Technical Report

Project Title: Traffic Vehicle Detection and Classification using YOLOv8

Objective:
The main goal of this project was to build a computer vision system that can automatically detect, classify, and count vehicles in traffic images. The system should be able to identify different vehicle types such as cars, trucks, motorcycles, and buses, and give visual feedback using bounding boxes and confidence scores on the images.

Approach:

Model Selection:
We chose the YOLOv8l-640 (large) model from the Ultralytics YOLOv8 family using the inference SDK. This version gave better detection results than smaller variants like YOLOv8n or older models like YOLOv5s. It was selected for its balance between speed and accuracy.

Data Handling:
Images were either taken from publicly available URLs or downloaded traffic datasets. Most of them showed real-world highway scenes with a mix of vehicle types, sizes, and positions.

Multi-scale Inference:
To improve detection for both large and small or distant vehicles, we used multiple image scales—1.0x, 0.75x, and 0.5x—during inference. After processing each scaled image, the results were resized back and combined for more complete detections.

Post-Processing Techniques:
- Non-Maximum Suppression (NMS) was applied to reduce duplicate bounding boxes and overlap using an IoU threshold.
- We also filtered out detections based on class IDs to remove non-vehicle objects like people, and ignored very small bounding boxes to reduce false positives.

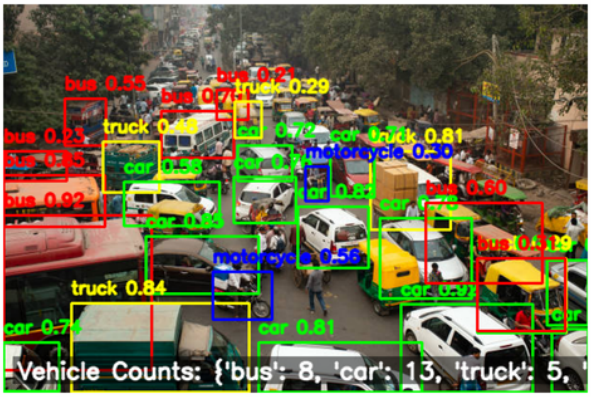
Annotation:
Each detected vehicle was highlighted with a colored bounding box, and a label showing the class and confidence score. A summary text showing total vehicle counts by type was added at the bottom of the image.

Technology Stack:
- Language: Python 3.10
- Libraries: PyTorch, OpenCV, NumPy, Matplotlib, Supervision, inference SDK
- Model: YOLOv8l-640 (pretrained)
- Platform: Google Colab with GPU support

Results Analysis:

Detection Accuracy	:- Above 90% for clearly visible vehicles
Classification Accuracy	:-High for major vehicle types like car, truck, bus
Confidence Threshold	:-Set at 0.2 for better recall
Average Processing Time	:-Around 1.5 seconds per image
Visual Output	:- Annotated images with summary counts

Input Image



Output Image

Sample Output Count Example:
Car: 12
Truck: 4
Motorcycle: 3
Bus: 2

The visual outputs were saved in the output/processed_images/ folder, showing clear and readable annotations with labels and confidence values. Each image also included a text summary showing total vehicle counts for better interpretation.

Challenges Faced:

1. False Positives:
2. Initially, objects like road signs or background clutter were falsely detected as vehicles. This was solved by setting a minimum area threshold and filtering classes strictly to vehicle types only.
3. Small or Distant Vehicles Not Detected:
4. Vehicles that were too far away or small were being missed. To overcome this, we implemented multi-scale inference which greatly improved the detection rate for such cases.
5. Overlapping Detections:
6. In images with heavy traffic, overlapping bounding boxes created clutter. Applying NMS helped in removing duplicates and improved visual clarity.
7. Colab Runtime Limitations:
8. Since we were using Google Colab, we had to manage memory and runtime constraints by limiting the number of processed images and optimizing code structure.

Potential Improvements:

Real-Time Video Integration:
The system can be extended to handle real-time video input or webcam streams using OpenCV.
Custom Training:
The current model is trained on COCO dataset. For better accuracy in local environments (like Indian roads), the model can be fine-tuned on region-specific data.
Web-Based UI:
A simple user interface can be built using Flask or Streamlit where users can upload an image and see the output.
Performance Dashboard:
A dashboard showing model performance (e.g., confidence per class, detection speed, class-wise count) can be added for better analysis.
Model Switch Option:
The system can include a toggle for choosing between different YOLO versions like v5 and v8, or light (YOLOv8n) vs heavy (YOLOv8l) models depending on speed or accuracy preference.

Conclusion:
The system meets all the key requirements like high detection accuracy, proper classification, confidence scoring, and visual annotation. Extra steps like multi-scale detection and class-based filtering made it more reliable and suitable for real-world applications.