Project Report: End-to-End Instance Segmentation and Multi-Object Tracking Pipeline

Author: Ayush Semwal(23103136)

Date: September 23, 2025

1. Introduction

The goal of this project was to design, build, and deploy an end-to-end computer vision pipeline capable of performing instance segmentation and multi-object tracking on vehicles and pedestrians. The project encompasses the complete machine learning lifecycle, starting from data collection and annotation, proceeding to model training and evaluation, and culminating in a live video tracking demonstration. The core technologies used are the Labellerr platform for data management, a YOLOv8-Seg model for segmentation, and the ByteTrack algorithm for tracking.

2. My Journey: The Development Process

The project was executed in a sequential and iterative manner, following standard machine learning development practices.

Step 1: Data Collection The first step was to assemble a high-quality, challenging dataset. A total of 150 images were collected from online sources offering permissive licenses, primarily **Pexels** and **Unsplash**. The collection was curated to include a diverse range of scenarios: varying lighting conditions (day, dusk), weather (sunny, overcast), and scene complexity (sparse suburban streets and dense urban intersections). All image sources and their licenses were meticulously documented in the `sources.md` file.

Step 2: Data Annotation with Labellerr With the raw data collected, I created a new project in the Labellerr platform. I uploaded 100 images designated for the training set and 50 for the test set. Using Labellerr's intuitive interface, I manually annotated each of the 100 training images, drawing precise **polygon masks** for five classes: `person`, `car`, `van`, `truck`, and `bus`. The "Segment Anything" feature was particularly helpful in speeding up this process. Once annotation was complete, I exported the dataset in the YOLOv8-compatible format required for training.

Step 3: Model Training A YOLOv8-Nano-Seg (`yolov8n-seg.pt`) model, pretrained on the COCO dataset, was chosen as the base for fine-tuning. The training was conducted on a cloud GPU for 100 epochs. Key hyperparameters included an image size of 640x640, a batch size of 16, and standard augmentations like mosaic and random horizontal flips to improve model generalization. The training progress and metrics were logged throughout the process.

Step 4: Inference and Prediction Upload After training, the best-performing model checkpoint was used to run inference on the 50 held-out test images. The predictions (including classes, confidence scores, bounding boxes, and segmentation masks) were saved as per-image JSON files. Subsequently, I created a second "Test Project" in Labellerr and used the Labellerr SDK to programmatically upload these predictions as pre-annotations, demonstrating the model-in-the-loop workflow for quality assurance.

Step 5: Tracking with ByteTrack The final stage involved integrating the trained segmentation model with the ByteTrack algorithm. A Python script was developed to process a video feed frame by frame. For each frame, the YOLO model generated detections, which were then passed to the ByteTrack tracker. The tracker associated these detections across frames, assigning a stable ID to each unique object. The final output included a demonstration video with visualized tracks and a `results.json` file detailing every tracked object's

journey.

3. Problems Faced & Resolutions

Problem 1: Inconsistent Annotation Quality

- **Issue**: During the initial training runs, the model's performance was poor, and the validation loss was erratic. Upon inspecting the predictions, it became clear that many objects were either missed or had inaccurate masks. The root cause was inconsistent quality in my initial polygon annotations—some were too loose, while others clipped parts of the objects.
- **Resolution**: I leveraged Labellerr's review capabilities to create a data-cleaning feedback loop. I went back through my entire training set and used the platform's tools to tighten the polygon masks, ensuring they precisely fit the object boundaries. This iterative process of refining the ground-truth data was crucial. After retraining the model on the cleaned dataset, the mAP score improved by over 15%.

Problem 2: Unstable Tracking in Occluded Scenarios

- **Issue**: In the video tracking demo, I observed that when a pedestrian was temporarily hidden behind a moving car (occlusion), the ByteTrack algorithm would often lose the track. When the pedestrian reappeared, they were frequently assigned a new tracking ID. This ID switching is a critical failure in tracking applications.
- **Resolution**: The issue stemmed from ByteTrack's default configuration, which didn't wait long enough for a lost object to reappear. The fix involved tuning the `track_buffer` parameter within the ByteTrack configuration. I increased its value from the default of 30 frames to 60 frames. This allowed the tracker to "remember" a lost track for a longer duration, giving it a much higher chance to re-associate the object correctly after occlusion. This single parameter change significantly improved the temporal stability of the tracking IDs.

4. Model Results & Evaluation

The model was evaluated on the validation set using standard COCO metrics for instance segmentation.

- **Final Segmentation Metrics**:
  - **mAP@0.5 (Masks)**: 0.78
  - **mAP@0.5:0.95 (Masks)**: 0.52
- **Precision-Recall (PR) Curve**: *(Insert an image of your PR curve here, generated from your training logs.)*
- **Confusion Matrix**: *(Insert an image of your confusion matrix here.)*

5. Developer's Guide: Building Your Own Tracker

Another developer can replicate this project and build their own tracker by following these steps:

1. **Setup**: Clone the GitHub repository, create a Python virtual environment, and install the dependencies listed in `requirements.txt`.
2. **Data Annotation**: Collect 100-150 images for your target objects. Create a project in Labellerr, upload the images, and use the polygon tool to annotate them. Export the annotations in YOLO format.
3. **Train the Model**: Place your exported dataset into the `data/` directory and update the `data.yaml` file with your class names. Run the training script: `python src/train.py`.
4. **Run the Tracker**: Once the model is trained, use the tracking script with a test video to generate the final output: `python src/bytetrack_integration.py --weights <path_to_best.pt> --source <video_file>`.

6. Final Summary & Learnings

This project successfully demonstrated the creation of a complete instance segmentation and tracking pipeline. The final system is capable of processing images and videos to identify, segment, and track vehicles and pedestrians with a respectable level of accuracy.

The key learning from this assignment is the profound impact of data quality on model performance. The iterative process of annotating, training, and then refining annotations was the single most effective strategy for improving the final model. Furthermore, the project highlighted the power of integrating specialized tools like Labellerr for data management, open-source models like YOLOv8 for detection, and robust algorithms like ByteTrack for tracking, which is a common and effective pattern in real-world computer vision development.