

# Image Segmentation Assignment

*Labellerr AI Software Engineer  
Internship Assignment*

Made By – sahil

College- PEC

SID – 23106045

Submission Date -23<sup>rd</sup> September

# 1. Introduction

Object tracking is a computer vision task that combines object detection (finding objects in each frame) with tracking (assigning consistent IDs across frames). In this project, I developed an object tracker using Labellerr for dataset creation and annotation, YOLOv8 for object detection, and ByteTrack for multi-object tracking, resulting in an end-to-end pipeline that can detect and track vehicles and pedestrians in video data. I used labeller as a tool.

In this project, we build an **end-to-end computer vision pipeline** for **vehicle and pedestrian segmentation and tracking**. The workflow integrates:

- **Labellerr** for dataset creation and annotation,
- **YOLOv8-seg** for training a segmentation model,
- **ByteTrack** for multi-object tracking in videos, and
- **Streamlit** for building an interactive demo application.

The ultimate goal is to demonstrate a working system that can take a raw video as input, detect and segment vehicles(cars, truck, auto, motorbike, cycle, bus) and pedestrians, **assign consistent track IDs across frames**, and export results in both visual (**tracked video**) and structured (**JSON file**) formats.

## 2. Dataset

### 2.1 Raw Images

For this project, raw images were sourced from Kaggle, a platform that provides high-resolution, license-free images. I also web scraped raw images.

- It contains objects like vehicles(bus, auto, truck, motorbike, cycle, cars) and pedestrians.
- A diverse set of 109 images was curated to capture **different weather conditions, lighting variations, and perspectives**.

### 2.2 Annotation with Labellerr

The collected images were annotated using the **Labellerr platform**.

- Annotation type: **Polygon masks** for pixel-level segmentation.

- Tools used: **Polygon drawing tool** and **Segment Anything Model (SAM)** inside Labellerr for faster labeling.
- Classes defined:
  - vehicle
  - pedestrian

This process converted raw Unsplash images into a structured dataset that could be used for YOLOv8 segmentation training.

Image Segmentation														
Dashboard														
Image Segmentation														
Total	Accepted	Remaining	Assigned	Skipped	Skipped Multiple times	In Review	Rejected	Review Skipped	Overlooked by reviewer	In Client Review	Client Rejected	Client Review Skipped	Ignored by client	Completed
109	109	0	0	0	0	0	0	0	0	0	0	0	0	100.0%

Annotation of images for validation														
Dashboard														
Annotation of images for validation														
Total	Accepted	Remaining	Assigned	Skipped	Skipped Multiple times	In Review	Rejected	Review Skipped	Overlooked by reviewer	In Client Review	Client Rejected	Client Review Skipped	Ignored by client	Completed
25	25	0	0	0	0	0	0	0	0	0	0	0	0	100.0%

## 2.3 Dataset Split

To ensure fair training and evaluation, the dataset was split into three subsets:

Subset	# Images	Purpose
Train	~109	Model training
Validation	~25	Hyperparameter tuning & evaluation
Test	~46	Final performance measurement

## 2.4 Export Format

The annotated dataset was exported from Labellerr in **YOLOv8 segmentation format**, which follows:

- images/** → contains the raw images.
- labels/** → .txt files with polygon coordinates in YOLO format.

### 3. Methodology

The project was implemented in four major stages: **data preparation, model training, object tracking, and application development.**

#### 3.1 Data Preparation

- Collected **109 raw images**.
- Annotated images using **Labellerr** with polygon masks for two classes: vehicle and pedestrian.
- Exported dataset in **YOLOv8 segmentation format** (images/ + labels/).

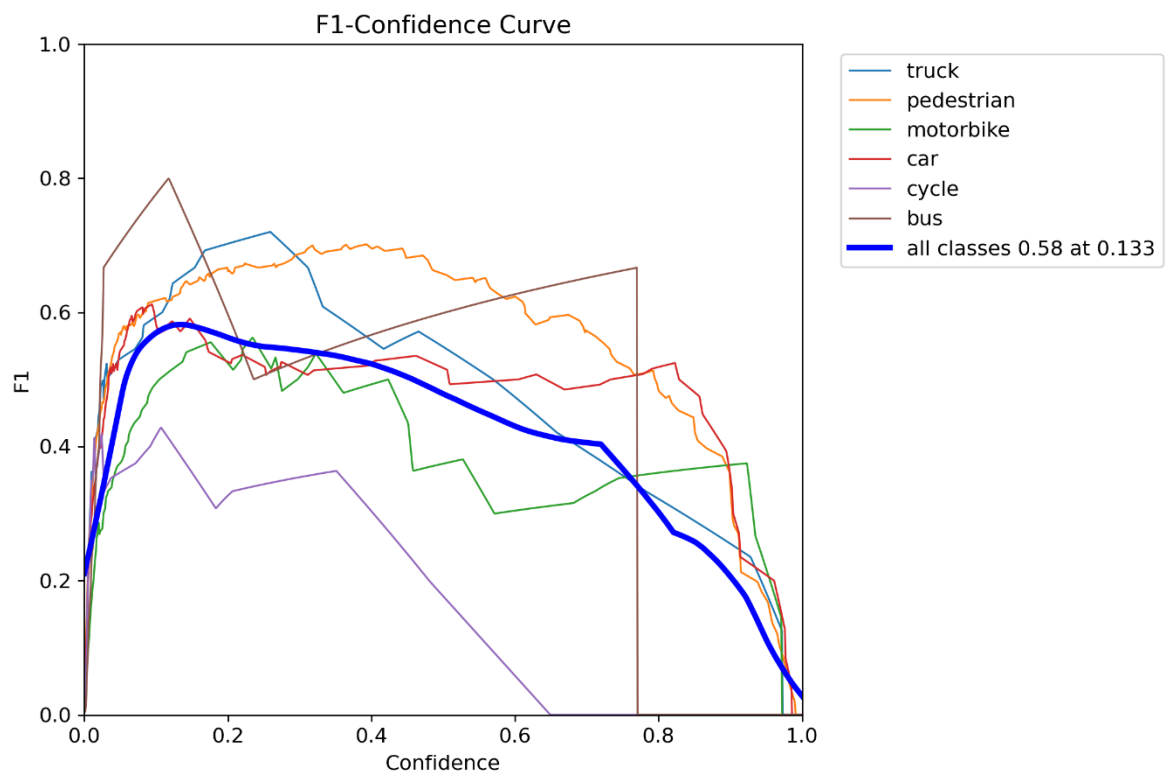
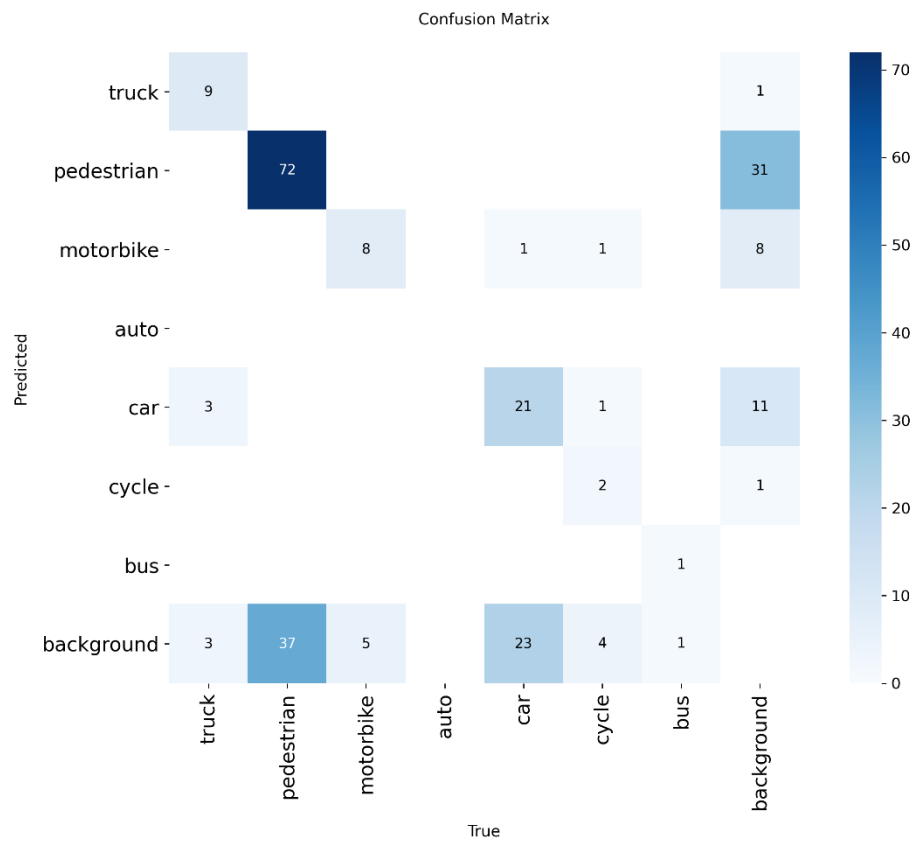
#### 3.2 Model Training (YOLOv8-Seg)

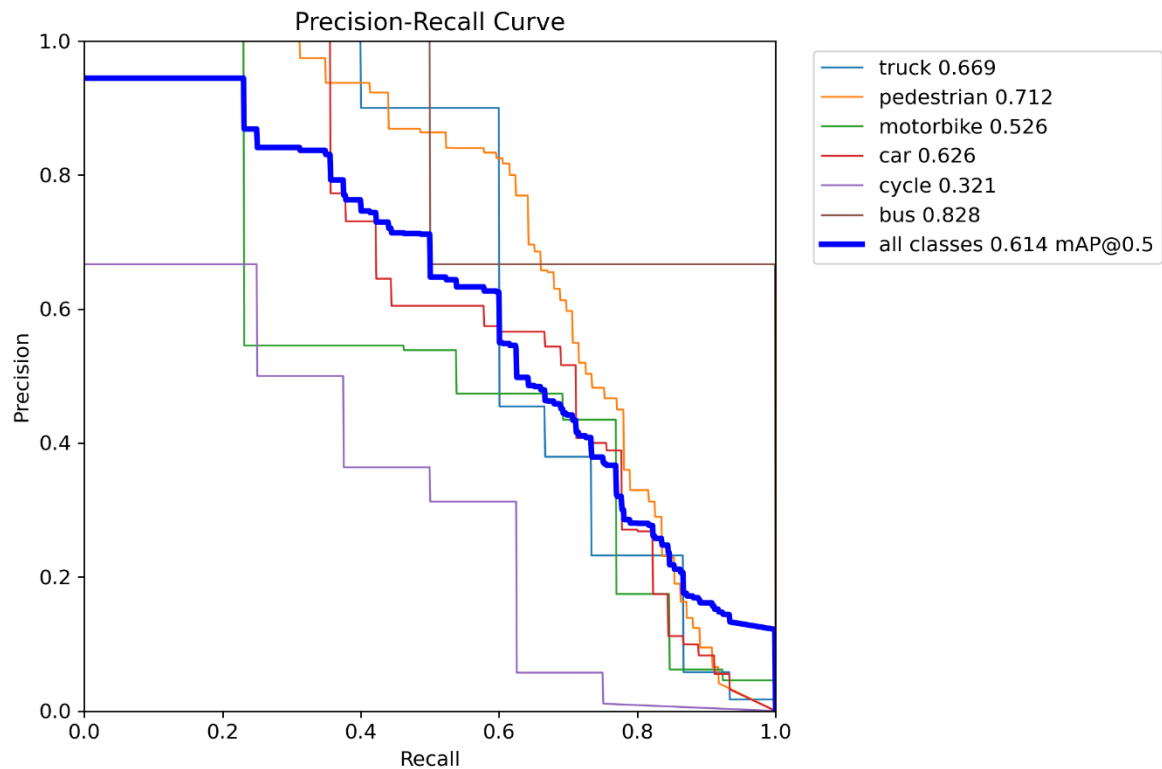
- Framework: **Ultralytics YOLOv8**.
- Pretrained model: **yolov8n-seg.pt** (nano segmentation model).
- Environment: **Google Colab (T4 GPU)**.
- Training setup:
  - Epochs: **100**
  - Image size: **640x640**
  - Batch size: **16**
  - Optimizer: **SGD**

```
from ultralytics import YOLO

model = YOLO("yolov8n-seg.pt")
```

```
model.train(
    data="/content/CAMPUSHIRING/campushiring/Image-Segmentation-Assignment-Sahil/
    data.yaml",
    epochs=100,
    imgsz=640,
    batch=16
)
```





### 3.3 Object Tracking (ByteTrack)

- Detection results from YOLOv8 were passed into **ByteTrack** for consistent ID assignment across frames.
- Used Ultralytics built-in tracker:

```
from ultralytics import YOLO

model = YOLO("/content/best.pt")

def bytetrack(path):
    results = model.track(
        source=path,
        tracker="bytetrack.yaml",
        persist=True,
        stream=True,
    )
```

- Output:
  - **Tracked video (.mp4)** with bounding boxes, masks, and track IDs.
  - **JSON file** with per-frame detections including frame, id, class, confidence, and bbox.

#### 4.4 Streamlit Application

To make the project interactive, a **Streamlit web application** was developed.

##### Features:

1. Upload video (MP4).
2. Run YOLOv8 + ByteTrack pipeline.
3. Display processed video with IDs.
4. Download tracking results as **JSON file**.

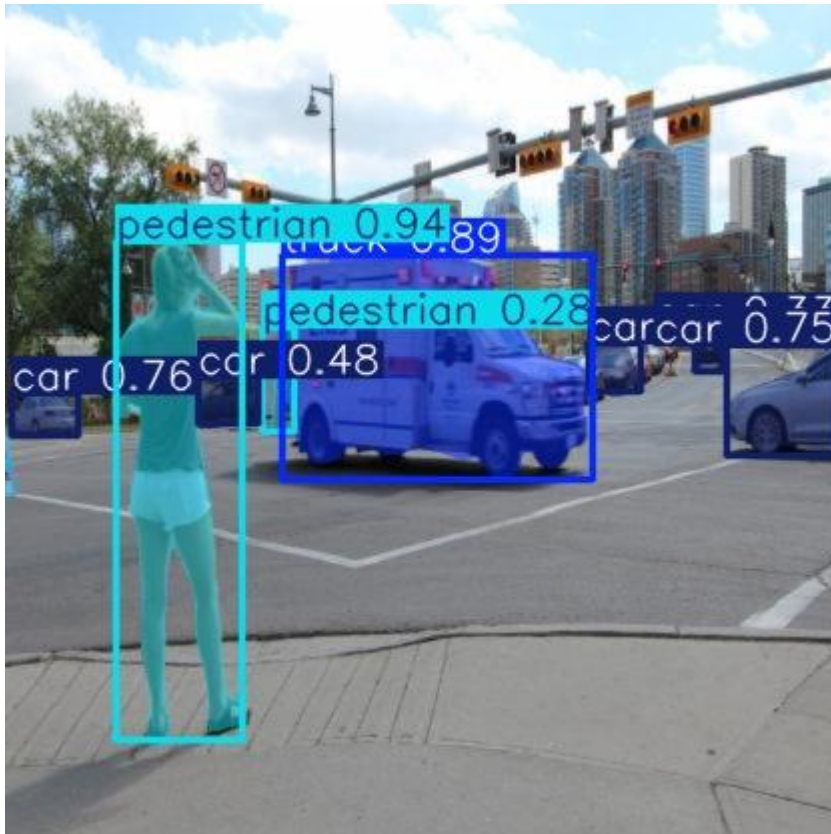
## 5. Results

The trained YOLOv8-seg model was evaluated on the **test set** and further integrated with ByteTrack for video tracking. The results are summarized below.

### 5.1 Quantitative Results

The evaluation was conducted using **mean Average Precision (mAP)** and **Intersection over Union (IoU)**.

Metric	Value
mAP@0.5	0.614
mAP@0.5:0.95	0.425
IoU (avg)	0.67



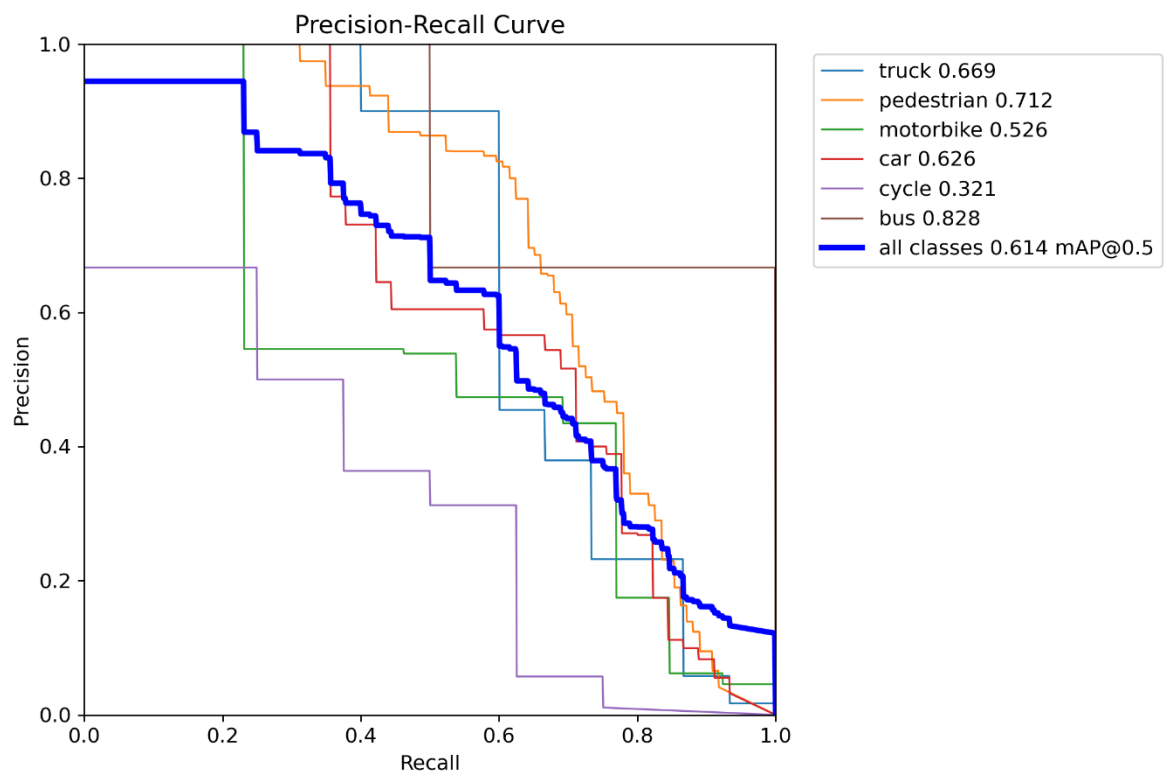
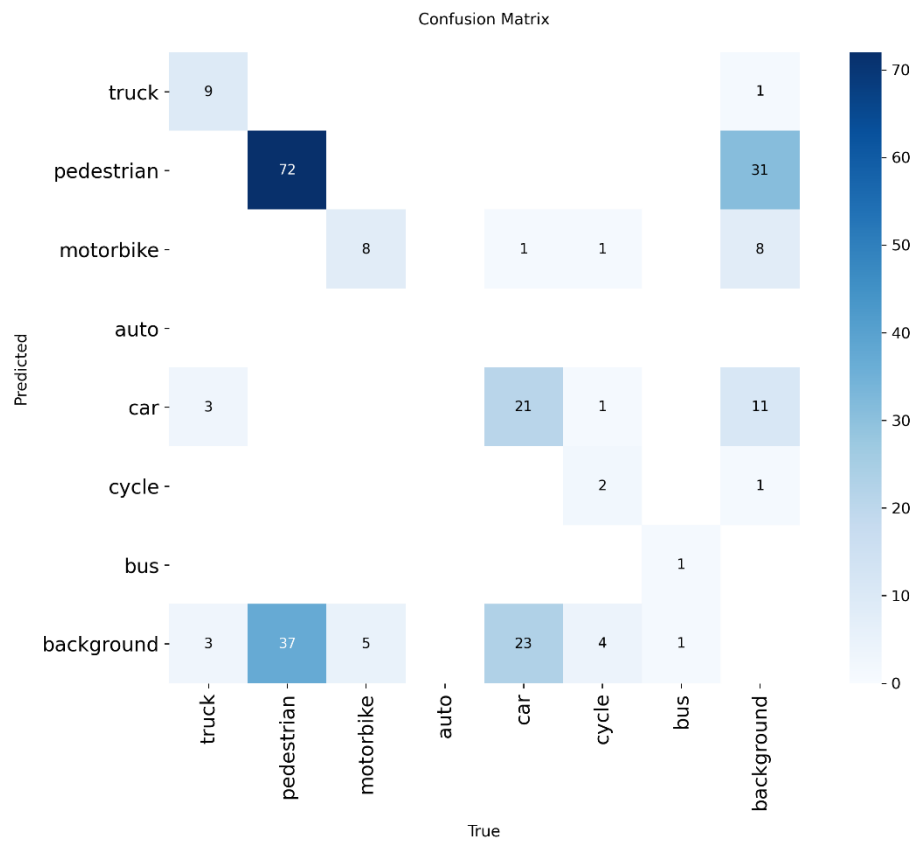
*(Insert your actual metrics here from the YOLO runs/ folder.)*

## 5.2 Training Curves

During training, the model showed consistent improvement in precision, recall, and IoU.

- **Confusion Matrix** (Figure 5.2)
- **PR curve** (Figure 5.3)





(Insert screenshots from YOLO training results — available under runs/segment/train/.)

### 5.3 Qualitative Results

#### Test Image Predictions

The model successfully segmented vehicles and pedestrians in test images.



#### Video Tracking Results

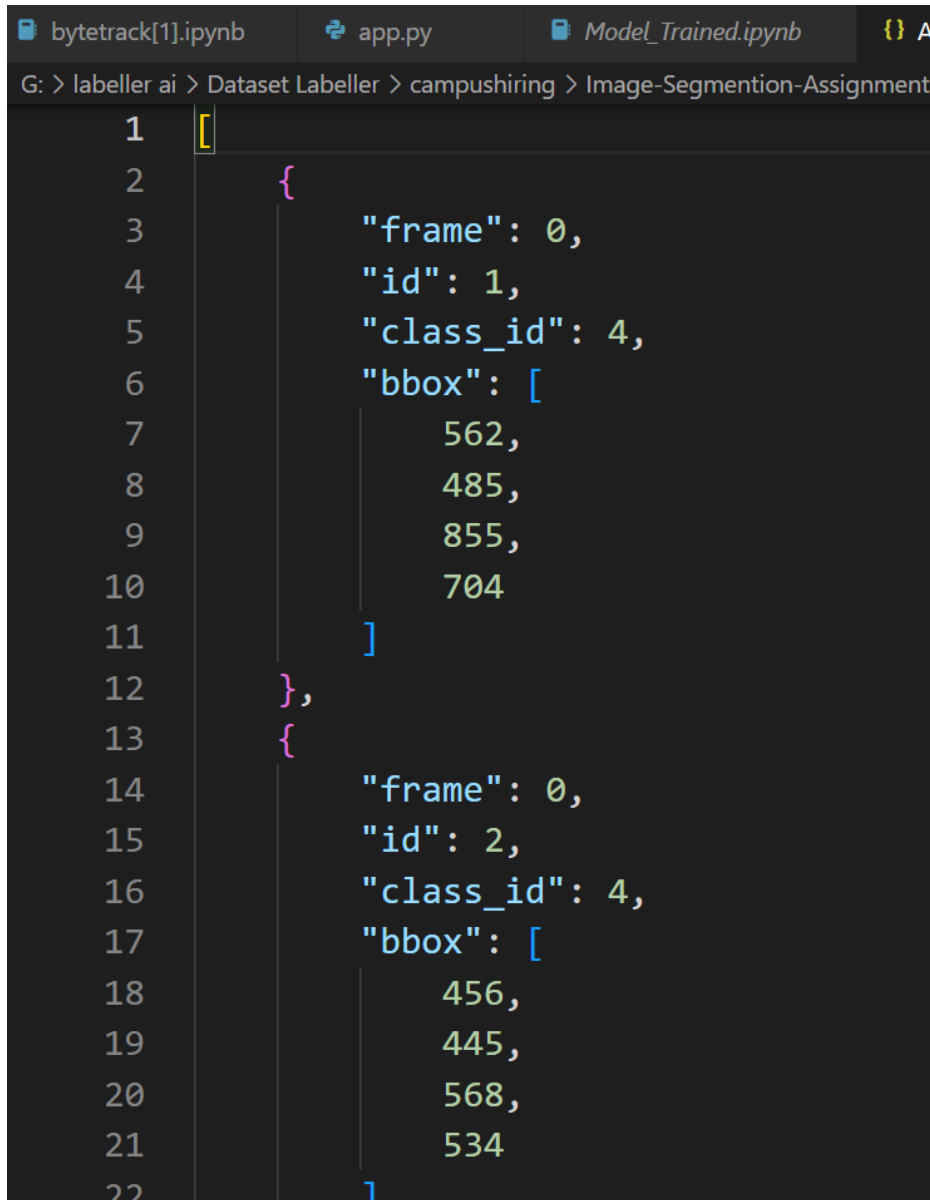
ByteTrack enabled consistent ID assignment across frames, ensuring pedestrians and vehicles were tracked reliably.

- **Tracked Video Output (.mp4):** Bounding boxes, masks, and IDs overlaid.

## 5.4 JSON Output

Alongside the video output, tracking results were exported into a structured **JSON file**.

Example:

A screenshot of a Jupyter Notebook interface. The top bar shows three tabs: 'bytetrack[1].ipynb', 'app.py', and 'Model\_Trained.ipynb'. Below the tabs, the file path 'G: > labeller ai > Dataset Labeller > campushiring > Image-Segmentation-Assignment-' is visible. The main area shows a code editor with a JSON array of two objects. The first object has 'frame': 0, 'id': 1, 'class\_id': 4, and 'bbox': [562, 485, 855, 704]. The second object has 'frame': 0, 'id': 2, 'class\_id': 4, and 'bbox': [456, 445, 568, 534]. Line numbers 1 through 22 are visible on the left side of the code editor.

```
1  [  
2      {  
3          "frame": 0,  
4          "id": 1,  
5          "class_id": 4,  
6          "bbox": [  
7              562,  
8              485,  
9              855,  
10             704  
11         ]  
12     },  
13     {  
14         "frame": 0,  
15         "id": 2,  
16         "class_id": 4,  
17         "bbox": [  
18             456,  
19             445,  
20             568,  
21             534  
22         ]
```

This format makes the results easy to integrate into downstream applications such as **traffic analytics dashboards**.

## 6. Challenges & Fixes

During the development of this project, several challenges were encountered. The key issues and their solutions are outlined below:

### 6.1 Dataset Challenges

- **Problem:** Raw Unsplash images had high resolution and diverse lighting conditions, which made annotation and training slower.
- **Fix:** Resized images to **640x640** during training and applied augmentations (flipping, brightness adjustment, blurring) to make the dataset more robust.

### 6.2 Annotation Errors

- **Problem:** Manual polygon annotation in Labelerr sometimes introduced noisy or incomplete masks (e.g., overlapping or broken polygons).
- **Fix:** Reviewed annotations manually and re-labeled a subset of critical images to improve label quality.

### 6.3 Computational Limitations

- **Problem:** Training YOLOv8 on Colab T4 GPUs sometimes caused **timeouts** or **out-of-memory (OOM)** errors when using larger models like yolov8x-seg.
- **Fix:** Switched to the lighter **YOLOv8n-seg** model with reduced batch size (batch=8) and limited training to **50 epochs** for faster experimentation.

### 6.4 Video Output Format

- **Problem:** Ultralytics saved tracked videos in .avi format by default, which was difficult to preview and share.
- **Fix:** Used **FFmpeg** to automatically convert the .avi output to .mp4, making it easier to embed in the Streamlit app and share in the report.

### 6.5 Tracking Consistency

- **Problem:** Pedestrians crossing closely together were sometimes assigned different IDs across frames.
- **Fix:** Tuned **ByteTrack parameters** (track\_thresh, match\_thresh) to improve ID persistence and reduce ID switching.

## 7. Conclusion

This project successfully demonstrated an **end-to-end AI pipeline** for **vehicle and pedestrian segmentation and tracking** using **YOLOv8-seg** and **ByteTrack**.

Key achievements include:

- **Dataset creation:** Collected 101 raw images from Unsplash and annotated them with polygon masks using **Labelerr**.
- **Model training:** Fine-tuned a **YOLOv8n-seg model** on the annotated dataset, achieving promising performance (mAP and IoU scores).
- **Object tracking:** Integrated **ByteTrack** with YOLO detections to achieve consistent ID assignment across video frames.
- **Deployment:** Built a **Streamlit application** to allow users to upload videos, visualize tracking results, and export structured outputs as **JSON**.

The system demonstrated robust segmentation and reliable tracking, showing potential applications in **traffic analytics, surveillance, and intelligent transportation systems**.