Introduction

This report outlines the development of a vehicle detection and classification system for urban street scenes. The system's primary objective is to detect vehicles and classify them into three categories: **cars, trucks, and buses**. Various techniques and frameworks were explored, including **MobileNetV2** for classification and **YOLOv8** for object detection. The project also involved deploying a **Streamlit application** for user interaction, enabling image and video uploads for real-time inference.

Dataset Selection and Preparation

For this project, an initial dataset was found on Kaggle: 5 Vehicles for Multi-category Classification. The dataset consisted of images of various vehicles, and I focused on **cars, buses, and trucks**, aligning with the project's requirements. Although SSD implementations like this TensorFlow SSD repository were considered, I opted for **YOLOv8** because it allowed training on **COCO-pretrained weights** without the need to download or label new datasets manually. YOLOv8 simplifies dataset handling with a single command-line invocation for training.

Model Architecture and Fine-tuning

MobileNetV2 was used for vehicle classification through **transfer learning**. MobileNetV2 is a lightweight neural network optimized for mobile and edge devices, making it suitable for real-time applications.

In parallel, I implemented **YOLOv8** from the Ultralyitcs package for object detection. YOLO (You Only Look Once) is known for its balance between accuracy and speed, which was crucial for this project, as it aimed for real-time performance.

YOLOv8 Object Detection

- Pre-trained on the **COCO dataset**, YOLOv8 was fine-tuned on three categories: **car, truck, and bus**.
- The model training process was simplified by leveraging pre-labeled data from the **COCO dataset**, eliminating the need for manual labeling.
- The **mAP (mean Average Precision)** metric was used to evaluate the object detection model's performance.

After fine-tuning, the model's performance metrics on a test set of 128 images were as follows:

Metric	Car	Bus	Truck	Overall
Precision	0.757	0.847	0.866	0.823
Recall	0.587	0.794	0.750	0.710
mAP@0.5	0.667	0.944	0.870	0.827
mAP@0.5:0.95	0.368	0.873	0.618	0.619

These results show:

- **Precision**: YOLOv8 performed well on all three classes, especially buses and trucks, where precision reached 0.847 and 0.866, respectively. This indicates the model's ability to accurately detect these vehicles with fewer false positives.
- **Recall**: The model's recall values are highest for buses (0.794) and slightly lower for cars (0.587), meaning the model was more effective in detecting buses.
- **mAP@0.5**: This metric, which measures accuracy at a 50% Intersection over Union (IoU) threshold, showed strong performance for buses (0.944) and trucks (0.870), while cars had a slightly lower score (0.667).
- **mAP@0.5:0.95**: This stricter metric, which measures accuracy across a range of IoU thresholds, showed that buses (0.873) and trucks (0.618) had better performance compared to cars (0.368), indicating that the model struggled slightly with precise car detections.

The **inference speed** of the model on the Apple M1 CPU was **142.2ms** per image, which is sufficient for near real-time applications, though further optimization could be pursued for deployment on edge devices.

Performance Optimization

To achieve real-time performance, I explored multiple optimizations:

- **Model Export to ONNX**: YOLOv8 was exported to the **ONNX** format for further optimization. This allowed leveraging **TensorRT** on systems supporting it, achieving lower inference latency.
- **Streamlit Deployment**: The system was deployed using **Streamlit**, allowing users to upload images or videos for object detection in real-time. The Streamlit interface was essential for demonstrating the model's capabilities interactively.

Challenges Faced

1. **Dataset Selection**: Initially, finding a suitable labeled dataset for the SSD implementation was a challenge. The SSD-TensorFlow repository required labeled data, which would have been time-consuming to generate. This led me to adopt YOLOv8 due to its ease of use with COCO data, providing a more streamlined approach.
2. **Performance vs. Speed**: While SSD offered a good balance between speed and performance, I opted for YOLOv8 due to its robustness and better out-of-the-box performance on the targeted categories. However, MobileNetV2 was chosen for classification because of its lightweight design, ideal for real-time applications.
3. **Model Optimization**: Exporting the YOLOv8 model to ONNX introduced challenges related to inference on different hardware. Ensuring compatibility and maintaining performance across systems was a primary concern during deployment.
4. **Real-time Deployment**: Deploying the model with **Streamlit** while ensuring that the system could handle video streams in real time was another challenge. Streamlit, while user-friendly, posed some latency issues during video processing, which I mitigated using efficient image resizing and batch processing.

Key Decisions

1. **YOLOv8 vs SSD**: After researching the pros and cons of SSD, YOLOv8 was chosen for object detection due to its ease of use, excellent pre-trained weights on COCO, and strong balance between speed and accuracy.
2. **MobileNetV2 for Classification**: MobileNetV2's efficiency and accuracy were ideal for the classification task, especially considering the system's focus on lightweight models suitable for real-time inference.
3. **ONNX for Optimization**: Exporting YOLOv8 to the ONNX format allowed me to take advantage of hardware-accelerated inference engines like **TensorRT**, significantly improving inference times for deployment.

Conclusion and Future Work

The vehicle detection and classification system successfully identifies and categorizes vehicles in urban street scenes, balancing speed and performance. The choice of YOLOv8, combined with MobileNetV2 and the ONNX export for optimization, proved effective in meeting the project's real-time requirements.

Future improvements include:

- Expanding the dataset to include more vehicle categories.
- Further optimizing the Streamlit application for video processing.
- Experimenting with more hardware accelerators (e.g., **NVIDIA TensorRT**) to boost performance.

The source code, trained model weights, and detailed setup instructions are available in my GitHub repository.