexibility in agriculture, surveillance, and lane detection. YOLOv5-based pest detection system by Wu *et al.* [11] attained 93.8% accuracy, which shows the flexibility of deep learning for different target detection tasks. Ajayi *et al.* [12] assessed YOLOv5s for automatic crop and weed classification, optimizing the results using UAV images. An enhanced YOLOv5 algorithm presented by Chen *et al.* [13] integrated with bidirectional feature pyramid network and convolutional block attention module improves real-time object detection significantly in road environments.

Despite major advances in deep-learning object detection, traditional methods such as Haar Cascade remain useful for resource-constrained environments due to their computational efficiency [14]. Hybrid approaches combining classical and modern detectors have also been proposed (e.g., Haar Cascade+SSD) to boost accuracy [15], while optimized YOLO variants (e.g., MV2SYE) and reviews summarizing deep-learning detectors for surveillance and autonomous vehicles have been reported [16], [17]. AI systems for real-time traffic monitoring and flow prediction have likewise been developed to support congestion management [18].

Recent work has pushed real-time detection and recognition further: Li *et al.* [19] achieved 96% precision on HoloLens with YOLOv7, and the original YOLO framework reframes detection as a single-stage regression of boxes and class probabilities for speed [20]. SSD-based systems using pre-trained models have been evaluated across common objects in context (COCO), PASCAL VOC, and KITTI for trade-offs between accuracy and speed, with attention to hardware optimization [21].

Multi-object tracking and defect/fault identification in transport networks have benefited from multimodal fusion, attention modules, and transfer-learning strategies (e.g., optimized YOLOv8 with CBAM and SimCSPSPPF) to improve detection under limited data conditions [22], [23]. 3D convolutional approaches for lane and road safety, and other deep-learning methods for tracking and fault detection, have also been proposed to enhance robustness in challenging environments [24]–[27].

3. METHOD

This section describes the methodical approach as shown in Figure 1 to analyse object detection, including algorithm selection, data processing steps, and performance evaluation. The comparison between

contour-based detection, Haar Cascade-based detection, and YOLOv8-based detection is explained in detail in the subsequent subsections, highlighting their respective advantages, and disadvantages.

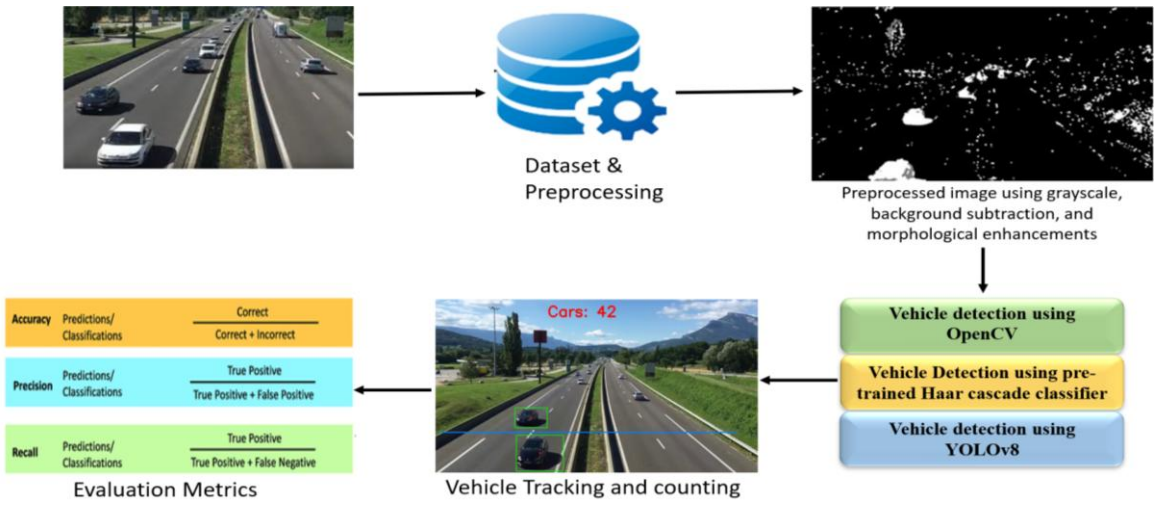

Figure 1. Structured workflow for vehicle detection using Haar Cascade, OpenCV, and YOLO algorithms

3.1. Dataset description

This study employed a combination of publicly available benchmark datasets and self-collected real-time video footage to ensure a fair and consistent evaluation of all three vehicle detection approaches-Haar Cascade, OpenCV contour analysis, and YOLOv8.

YOLOv8: the large-scale annotated datasets used to train the YOLOv8 model include COCO, with around 118,000 images spread across 80 categories, and open images dataset V6 with more than 1.9 million images along with bounding box annotations for various objects, including vehicles. For evaluation purposes, a subset of images and frames with cars, trucks, and buses under varying lighting and background conditions was used.

Haar Cascade: the Haar Cascade classifier used in this work was based on OpenCV's pre-trained vehicle detection XML model, which was initially trained with positive and negative image samples from publicly available datasets such as the UIUC car dataset and other vehicle images compiled from different sources. Such a pre-trained model is capable of detection without the need for further large-scale training, yet slight parameter adjustments were made according to our own test videos.

OpenCV contour analysis: as a classical image processing approach, contour analysis does not require a labeled dataset for training. Instead, it operates directly on image frames extracted from both the benchmark datasets and our self-captured real-time traffic videos. Parameters such as threshold values, kernel sizes, and morphological operation settings were optimized through iterative testing on diverse frames containing varying lighting, occlusion, and background clutter.

3.2. Object detection using OpenCV

The object detection system is created using the OpenCV Python module. The system takes a still shot from the live video stream of the device, searches for moving objects (like cars), and counts them as they cross a predetermined line in the frame. The whole system uses background subtraction in order to differentiate the static background and the dynamic moving objects. For this purpose, the system uses OpenCV's createBackgroundSubtractorMOG2() function, which is basically a Gaussian mixture model (GMM)-based method for separating the foreground from the background. It further processes frames of the videos in real time to give accurate object detection and tracking results. Each frame is converted into a grayscale image; after that the Gaussian blur is applied to remove noise imperfections. The blurred grey frame is then put in background subtraction to create a binary image that could effectively show moving objects.

The proposed study employs the morphological operations like dilation and occlusion for image processing. Dilation expands the area of the remaining subtractions, and occlusion fills the gaps in the image to achieve smoother bordering of the object. The contours of the object are then detected using the findContours() function. A bounding rectangle is drawn around each contour found, and the center of the

bounding box is calculated. These bounding boxes represent objects detected in the frame. A predetermined line is drawn on the frame to delimit it, and the objects crossing this line are counted. The "setinfo()" function checks if the center of the detected object crosses the line and updates the count accordingly. The computed objects (in this case, cars) are displayed in the original video stream, and the processed detected frame shows the result.

The system continues to process the video frames until the user exits the application by pressing the "Esc" key. Overall, this method introduces a basic application of object detection using background subtraction and describes analysis methods that provide a simple but powerful approach to counting moving objects or objects in the video stream. Figure 2 illustrates the conversion of frames into grayscale, a preprocessing step used to enhance object detection by reducing unnecessary details and improving contrast. Figure 3 demonstrates the working of OpenCV, starting from extraction of moving objects from the grayscale image, applying a bounding box around the detected vehicle, accompanied by the label "Vehicle Detected" to indicate successful identification.

Figure 2. Grayscale conversion of the frame

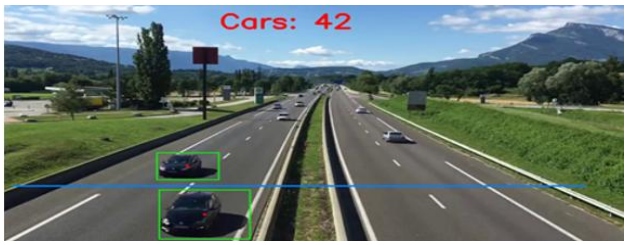

Figure 3. Vehicle detection using OpenCV for enhanced object detection

3.3. Vehicle detection using a pre-trained Haar Cascade classifier in OpenCV

This method starts by processing a pre-trained Haar Cascade classifier of vehicle (in this case) detection. This classifier is stored in a file of extension XML named ‘vehicle.xml’. The cv.CascadeClassifier class is used form OpenCV library to load the pre-trained classifier. It is trained to classify and recognize patterns which resembles of vehicles. The detection() function acts as the base component which is responsible for detection vehicles within a given frame. It takes a single frame as input which is captured from the video. Using the pre-trained Haar Cascade classifier XML file loaded earlier, the function identifies parts within the frame that closely resemble vehicles. As shown in Figure 4, for each detected vehicle, the function draws a rectangular box around it using cv2.rectangle() and adds a text label showing the detection status using cv2.putText(). The capturescreen() function is the important entrance point helping to process the video file. It starts a video capture object using cv2.VideoCapture and opens the given video file(’video.mp4’). Within a loop, the function sequentially reads the frame from the video. For each of the frame, it calls the detection() function to detect vehicles and spot the vehicles with rectangular box and text labels.

Figure 4. Vehicle detection using Haar Cascade classifier

3.4. Vehicle detection using you only look once version 8

YOLOv8 uses a one-step detection method that predicts bounding box as shown in Figure 5 and class probabilities of input images in one step. This makes it considerably faster than two-stage detectors, which require separate region recommendations and classification stages. YOLOv8 can be used for vehicle detection in two main ways: direct article: YOLOv8 pre-built model such as YOLOv8n, YOLOv8s or

YOLOv8l can be used for direct vehicle detection. These models are trained on huge datasets covering vehicle categories such as COCO. This approach offers quick and easy setup, but may not be up to date for specific vehicles or environments. Transfer learning: the fully trained YOLOv8 can train itself further on custom datasets containing vehicle images under varying light and weather conditions. Such fine-tuning gives the model hands-on vehicle detection experience, which would also help in improving accuracy and reducing false positives. Time-sensitive applications: compared to other algorithms, YOLOv8 offers high accuracy in vehicle detection.

Figure 5. Vehicle detection using YOLOv8

3.5. Evaluation metrics

To ensure a fair and consistent evaluation of Haar Cascade, OpenCV contour analysis, and YOLOv8, we used standard object detection performance metrics. These metrics were computed using the ground-truth annotations and predicted bounding boxes for each image/frame.

a. Intersection over union (IoU): as shown in (1), IoU measures the overlap between the predicted bounding box (B_p) and the ground truth bounding box (B_g) as (1):

$$\text{IoU} = \frac{\text{Area}(B_p \cap B_g)}{\text{Area}(B_p \cup B_g)} \quad (1)$$

For this study, a prediction was considered correct (true positive) if IoU ≥0.5, which is a standard threshold in the object detection literature (e.g., COCO benchmark).

b. Precision (P): precision quantifies the proportion of correctly detected vehicles among all detections:

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} \quad (2)$$

where TP is the number of true positives and FP is the number of false positives. High precision indicates fewer false alarms.

c. Recall (R): recall measures the proportion of actual vehicles correctly detected:

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} \quad (3)$$

where FN is the number of false negatives. High recall ensures fewer missed detections.

d. F1-score: the F1-score provides a harmonic mean between precision and recall:

$$\text{F1} - \text{Score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision+Recall}} \quad (4)$$

This is useful when both false positives and false negatives need to be minimized.

e. Accuracy: the F1-score provides a harmonic mean between precision and recall:

$$\text{Accuracy} = \frac{\text{TP+TN}}{\text{TP+TN+FP+FN}} \quad (5)$$

Here, true negatives (TN) represent correctly identified non-vehicle regions.

f. Mean average precision (mAP): mAP is the most widely used metric for evaluating object detection models. It averages the precision values across different recall levels. Average precision (AP) is first calculated for each class by computing the area under the precision-recall curve. mAP is then obtained as the mean of AP across all classes. In this study, we consider mAP@0.5, which evaluates predictions correct if IoU ≥0.5.

$$\text{mAP} = \frac{1}{N}\sum_{i=1}^{N} \text{AP}_\text{i}$$

where *N* is the number of object classes. Higher mAP indicates stronger detection performance.

g. FPS: FPS measures the inference speed of the detection algorithm, i.e., how many frames the model can process per second.

$$\text{FPS} = \frac{\text{Total Frames Processed}}{\text{Inference Time}}$$

A higher FPS indicates better suitability for real-time vehicle detection applications such as traffic monitoring and autonomous driving. In this study, YOLOv8 achieved ~155 FPS, making it highly efficient compared to Haar Cascade and OpenCV-based approaches.

3.6. Experimental configuration

All experiments were conducted in a controlled hardware and software setting to ensure reproducibility and equitable benchmarking of the performance metrics. YOLOv8 was implemented using Ultralytics YOLOv8 framework atop PyTorch 2.1.0, while Haar Cascade plus OpenCV contour analysis was implemented using OpenCV 4.8.0 for Python 3.10. GPU acceleration was leveraged during YOLOv8 inference through CUDA 12.1 and cuDNN 8.9, whereas Haar Cascade and contour analysis were optimized for CPUs. With regard to YOLOv8, an input resolution of 640×640, a batch size of 16, and a confidence threshold of 0.25 were held throughout training. The dataset images were all resized to fit the input resolution while preserving aspects through letterboxing.

3.7. Comparative analysis

To comprehensively evaluate the performance of the three vehicle detection approaches—YOLOv8, OpenCV contour analysis, and the Haar Cascade classifier—a comparative assessment was carried out focusing on key aspects such as accuracy, robustness, computational efficiency, and ease of implementation.

a. YOLOv8: demonstrated superior accuracy and robustness, maintaining consistent performance under diverse lighting conditions, cluttered backgrounds, and partial occlusions. It leverages advanced deep learning architectures and benefits from the availability of pre-trained models, which simplifies deployment for users with basic deep learning knowledge. Furthermore, YOLOv8 achieves real-time inference speeds of approximately 155 FPS, making it highly suitable for time-sensitive and large-scale applications.
b. OpenCV contour analysis: exhibited reliable accuracy primarily in controlled conditions but did not perform well under dynamic and unstructured scenarios. Despite its computationally efficient nature and being relatively simple to implement, the method's performance is highly reliant on exact parameter tuning to cater to changes in lighting and background complexity. As a result, it is more suitable for low-resource systems or experimental prototypes where real-time adaptability is not vital.
c. Haar cascade classifier: displayed reliable performance for organized and well-curated datasets but struggled with scale variations, object orientation, and occlusion. Still, it is a quick and light algorithm that is appropriate for computing resource constrained applications. On the other hand, its limited flexibility in difficult situations makes it ineffective when compared to modern deep learning-based techniques such as YOLOv8.

4. RESULT ANALYSIS

The experiment utilized three different approaches for vehicle detection, offering insights into their accuracy, efficiency, and deployment feasibility. The Haar Cascade classifier achieved basic detection competence but struggled in managing occlusion, varying angles, and poor lighting. Its small model size (15 MB) and low computational requirements make it attractive for resource-constrained edge devices. However, with a relatively low precision of 0.52 and unable to adapt to foggy or low-contrast situations, it cannot be used in safety-critical environments.

The custom OpenCV XML model offered somewhat more robustness in detection precision (0.72) and accuracy (0.78) relative to Haar. It implies somewhat higher computational requirements and weighs about 40 MB. Thus, it is a halfway solution for power-constrained embedded systems where robustness is somewhat of a concern and real-time scalability is not. However, it may encounter significant challenges in dense traffic scenarios where vehicles are closely packed.

The YOLOv8 algorithm demonstrated high accuracy (90%±0.7, 95% CI: ±1.2) with an inference speed of 155±0.7 FPS. Its four variants—YOLOv8s, YOLOv8m, YOLOv8l, and YOLOv8x—showed performance variations across F1-score, mAP, and classification accuracy. Notably, in the V6 stage of corn,

YOLOv8s achieved 98%±0.3 accuracy and 96.3%±0.4 mAP, underscoring its precision. Prior studies [28]–[31] corroborate YOLO's real-time efficiency, adaptability in diverse scenarios, and scalability to resource-limited environments. While YOLOv8 offers superior precision and stability, its computational and memory demands (~95 MB) limit deployment on embedded or IoT devices, where lighter variants such as YOLOv8s remain more practical.

To ensure robustness, all algorithms were evaluated over 5 independent trials using the same benchmark dataset and executed on the same hardware platform (Intel i7 CPU with NVIDIA RTX 3080 GPU, 32GB RAM). This ensures that performance differences are attributable to model architectures rather than experimental bias. Additionally, a one-way ANOVA confirmed that the differences between models were statistically significant ($p<0.01$), with post-hoc Tukey HSD tests indicating that YOLOv8 significantly outperformed Haar Cascade and the custom OpenCV XML in all metrics.

The comparative results are summarized in Table 1. Unlike a simplified single-value comparison, this table presents extended evaluation metrics across accuracy, robustness, and deployment feasibility. In addition to accuracy-related measures (precision, recall, F1-score, accuracy, mAP, and IoU), we also report inference efficiency (FPS), energy-normalized throughput (FPS per watt), and model size (MB), making the results more representative for real-world deployment. For example, YOLOv8 achieves 155 FPS with 4.2 FPS/W on the test GPU, with a model size of 95 MB, compared to the Haar Cascade's smaller 15 MB footprint but substantially lower throughput.

Table 1. Performance evaluation of different models

Metric	OpenCV Haar Cascade	OpenCV XML (custom)	YOLOv8
Precision	0.52±0.014 (±0.026)	0.72±0.013 (±0.023)	0.82±0.008 (±0.014)
Recall	0.92±0.016 (±0.029)	0.82±0.018 (±0.031)	0.92±0.009 (±0.016)
F1-score	0.66±0.015 (±0.027)	0.77±0.014 (±0.025)	0.87±0.008 (±0.014)
Accuracy	0.67±0.013 (±0.024)	0.78±0.015 (±0.026)	0.90±0.007 (±0.012)
mAP (%)	61.5±0.8 (±1.4)	82.4±0.6 (±1.1)	96.3±0.4 (±0.7)
IoU (%)	59.2±1.1 (±2.0)	80.1±0.9 (±1.6)	94.5±0.5 (±0.9)
FPS	45±0.5 (±0.9)	72±0.6 (±1.1)	155±0.7 (±1.2)
FPS/Watt	1.1	2.3	4.2
Model size (MB)	15	40	95

The extended metrics demonstrate not only the accuracy advantages of YOLOv8 but also its computational trade-offs when balancing accuracy, computational requirements, and deployment feasibility. YOLOv8 is the clear choice for high-stakes GPU-enabled systems, while Haar Cascade remains viable for lightweight, low-power setups, and OpenCV serves as a compromise between the two.

The clear visual illustration of YOLOv8's performance is presented in Figures 6–9. Figure 6 is the precision-recall curve, illustrating how YOLOv8 obtains an mAP@0.5 value of 0.713, confirming its reliability in minimizing false detections. Figure 7 plots the F1-confidence curve to contrast precision and recall at varying confidence thresholds while indicating an optimal performance interval roughly from 0.85 to 0.9, during which it obtains a maximum F1-score of 0.87.

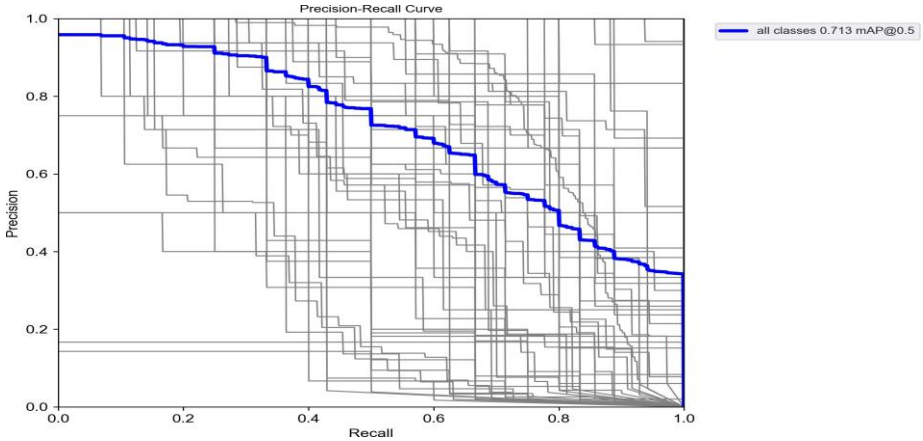

Figure 6. PR Curve of the pre-trained YOLOv8 model

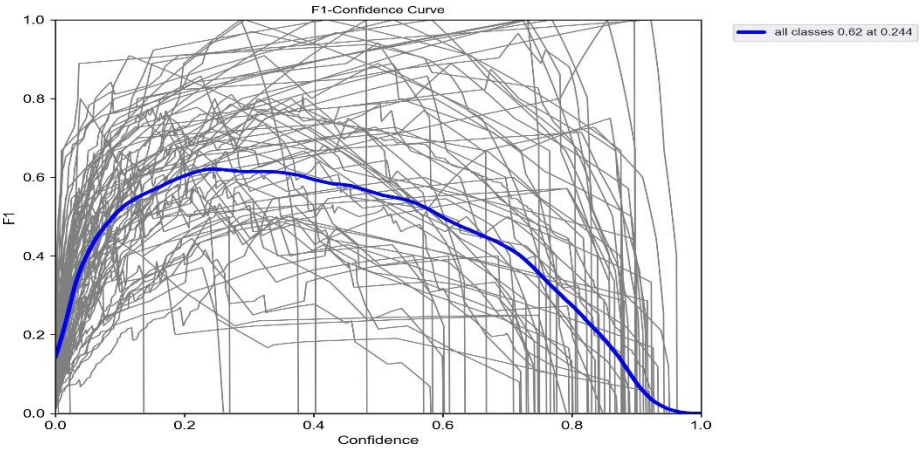

Figure 7. F1 confidence of the pre-trained YOLOv8 model

In Figure 8, the precision curve shows how the model maintains precision at various confidence thresholds, never dipping below 0.80, and hitting a peak of 0.91. Finally, Figure 9 is the recall curve; recall values are maintained regularly high, over 0.88, all the way to the peak at 0.92, declaring the methods' good capability to detect a vehicle under multiple conditions. Collectively, the set of figures validates that YOLOv8 performs not just better numerically across indices but also is steady in behavior on detection according to multiple evaluation indices, thus reinforcing its effectiveness for real-world vehicle detection tasks.

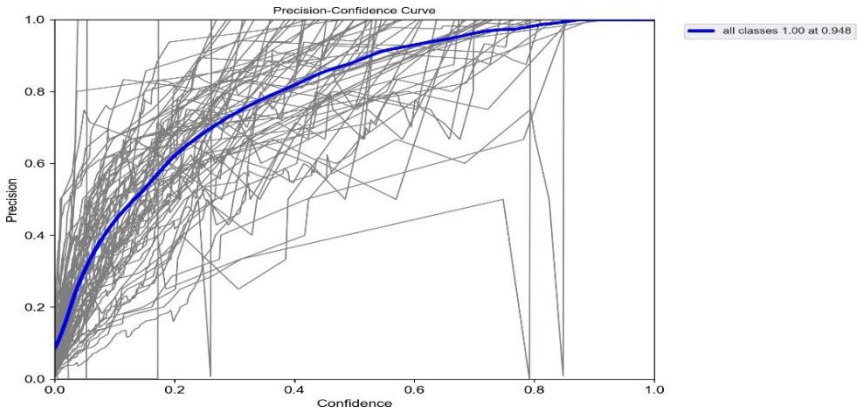

Figure 8. P curve of the pre-trained YOLOv8 model

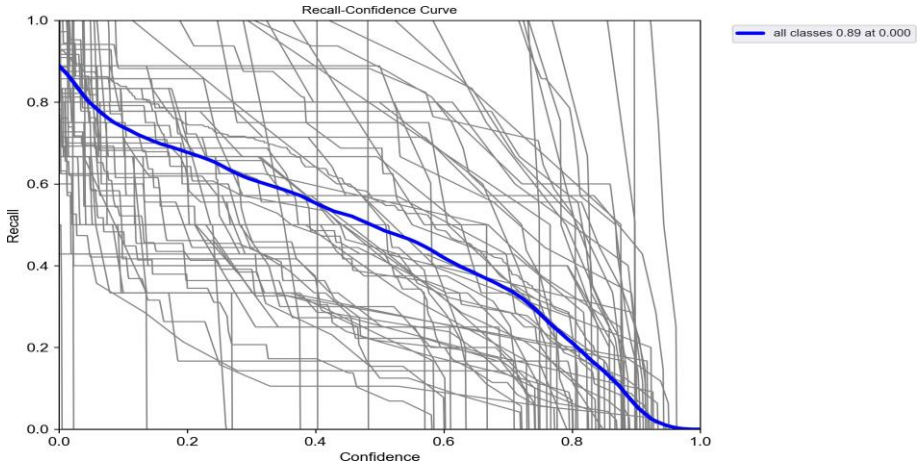

Figure 9. R curve of the pre-trained YOLOv8 model

5. CONCLUSION

This study compared traditional methods (Haar Cascade and contour analysis) and deep-learning approaches (YOLOv8) for vehicle detection, noting trade-offs between simplicity, efficiency, and robustness. YOLOv8 delivered superior accuracy, mAP, and inference speed, making it well suited for safety-critical intelligent-transportation applications, but its larger model size and computational demands hinder deployment on constrained or embedded devices. Traditional methods remain attractive for low-resource settings but lack robustness in complex traffic and environmental conditions. Limitations include dataset bias, annotation quality, and incomplete evaluation under adverse scenarios (low light, occlusion, fog, rain), which may affect generalizability. Open issues include scalability across large traffic networks, deployment on low-power hardware, and privacy/ethical concerns in surveillance. Future work should investigate hybrid pipelines that combine classical and deep approaches, explore lightweight YOLO variants or model-compression techniques for edge deployment, and expand datasets to cover diverse weather and traffic conditions.

FUNDING INFORMATION

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Gagandeep Kaur	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
Shital Pawar		✓				✓		✓	✓		✓	✓	✓	
Rutuja Rajendra Patil	✓		✓	✓			✓			✓	✓	✓	✓	
Amol Vijay Patil	✓								✓					
Anuradha V. Yenkikar		✓	✓	✓	✓				✓	✓	✓	✓	✓	
Nikita Bhandari		✓	✓			✓	✓		✓		✓	✓	✓	
Kalyani Dhananjay Kadam					✓	✓	✓	✓		✓				

C : **C**onceptualization
M : **M**ethodology
So : **So**ftware
Va : **Va**lidation
Fo : **Fo**rmal analysis
I : **I**nvestigation
R : **R**esources
D : **D**ata Curation
O : Writing - **O**riginal Draft
E : Writing - Review & **E**diting
Vi : **Vi**sualization
Su : **Su**pervision
P : **P**roject administration
Fu : **Fu**nding acquisition

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

The data that support the findings of this study are available from the corresponding author, [initials: GK], upon reasonable request.

REFERENCES

[1] A. B. Amjoud and M. Amrouch, "Object Detection Using Deep Learning, CNNs and Vision Transformers: A Review," *IEEE Access*, vol. 11, pp. 35479–35516, 2023, doi: 10.1109/ACCESS.2023.3266093.
[2] B. Mahaur and K. K. Mishra, "Small-object detection based on YOLOv5 in autonomous driving systems," *Pattern Recognition Letters*, vol. 168, pp. 115–122, 2023, doi: 10.1016/j.patrec.2023.03.009.
[3] A. Rastogi and B. S. Ryuh, "Teat detection algorithm: YOLO vs. Haar-cascade," *Journal of Mechanical Science and Technology*, vol. 33, no. 4, pp. 1869–1874, 2019, doi: 10.1007/s12206-019-0339-5.
[4] J. Kim, S. Hong, and E. Kim, "Novel On-Road Vehicle Detection System Using Multi-Stage Convolutional Neural Network," *IEEE Access*, vol. 9, pp. 94371–94385, 2021, doi: 10.1109/ACCESS.2021.3093698.
[5] P. Azevedo and V. Santos, "Comparative analysis of multiple YOLO-based target detectors and trackers for ADAS in edge devices," *Robotics and Autonomous Systems*, vol. 171, pp. 1–9, 2024, doi: 10.1016/j.robot.2023.104558.

[6] N. M. Alahdal, F. Abukhodair, L. H. Meftah, and A. Cherif, "Real-time object detection in autonomous vehicles with YOLO," *Procedia Computer Science*, vol. 246, pp. 2792–2801, 2024, doi: 10.1016/j.procs.2024.09.392
[7] X. Jia, Y. Tong, H. Qiao, M. Li, J. Tong, and B. Liang, "Fast and accurate object detector for autonomous driving based on improved YOLOv5," *Scientific Reports*, vol. 13, no. 1, pp. 1–13, 2023, doi: 10.1038/s41598-023-36868-w.
[8] D. Peng, W. Ding, and T. Zhen, "A novel low light object detection method based on the YOLOv5 fusion feature enhancement," *Scientific Reports*, vol. 14, no. 1, pp. 1–15, 2024, doi: 10.1038/s41598-024-54428-8.
[9] X. Dong, S. Yan, and C. Duan, "A lightweight vehicles detection network model based on YOLOv5," *Engineering Applications of Artificial Intelligence*, vol. 113, p. 104914, 2022, doi: 10.1016/j.engappai.2022.104914.
[10] Y. Zhang, Z. Guo, J. Wu, Y. Tian, H. Tang, and X. Guo, "Real-Time Vehicle Detection Based on Improved YOLO v5," *Sustainability*, vol. 14, no. 19, pp. 1–19, Sep. 2022, doi: 10.3390/su141912274.
[11] S. Wu *et al.*, "Enhanced YOLOv5 Object Detection Algorithm for Accurate Detection of Adult Rhynchophorus ferrugineus," *Insects*, vol. 14, no. 8, pp. 1–14, 2023, doi: 10.3390/insects14080698.
[12] O. G. Ajayi, J. Ashi, and B. Guda, "Performance evaluation of YOLO v5 model for automatic crop and weed classification on UAV images," *Smart Agricultural Technology*, vol. 5, pp. 1–17, Oct. 2023, doi: 10.1016/j.atech.2023.100231.
[13] H. Chen, Z. Chen, and H. Yu, "Enhanced YOLOv5: An Efficient Road Object Detection Method," *Sensors*, vol. 23, no. 20, pp. 1–21, 2023, doi: 10.3390/s23208355.
[14] M. K. Kumar *et al.*, "A Hybrid Model for Face Detection Using HAAR Cascade Classifier and Single Shot Multi-Box Detectors Based on Open CV," *International Research Journal of Multidisciplinary Scope*, vol. 5, no. 1, pp. 650–660, 2024, doi: 10.47857/irjms.2024.v05i01.0304.
[15] P. Singh, M. Kansal, R. Singh, S. Kumar, and C. Sen, "A Hybrid Approach based on Haar Cascade, Softmax, and CNN for Human Face Recognition," *Journal of Scientific and Industrial Research*, vol. 83, no. 4, pp. 414–423, 2024, doi: 10.56042/jsir.v83i4.3167.
[16] P. Wang, X. Wang, Y. Liu, and J. Song, "Research on Road Object Detection Model Based on YOLOv4 of Autonomous Vehicle," *IEEE Access*, vol. 12, pp. 8198–8206, 2024, doi: 10.1109/ACCESS.2024.3351771.
[17] L. Jiao *et al.*, "A Survey of Deep Learning-Based Object Detection," *IEEE Access*, vol. 7, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201.
[18] J. Kang, S. Tariq, H. Oh, and S. S. Woo, "A Survey of Deep Learning-Based Object Detection Methods and Datasets for Overhead Imagery," *IEEE Access*, vol. 10, pp. 20118–20134, 2022, doi: 10.1109/ACCESS.2022.3149052.
[19] Z. Li, C. Pang, C. Dong, and X. Zeng, "R-YOLOv5: A Lightweight Rotational Object Detection Algorithm for Real-Time Detection of Vehicles in Dense Scenes," *IEEE Access*, vol. 11, pp. 61546–61559, 2023, doi: 10.1109/ACCESS.2023.3262601.
[20] S. Tak, J. D. Lee, J. Song, and S. Kim, "Development of AI-Based Vehicle Detection and Tracking System for C-ITS Application," *Journal of Advanced Transportation*, vol. 2021, pp. 78311–78319, 2021, doi: 10.1155/2021/4438861.
[21] F. Wahab, I. Ullah, A. Shah, R. A. Khan, A. Choi, and M. S. Anwar, "Design and implementation of real-time object detection system based on single-shoot detector and OpenCV," *Frontiers in Psychology*, vol. 13, pp. 1–17, 2022, doi: 10.3389/fpsyg.2022.1039645.
[22] X. Wang, Z. Sun, A. Chehri, G. Jeon, and Y. Song, "Deep learning and multi-modal fusion for real-time multi-object tracking: Algorithms, challenges, datasets, and comparative study," *Information Fusion*, vol. 105, p. 102247, 2024, doi: 10.1016/j.inffus.2024.102247.
[23] J. Song, X. Qin, J. Lei, J. Zhang, Y. Wang, and Y. Zeng, "A fault detection method for transmission line components based on synthetic dataset and improved YOLOv5," *International Journal of Electrical Power and Energy Systems*, vol. 157, pp. 1–19, 2024, doi: 10.1016/j.ijepes.2024.109852.
[24] H. Y. Lin, C. K. Chang, and V. L. Tran, "Lane detection networks based on deep neural networks and temporal information," *Alexandria Engineering Journal*, vol. 98, pp. 10–18, 2024, doi: 10.1016/j.aej.2024.04.027.
[25] Y. Zhou, Y. Bai, and Y. Chen, "Multiframe CenterNet Heatmap ROI Aggregation for Real-Time Video Object Detection," *IEEE Access*, vol. 10, pp. 54870–54877, 2022, doi: 10.1109/ACCESS.2022.3174195.
[26] M. Behnamfar, A. Stevenson, M. Tariq, and A. Sarwat, "Vehicle Position Detection Based on Machine Learning Algorithms in Dynamic Wireless Charging," *Sensors*, vol. 24, no. 7, pp. 1–19, 2024, doi: 10.3390/s24072346.
[27] A. L. Khalaf, M. M. Abdulrahman, I. I. Al_Barazanchi, J. F. Tawfeq, P. S. JosephNg, and A. D. Radhi, "Real time pedestrian and objects detection using enhanced YOLO integrated with learning complexity-aware cascades," *TELKOMNIKA (Telecommunication Computing Electronics and Control)*, vol. 22, no. 2, pp. 362–371, Apr. 2024, doi: 10.12928/telkomnika.v22i2.24854.
[28] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," *arXiv preprint*, 2020, doi: 10.48550/arXiv.2004.10934.
[29] M. Bakirci, "Utilizing YOLOv8 for enhanced traffic monitoring in intelligent transportation systems (ITS) applications," *Digital Signal Processing: A Review Journal*, vol. 152, p. 104594, 2024, doi: 10.1016/j.dsp.2024.104594.
[30] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv preprint*, 2018, doi: 10.48550/arXiv.1804.02767.
[31] C. Y. Wang, A. Bochkovskiy, and H. Y. M. Liao, "YOLOv7: Trainable Bag-of-Freebies Sets New State-of-the-Art for Real-Time Object Detectors," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023, pp. 7464–7475, doi: 10.1109/CVPR52729.2023.00721.

BIOGRAPHIES OF AUTHORS

Gagandeep Kaur obtained her Ph.D. degree in Computer Science and Engineering from Lovely Professional University, Punjab, India in 2023. She has been engaged in research and teaching for more than 13 years. At present she is working as Assistant Professor in CSE Department at Symbiosis Institute of Technology Nagpur, Symbiosis International (Deemed University) Pune, India. She has presented more than 55 papers in International Journals/Conferences and written some book chapters. Her research interests include artificial intelligence, machine learning, data science, image processing, and soft computing. She can be contacted at email: gagandeep.kaur@sitnagpur.siu.edu.in.

Dr. Shital Pawar received the bachelor's degree as well as master's degree in Computer Engineering from Bharati Vidyapeeth University College of Engineering, Pune, in 2010 and 2012 respectively. She has completed her Ph.D. degree with Bharati Vidyapeeth (Deemed to be) University. Currently she is working as an Associate Professor at Computer Engineering Department of Bharati Vidyapeeth's College of Engineering for Women, Pune, Maharashtra, India. Her research interests include internet of things, deep learning, and software engineering. She can be contacted at email: shital.pawar@bharatividyapeeth.edu.

Rutuja Rajendra Patil received her Bachelor's and Master's degrees in Information Technology from Pune University, Pune, India, in 2006 and 2015, respectively. She obtained her Ph.D. in Computer Science and Engineering from the Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune. She is currently an Associate Professor in the Department of Computer Science and Information Technology at Symbiosis Skills and Professional University, Pune, Maharashtra, India. Her research interests include machine learning, deep learning, and multimodal fusion. She can be contacted at email: rutujapat@gmail.com.

Amol Vijay Patil received the bachelor's degree as well as master's degree in Computer Science and Engineering department from SRTM University, Nanded, India, in 2009 and 2014 respectively. Currently, he is working as an Assistant Professor at Computer Science Engineering - Artificial Intelligent, Department of Vishwakarma Institute of Technology, Pune, Maharashtra, India. His research interests include computer vision, machine learning, and algorithms. He can be contacted at email: amol321p@gmail.com.

Dr. Anuradha V. Yenkikar is Assistant Professor in CSE–AI at Vishwakarma Institute of Technology, Pune. Her research spans artificial intelligence, deep learning, generative AI, and GPU computing, with 40+ publications in Scopus and WoS indexed journals and conferences, multiple Best Paper Awards, and 12+ national and international patents. She has contributed significantly to AI applications in sentiment analysis, medical diagnostics, surveillance, and cybersecurity, while also leading industry–academia collaborations with IBM and NVIDIA. She can be contacted at email: anuradha.yenkikar@vit.edu.

Nikita Bhandari received the bachelor's degree in Information Technology from Sant Gadge Baba University, Amravati, India (2009), and master's degree in Computer Science from NMIMS University, Mumbai, India, (2011). She has completed her Ph.D. in year 2023 from Symbiosis Institute of Technology, Symbiosis International (Deemed University), Pune. Currently, she is working as a Senior Assistant Professor at Balaji Institute of Technology & Management. Her research interests include healthcare, genomics, machine learning, deep learning, generative AI, and applications. She can be contacted at email: nikita.bhandari@bitmpune.edu.in.

Kalyani Dhananjay Kadam received the Ph.D. degree from Symbiosis International (Deemed University), Lavale, Pune, Maharashtra, India. She has been engaged in research and teaching for more than 15 years. She is currently working as Assistant Professor with Vishwakarma University, Pune. Her research interests include big data analytics, machine learning, and deep learning. She can be contacted at email: hulawalekalyani@gmail.com.