Vehicle Tracking System

Pre-Interview Task (Senior Machine Learning Engineer)

Project Overview

This project implements a real-time **Vehicle Tracking System** using YOLOv8 for object detection and tracking. The system is designed to run efficiently on edge devices like the NVIDIA Jetson Nano while providing a graphical user interface (GUI) for user-friendly interaction. The application supports multiple video sources, including webcam, IP cameras, and video files.

Key Tasks

Task 1: Model Development and Optimization

For this task, I used YOLOv8n to develop a custom vehicle tracking system, selecting this model for its stability and efficiency. I created a dataset of 398 images across 11 vehicle classes (3-Wheeler, 4-Wheeler, Bus, Heavy-Truck, Jeep, Medium-Truck, Micro-Bus, Mini-Bus, Mini-Truck, Motorcycle, and Sedan-Car) collected from Uttara Rajlokkhi for R&D purposes and annotated using Roboflow. The model was fine-tuned with 25 epochs, an image size of 800, batch size of 8, and patience set to 10. I used the AdamW optimizer (lr=0.000625, momentum=0.9), with an initial learning rate of 0.01 and momentum of 0.937, while enabling plot visualization (plots=True) to monitor the training progress effectively.

Features:

- Custom YOLOv8n model for real-time vehicle detection.
- 398-image dataset covering 11 vehicle types.
- Annotated using Roboflow for precise labeling.
- Optimized training with:
  - 25 epochs for proper convergence.
  - Image size: 800 × 800 for feature extraction.
  - Batch size: 8 balancing memory and performance.
  - Patience: 10 preventing premature stopping.
- AdamW optimizer for efficient learning.

Task 2: Real-Time Object Tracking

For real-time object tracking, I used Supervision's ByteTrack, which offers efficient multi-object tracking with strong re-identification capabilities. Supervision enhances tracking accuracy, reduces ID switches, and ensures object consistency across frames. To handle occlusions and re-identification, I set track_thresh=0.5, which controls the detection confidence threshold for tracking, ensuring only reliable detections are tracked. Additionally, track_buffer=30 determines how many frames an object can be lost before it is removed, allowing for smoother tracking during temporary occlusions.

Features:

- Implemented Supervision's ByteTrack for multi-object tracking.
- Enhanced re-identification for accurate tracking.
- Reduced ID switches for better object continuity.
- track_thresh:0.5 ensures only high-confidence detections are tracked.
- track_buffer:30 allows tracking objects even when briefly occluded.

Task 3: Edge Device Deployment

For edge device deployment, I converted my pre-trained YOLOv8 model to ONNX format for efficient inference. Using Ultralytics YOLO, I exported the model with model.export(format="onnx"). Additionally, I optimized it for NVIDIA TensorRT by generating a TensorRT engine with FP16 precision using "***trtexec --onnx=final.onnx --saveEngine=final.trt --fp16***", ensuring faster and more efficient inference on edge devices.

Features:

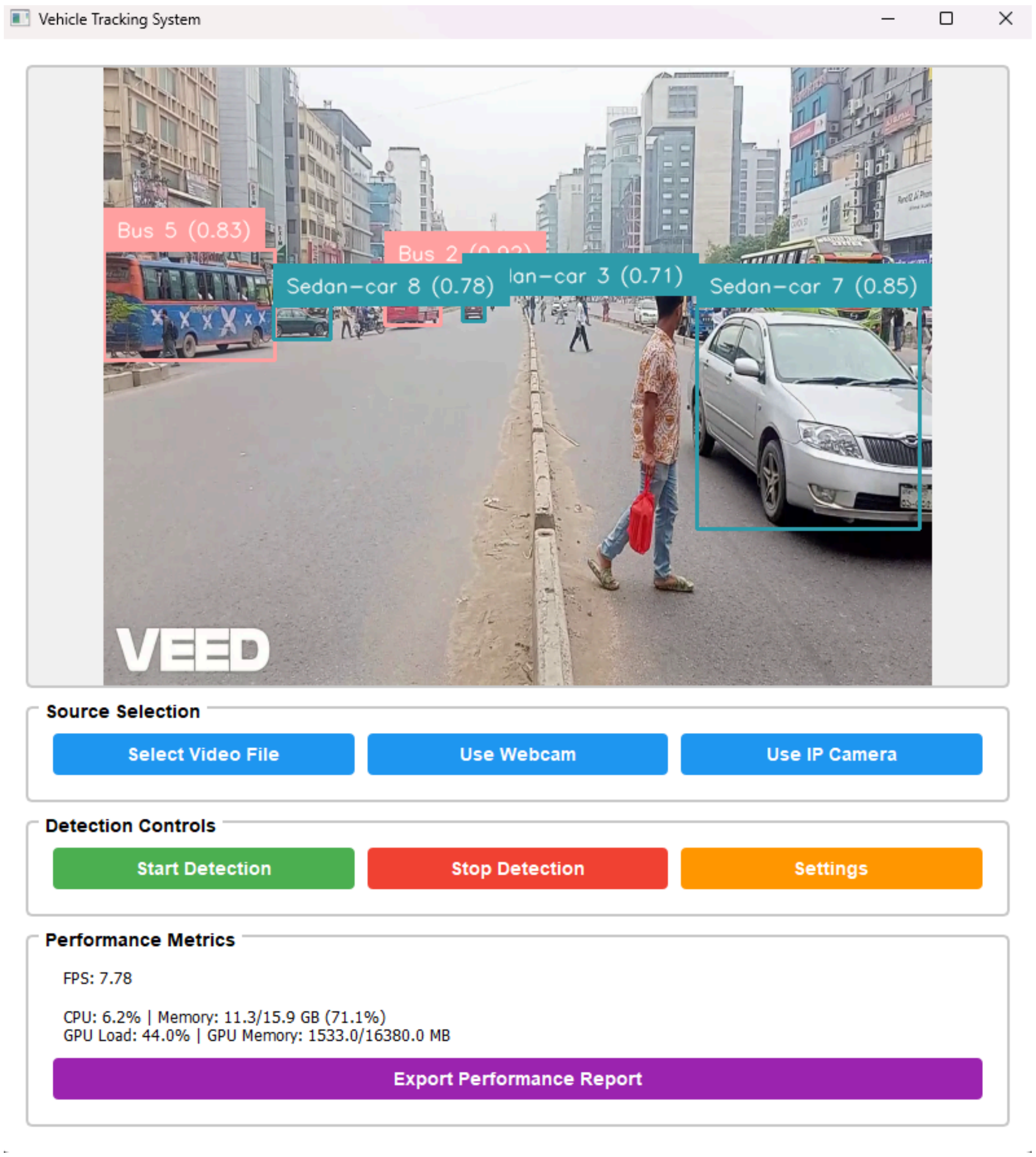
- Converted YOLOv8 model to ONNX format for edge inference.
- Optimized inference with NVIDIA TensorRT for speed improvement.
- Used FP16 precision to enhance computational efficiency.
- Enabled deployment on NVIDIA Jetson Nano for real-world applications.

Task 4: Graphical User Interface (GUI)

For the Graphical User Interface (GUI), I used PyQt5 to create an intuitive and responsive interface. The GUI displays live annotated video feeds with detected vehicles and includes controls for starting, stopping, and selecting video sources such as video files, webcams, and IP cameras. It features a settings dialog for adjusting detection parameters, including tracking threshold, confidence score, IOU threshold, and tracking buffer, allowing real-time tuning for optimal performance.

Features:

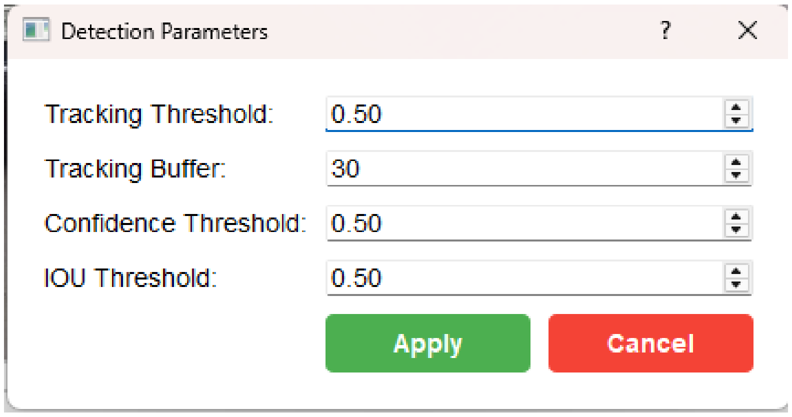
- Developed GUI with PyQt5 for an interactive user experience.
- Live annotated video feed with real-time vehicle tracking.
- Controls for video source selection (file, webcam, IP camera).
- Start/Stop functionality for seamless operation.
- Settings panel for real-time parameter tuning (tracking threshold, confidence score, IOU threshold, tracking buffer).



FPS: 7.78

CPU: 6.2% | Memory: 11.3/15.9 GB (71.1%)
GPU Load: 44.0% | GPU Memory: 1533.0/16380.0 MB

Figure 1: User Interface of Vehicle Tracking System



Tracking Threshold: 0.50
Tracking Buffer: 30
Confidence Threshold: 0.50
IOU Threshold: 0.50

Figure 2: User Interface of Settings panel

Performance Monitoring & Reporting

For performance monitoring, the system tracks FPS, CPU usage, memory consumption, and GPU utilization, providing real-time insights. Metrics are updated every second to ensure continuous monitoring of resource usage. Additionally, it allows exporting performance reports with system metrics for further analysis and evaluation.

Features:

- Real-time monitoring of system performance (FPS, CPU, memory, GPU).
- Updated every second for live performance tracking.
- Performance reports export for further analysis.
- Ensures smooth and efficient tracking system operation.

Setup and Installation Guide

Prerequisites

- **OS:** Windows, Linux, or macOS
- **Python:** 3.8+
- **Optimization:** CUDA (GPU acceleration), PyTorch, NVIDIA TensorRT
- **Dependencies:** Listed in *requirements.txt*

Installation Steps

1. **Create a Virtual Environment**

2. **Install Dependencies**

   pip install -r requirements.txt

Running the Application

- **Start GUI:** *python Gui.py*
- **Train Model:** Run *yolo_train.ipynb* in Jupyter Notebook
- **Convert Model to ONNX:** *python Convert_to_Onnx.py*
- **Optimize for TensorRT:**
  - *trtexec --onnx=final.onnx --saveEngine=final.trt --fp16*

Conclusion

This vehicle tracking system provides a robust real-time solution for edge-based object detection and tracking. With YOLOv8's accuracy, ByteTrack's efficiency, and GUI-based controls, the system is well-suited for automated traffic monitoring, fleet management, and surveillance applications.