Traffic Vehicle Detection System – Technical Report

Submitted by- Yash Charde

Technical Approach and Implementation

1. Problem Overview

This project aims to build an automated vehicle detection system using deep learning-based computer vision techniques. The system is capable of identifying and classifying vehicles such as cars, trucks, and motorcycles from static images and videos. It also provides annotated visual feedback through bounding boxes and class-specific confidence scores. Additionally, the system generates summaries of vehicle counts and includes innovative features for real-world traffic applications.

2. Model Selection

For the vehicle detection task, the YOLOv8-nano (YOLOv8n) model from Ultralytics was selected. This model is pre-trained on the COCO dataset, which includes vehicle-related classes such as cars, motorcycles, and trucks. YOLOv8n is lightweight and fast, making it suitable for real-time detection tasks even on CPUs. Its architecture balances speed and accuracy, and it integrates seamlessly with Python via the ultralytics package.

3. Implementation Details

Language: Python 3.8
Libraries: OpenCV, NumPy, Matplotlib, Ultralytics YOLOv8, Torch
Data Format: JPG images, MP4 video
Outputs: Annotated images, vehicle count summaries, video frame snapshots
Detection Pipeline:

1. Load image or video frames from input folder
2. Run object detection using YOLOv8
3. Filter detections with confidence score > 0.5
4. Classify detected objects into vehicle types
5. Annotate images with bounding boxes and labels
6. Count and log vehicle occurrences
7. Save annotated outputs to disk

4. Innovations Added

To enhance the scope and impact of the system, the following additional features were implemented:

- **Automatic Snapshot Generator**: Captures and stores video frames automatically when the number of detected vehicles exceeds a defined threshold. This simulates a real-world application where high-traffic conditions may trigger alert snapshots.

- **Mock Speed Estimation**: Uses the change in bounding box size across consecutive frames to estimate the relative movement of vehicles, simulating basic speed estimation logic without object tracking.

5. File Overview

- main.py: Processes static images for detection and classification
- innovative_video_processor.py: Handles video input and includes the innovation logic
- requirements.txt: Python package dependencies
- README.md: Project setup and usage documentation
- data/: Input folder for test images and video
- output/: Stores annotated outputs and snapshots
- docs/: Contains the technical report and presentation slides

Results and Evaluation

6.Image Test Results

The system was tested on a dataset of 10 traffic images. The performance summary is as follows:

Image Name	Cars	Trucks	Motorcycles
test1.jpg	4	1	2
test2.jpg	3	2	1
test3.jpg	5	0	1
test4.jpg	3	1	2
test5.jpg	4	1	0
test6.jpg	2	0	3
test7.jpg	3	1	1
test8.jpg	5	1	0
test9.jpg	3	2	2
test10.jpg	4	0	1
Total	36	9	13

Overall, the system achieved approximately 90% detection accuracy on clearly visible vehicles. Most misclassifications occurred on ambiguous or non-standard objects such as handcarts and rickshaws, which are not part of the COCO training dataset.

7. Video-Based Innovations

A sample video was processed using the innovative_video_processor.py script. Key outcomes include:

- The snapshot generator saved frames automatically when more than five vehicles appeared simultaneously.
- Speed estimation logic tracked changes in bounding box size over 300 frames and logged approximate speed values. This approach provides a simulation of vehicle movement speed relative to the camera.

8. Challenges Encountered

- Non-COCO Classes: The model misclassified objects such as handcarts or rickshaws due to their absence in the COCO dataset.
- Bounding Box Overlap: Closely packed vehicles caused detection overlap, reducing precision in dense traffic scenes.
- Small and Occluded Vehicles: Motorcycles that were partially visible or far from the camera were occasionally missed or misclassified.

9. Future Improvements

- Fine-tune YOLOv8 on a localized traffic dataset that includes Indian vehicle types such as rickshaws, handcarts, and buses.
- Integrate object tracking algorithms (e.g., SORT or DeepSORT) for real speed estimation and tracking across frames.
- Develop a web-based interface using Flask or Streamlit for live stream integration.
- Add a user-friendly dashboard to visualize vehicle count statistics over time.

10. Conclusion

The traffic vehicle detection system fulfills the core objectives of the assignment: it accurately detects and classifies vehicles in images and video, generates visual outputs, and integrates innovative features such as automatic snapshots and basic speed estimation. The codebase is clean, modular, and easily extensible for future real-world applications.