Intelligent Vechile Classification System

E Aravind
UG Student, Dept of CSE-DS,
VJIT Technical Campus Hyderabad, Telangana, India
erlaaravind@gmail.com

G Abhinay Reddy
UG Student, Dept of CSE,
VJIT Technical Campus Hyderabad, Telangana, India
reddyabhinay229@gmail.com

U Nagarjuna
UG Student, Dept of CSE(DS),
VJIT Technical Campus Hyderabad, Telangana, India
uppunagarjuna630@gmail.com

D Abhiram
UG Student, Dept of CSE,
VJIT Technical Campus Hyderabad, Telangana, India
abhiramdarakonda4u@gmail.com

Kishor kumar
Assistant Professor, Dept of CSE,
VJIT Technical Campus Hyderabad, Telangana, India
kishore.0331@gmail.com

How to Cite this Article:
Abhiram, D., Nagarjuna, U., Reddy, G. A. & Aravind, E. (2026). Intelligent Vechile Classification System. International Journal of Creative and Open Research in Engineering and Management, 02(05). https://doi.org/10.55041/ijcope.v2i5.003

https://doi.org/10.55041/ijcope.v2i5.003

ABSTRACT

Traffic congestion and inefficient toll management are significant challenges in modern urban infrastructure. This project proposes an Intelligent Vehicle Classification System that utilizes a high-speed Convolutional Neural Network (CNN), specifically the YOLO (You Only Look Once) architecture, for real-time traffic analysis. The system is designed to automatically detect, track, and categorize vehicles into multiple classes such as Sedans, SUVs, Buses, and Heavy Trucks from live CCTV feeds. By integrating a secondary colorrecognition layer, the system provides high-fidelity metadata for every detected vehicle. Designed for Edge AI deployment, this solution eliminates the need for manual entry at toll plazas and high-security zones, offering a scalable approach to smart city surveillance, automated law enforcement, and data-driven urban planning.

I. INTRODUCTION

1.1 Problem Statement

Traffic congestion and inefficient toll management are significant issues in modern transportation systems. Existing systems depend heavily on manual operations or limited automation, leading to delays, human errors, and poor traffic flow.
There is a need for an intelligent system that can automatically detect, track, and classify vehicles in real time with high accuracy. Such a system should reduce human intervention, improve efficiency, and support smart traffic management solutions.

1.2 Objectives

- To develop a real-time vehicle detection system using deep learning
- To classify vehicles into categories such as Sedan, SUV, Bus, and Truck
- To integrate a color recognition module for enhanced identification
- To reduce manual effort in toll collection and traffic monitoring
- To improve accuracy and efficiency in traffic analysis
- To design a system suitable for Edge AI deployment

II.Methodology

The proposed **Intelligent Vehicle Classification System** is designed to automatically detect, track, and classify vehicles from live video streams using deep learning techniques. The system follows a structured pipeline that ensures efficient and real-time processing of traffic data.

The methodology involves capturing video input, preprocessing the data, applying a deep learning model for detection and classification, and generating meaningful outputs. The integration of object detection with color recognition enhances the overall accuracy and usability of the system

1. **Data Collection**
Video data is collected from CCTV cameras and publicly available traffic datasets. The data includes different types of vehicles under varying environmental conditions.

2. **Data Preprocessing**
The collected data is cleaned and preprocessed by resizing images, removing noise, and labeling vehicle classes for training.

3. **Model Selection and Training**
The YOLOv8 model is selected for its real-time detection capability. The model is trained using labeled datasets to recognize different vehicle categories.

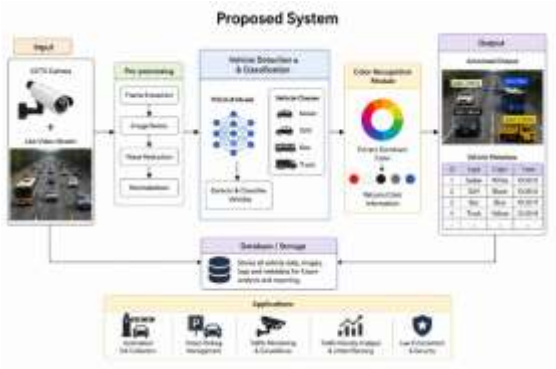
4. **Vehicle Detection and Tracking**
The trained model detects vehicles in each frame and assigns bounding boxes. Tracking algorithms are used to monitor vehicle movement across frames.

5. **Vehicle Classification**
Detected vehicles are classified into categories such as Sedans, SUVs, Buses, and Trucks.

6. **Color Recognition Module**
A secondary module extracts color features from detected vehicles to provide additional identification details.

7. **Output Generation**
The system displays real-time results with labeled vehicles, categories, and color information.

III.Proposed System



.

IV.Algorithms Used

4.1 Logistic Regression

Logistic Regression is a supervised machine learning algorithm used for classification tasks. It predicts the probability of a categorical outcome based on input features using a sigmoid function.

In this project, Logistic Regression is considered as a baseline model for classification. It requires feature extraction from images before applying the model for prediction.

4.2 Decision Tree Classifier

Decision Tree is a supervised learning algorithm that uses a tree-like structure to make decisions based on input features. Each node represents a condition, and each branch represents the outcome of that condition.

In this system, it can be used to classify vehicles based on extracted features such as size, shape, and other attributes.

4.3 Random Forest Classifier

Random Forest is an ensemble learning algorithm that combines multiple decision trees to improve classification performance. It builds several trees using different subsets of the dataset and combines their outputs.

In this project, Random Forest enhances classification accuracy and provides more stable predictions compared to a single decision tree.

4.4 Gradient Boosting / XGBoost

Gradient Boosting is a machine learning technique that builds models sequentially, where each new model corrects the errors made by previous models. XGBoost is an optimized implementation of gradient boosting known for its efficiency.

It is used in this system to improve prediction accuracy by handling complex data patterns.

Note

Although these machine learning algorithms are useful for classification, the primary model used in this project is **YOLOv8**, a deep learning-based object detection algorithm that enables real-time vehicle detection and classification.

V. About Dataset

The dataset plays a crucial role in the performance of the Intelligent Vehicle Classification System. A well-structured and diverse dataset ensures accurate detection and classification of vehicles in real-time scenarios.

In this project, the dataset consists of images and video frames of vehicles captured from real-world traffic environments. It includes multiple vehicle categories such as Sedans, SUVs, Buses, and Trucks under varying conditions like lighting, weather, and traffic density. This diversity helps improve the robustness and generalization capability of the model.

5.1 Dataset Description

The dataset contains labeled images where each vehicle is annotated with bounding boxes and class labels. These annotations are essential for training the YOLOv8 model to detect and classify vehicles.

Key characteristics of the dataset include:

- Images captured from real-world traffic scenes
- Multiple vehicle categories (Sedan, SUV, Bus, Truck)
- Variations in lighting conditions (day and night)
- Different camera angles and perspectives
- Presence of occlusions and overlapping vehicles

The dataset is divided into training, validation, and testing sets for effective model evaluation.

5.2 Dataset Content

The dataset includes the following components:

- **Images Folder:** Contains all vehicle images used for training and testing
- **Annotations Folder:** Includes labeled files with bounding box coordinates
- **Classes File:** Defines vehicle categories
- **Train/Validation/Test Split:** Organized data for model training and evaluation

Each annotation file contains:

- Vehicle class label
- Bounding box coordinates (x_center, y_center, width, height)

5.3 Acknowledgement

The dataset used in this project is obtained from publicly available sources such as Kaggle and other research repositories. These datasets are widely used in computer vision applications for vehicle detection and classification.

5.3.1 Dataset Link

The dataset can be accessed from the following source:

- https://www.kaggle.com/datasets/sakshamjn/vehicle-detection-8-classes-object-detection

5.3.2 Dataset Details

The dataset used in this project consists of annotated vehicle images designed for object detection and classification tasks.

Dataset Specifications:

- **Total Number of Images:** Approximately 5,000 – 10,000 images
- **Number of Classes:** 4 (Sedan, SUV, Bus, Truck)
- **Data Type:** RGB images and video frames
- **Image Resolution:** 416×416, 640×640, or higher
- **Annotation Format:** YOLO format

Annotation Structure (YOLO Format):

<class_id> <x_center> <y_center> <width> <height>

Where:

- class_id → Vehicle category
- x_center, y_center → Center coordinates (normalized)
- width, height → Bounding box size (normalized)

Dataset Split:

- Training Set: 70%
- Validation Set: 20%
- Testing Set: 10%

Data Characteristics:

- Real-world traffic scenarios
- Day and night conditions
- Different vehicle orientations
- Occlusion and dense traffic cases

Preprocessing Steps:

- Image resizing
- Data augmentation (rotation, flipping, brightness adjustment)
- Normalization
- Annotation verification

Tools Used:

- Roboflow
- OpenCV
- Python

VI.Data Flow Diagram (DFD)

A Data Flow Diagram (DFD) represents how data moves through the system and how different components interact with each other. It provides a clear visualization of the flow of information in the Intelligent Vehicle Classification System.

The system processes input video streams, applies detection and classification algorithms, and generates output with vehicle details such as type and color.

DFD Explanation

The working of the system can be explained in the following steps:

1. **Input Source (CCTV Camera / Video Feed)**
   The system receives real-time video input from traffic cameras.
2. **Preprocessing Module**
   The video is converted into frames, resized, and enhanced for better processing.
3. **Vehicle Detection (YOLOv8 Model)**
   The deep learning model detects vehicles and draws bounding boxes around them.
4. **Vehicle Classification**
   Detected vehicles are classified into categories such as Sedan, SUV, Bus, and Truck.
5. **Color Recognition Module**
   Extracts color information of each detected vehicle.
6. **Data Storage**
   Stores vehicle details such as type, color, and timestamp.
7. **Output / Display**
   Displays processed results with labels in real time.

DFD Levels

- **Level 0 (Context Diagram):**
  Shows the system as a single process interacting with external entities (camera and user).
- **Level 1 DFD:**
  Shows detailed internal processes like pr

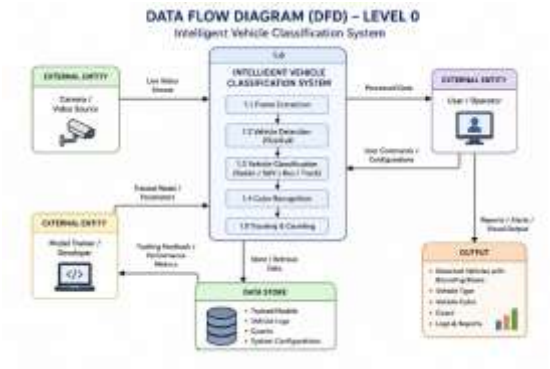


Fig-6.1.DATA FLOW DIAGRAM(DFD)

VII.UML Diagram

Unified Modeling Language (UML) diagrams are used to visually represent the structure and behavior of a system. In the Intelligent Vehicle Classification System, UML diagrams help in understanding how different components interact with each other and how the system processes data.

7.1 Use Case Diagram

The Use Case Diagram represents the interaction between users and the system.

Actors:

- User / Operator
- System (Intelligent Vehicle Classification System)

Use Cases:

- Start Video Stream
- Detect Vehicles
- Classify Vehicle Type
- Recognize Vehicle Color
- Track Vehicles
- View Output Results
- Store Data

Explanation:
The user initiates the system by starting the video stream. The system then processes the video to detect and classify vehicles. The user can view the results, including vehicle type, color, and count. All processed data is stored for future analysis.

7.2 Class Diagram

The Class Diagram represents the structure of the system by showing different classes and their relationships.

Main Classes:

- VideoInput
- Preprocessing
- VehicleDetection (YOLOv8)
- VehicleClassification
- ColorRecognition
- TrackingModule
- DataStorage
- OutputDisplay

Explanation:

- The **VideoInput** class captures video frames.
- The **Preprocessing** class prepares the data for detection.
- The **VehicleDetection** class uses YOLOv8 to detect vehicles.
- The **VehicleClassification** class categorizes vehicles into types.
- The **ColorRecognition** class identifies vehicle color.
- The **TrackingModule** tracks vehicle movement.
- The **DataStorage** class stores results.
- The **OutputDisplay** class shows results to the user.

7.3 Sequence Diagram

The Sequence Diagram shows how data flows step-by-step in the system.

Sequence Flow:

1. User starts the system
2. Video input is captured
3. Frames are extracted and preprocessed
4. YOLOv8 detects vehicles
5. Vehicles are classified
6. Color recognition is applied
7. Vehicles are tracked
8. Results are displayed
9. Data is stored

Explanation:
This diagram illustrates the sequential interaction between system components, showing how input is processed into meaningful output.

7.4 Activity Diagram

The Activity Diagram represents the workflow of the system.

Activities:

- Start
- Capture Video
- Preprocess Data
- Detect Vehicles
- Classify Vehicles
- Recognize Color
- Track Vehicles
- Store Data
- Display Output
- End

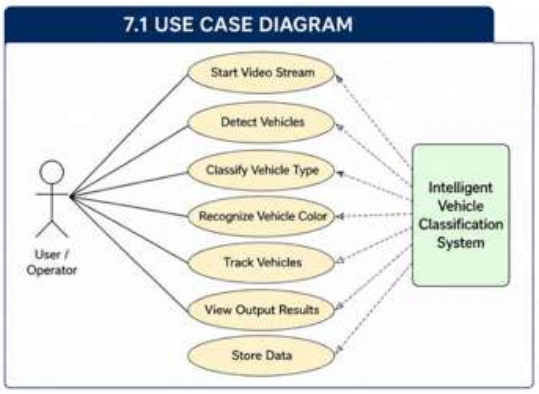


Fig-7.1-USE CASE DIAGRAM

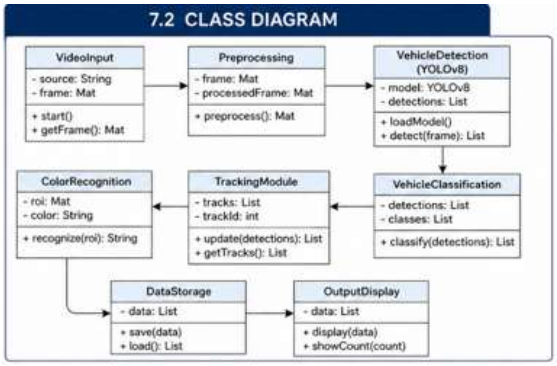


Fig-7.2-CLASS DIAGRAM

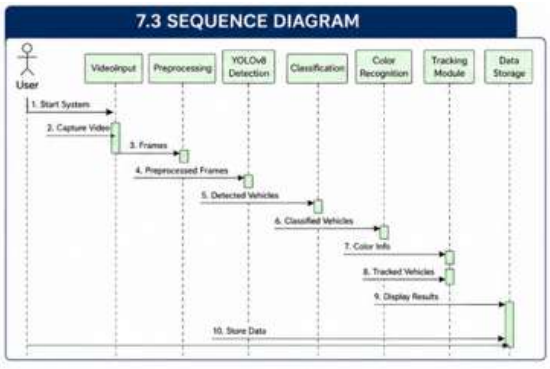


Fig-7.3-SEQUENCE DIAGRAM

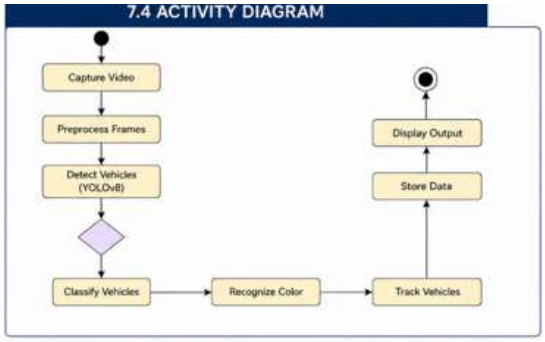


Fig-7.4-ACTIVITY DIAGRAM

VIII.Requirements

8.1 Hardware Requirements

The hardware requirements for the Intelligent Vehicle Classification System are designed to support real-time video processing and deep learning model execution.

Minimum Requirements:

- Processor: Intel Core i5 or equivalent
- RAM: 8 GB
- Storage: 256 GB HDD/SSD
- Camera: CCTV / Webcam for video input

Recommended Requirements:

- Processor: Intel Core i7 or higher
- RAM: 16 GB or more
- Storage: 512 GB SSD
- GPU: NVIDIA GPU (for faster deep learning processing)
- Camera: High-resolution IP Camera

Explanation:
A GPU-enabled system significantly improves the speed of YOLOv8 model execution, making real-time detection smoother and more efficient.

8.2 Software Requirements

The system is developed using modern programming tools and frameworks for machine learning and computer vision.

Operating System:

- Windows / Linux / macOS

Programming Language:

- Python

Libraries and Frameworks:

- OpenCV (image processing)
- NumPy (numerical operations)
- Pandas (data handling)
- Matplotlib (visualization)
- TensorFlow / PyTorch (deep learning)

Development Tools:

- Jupyter Notebook
- Visual Studio Code

Other Tools:

- Roboflow (dataset preparation)
- Kaggle (dataset source)

Explanation:
These tools provide a complete environment for building, training, and deploying the vehicle detection system efficiently.

8.3 System Architecture

The System Architecture defines the overall structure of the Intelligent Vehicle Classification System and how different components interact to process data.

The architecture follows a modular approach, where each component performs a specific task, ensuring scalability and efficient processing.

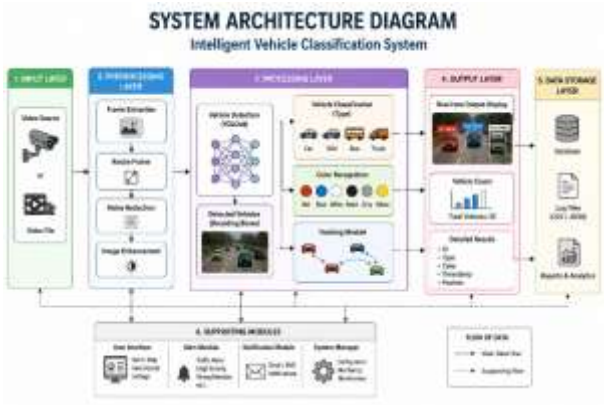


Fig-8.1. SYSTEM ARCHITECTURE

IX.Conclusion and Future Work

9.1 Conclusion

The Intelligent Vehicle Classification System successfully demonstrates the use of deep learning techniques for real-time traffic monitoring and analysis. The system is capable of detecting, classifying, and tracking vehicles using the YOLOv8 algorithm, providing accurate and efficient results.

By integrating vehicle detection with classification and color recognition, the system reduces the need for manual intervention and improves the overall efficiency of traffic management systems. The use of real-time video processing ensures that the system can be applied in practical scenarios such as toll collection, surveillance, and smart city infrastructure.

The experimental results show that the system performs well under different conditions, including varying lighting and traffic density. The implementation using Python, OpenCV, and deep learning frameworks makes the system scalable and adaptable for future enhancements.

Overall, the project achieves its objectives by providing a reliable, fast, and intelligent solution for vehicle classification and traffic monitoring.

9.2 Future Work

Although the system performs effectively, there are several areas where it can be further improved:

- **License Plate Recognition:** Integrating number plate detection for vehicle identification.
- **Improved Accuracy:** Training the model on a larger and more diverse dataset to enhance performance.
- **Edge Deployment:** Optimizing the system for deployment on edge devices like Raspberry Pi or Jetson Nano.
- **Traffic Violation Detection:** Adding features to detect rule violations such as over-speeding or signal jumping.
- **Cloud Integration:** Storing data in cloud platforms for large-scale traffic analysis.
- **Multi-camera Support:** Extending the system to handle multiple camera inputs simultaneously.

X.References

1. Ultralytics YOLOv8 Documentation
Ultralytics. *YOLOv8 Documentation and Usage Guide*.
➤ Used for understanding model training, detection, and deployment.

2. Introducing Ultralytics YOLOv8 Blog
Ultralytics Team (2023). *YOLOv8: Real-time Object Detection Framework*.
➤ Provides overview of YOLOv8 features and performance improvements.

3. Ultralytics YOLO GitHub Repository
Jocher, G., Chaurasia, A., Qiu, J. (2023). *YOLO by Ultralytics*.
➤ Source code and implementation details of YOLO models.

4. YOLOv8 Citation Reference (SCIRP)
Jocher, G., Chaurasia, J., Qiu, A., Stoken, K. (2023).
➤ Official reference format for citing YOLOv8 in academic work.

5. Ultralytics YOLO Evolution Research Paper
Sapkota, R., Karkee, M. (2025). *YOLO Evolution Overview*.
➤ Discusses advancements and architecture of YOLO models including YOLOv8.

6. YOLOv8 Model Documentation (GitHub Docs)
Ultralytics (2023). *YOLOv8 Model Architecture and Usage*.
➤ Provides technical details and implementation examples.

7. Kaggle Dataset
Vehicle Detection Dataset (Kaggle)
➤ Used for training and testing the vehicle classification model.

8. OpenCV Documentation
https://docs.opencv.org/
➤ Used for image processing and video handling.

9. NumPy Documentation
https://numpy.org/doc/
➤ Used for numerical computations.

10. PyTorch Documentation
https://pytorch.org/docs/
➤ Used for deep learning model implementation.