Drone-Based Object Detection and Tracking: Implementation and Analysis Using the Visdrone Dataset by Yolov8

Student: AmirHossein Eslami

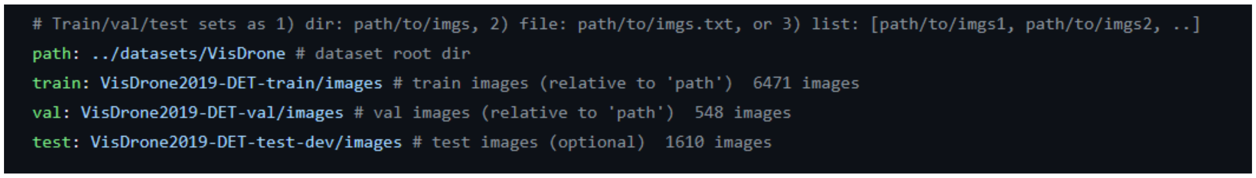
Iran University Of Science And Technology

Spring 2024

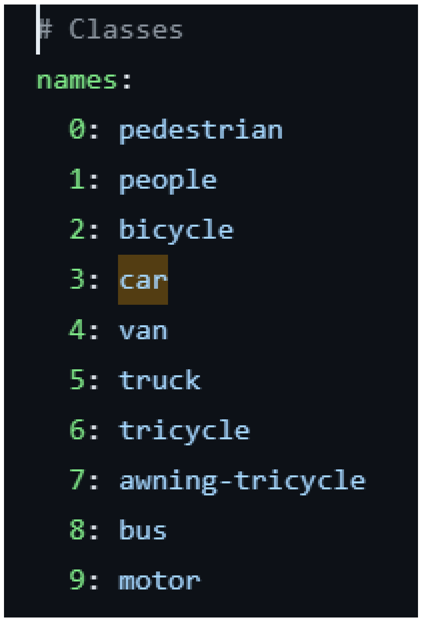
Table of Contents

The dataset we need to use to detect vehicles from above is one that consists of aerial images. Among the ready-to-use datasets available on the Ultralytics website, the Visdrone dataset meets our requirements.

Introduction to the Visdrone Dataset To get more familiar with this dataset, we visit the Ultralytics website and open the relevant dataset section. "The VisDrone dataset is a large-scale benchmark created by the AISKYEYE team at the Machine Learning and Data Mining Lab, Tianjin University, China. It includes accurately annotated ground-truth data for various computer vision tasks related to drone-based image and video analysis.



```
# Train/val/test sets as 1) dir: path/to/imgs, 2) file: path/to/imgs.txt, or 3) list: [path/to/imgs1, path/to/imgs2, ..]
path: ../datasets/VisDrone # dataset root dir
train: VisDrone2019-DET-train/images # train images (relative to 'path')  6471 images
val: VisDrone2019-DET-val/images # val images (relative to 'path')  548 images
test: VisDrone2019-DET-test-dev/images # test images (optional)  1610 images
```



```
# Classes
names:
  0: pedestrian
  1: people
  2: bicycle
  3: car
  4: van
  5: truck
  6: tricycle
  7: awning-tricycle
  8: bus
  9: motor
```

VisDrone consists of 288 video clips with 261,908 frames and 10,209 still images captured by various cameras mounted on drones. The dataset covers a wide range of aspects, including location (14 different cities across China), environment (urban and rural), objects (pedestrians, vehicles, bicycles, etc.), and density (sparse and crowded scenes). The dataset was collected using different drone platforms under various scenarios and weather and lighting conditions. These frames are manually annotated with over 2.6 million bounding boxes for objects like pedestrians, cars, bicycles, and tricycles. Features like scene visibility, object class, and occlusion are also provided for better data utilization."

Structure of the Visdrone Dataset The VisDrone dataset is organized into five main subsets, each focusing on a specific task:

1. Task 1: Object Detection in Images
2. Task 2: Object Detection in Videos
3. Task 3: Single Object Tracking
4. Task 4: Multiple Object Tracking
5. Task 5: Crowd Counting

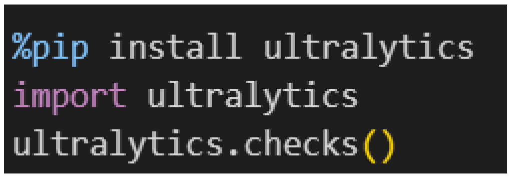
Applications The VisDrone dataset is widely used for training and evaluating deep learning models in drone-based computer vision tasks such as object detection, object tracking, and crowd counting. The diverse sensor data, annotations, and dataset features make it a valuable resource for researchers and professionals in drone-based computer vision.

Dataset YAML A YAML (Yet Another Markup Language) file is used to define the dataset configuration. It includes information about dataset paths, classes, and other relevant details. For the Visdrone dataset, the VisDrone.yaml file is maintained at this link, which we will review next.

Information from the Visdrone Dataset GitHub Page This section provides the basic information needed to access and use the dataset, such as the dataset address and images used for training. Additionally, approximately 1610 images are available for testing the dataset.

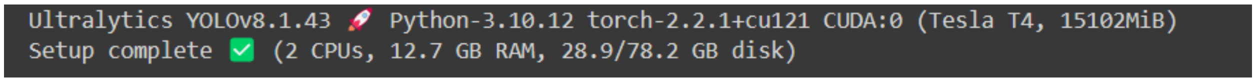
The list of dataset classes is shown in this image. As seen, this dataset includes 10 classes of objects and entities that it can track and detect. Further, the download codes for the dataset are provided for developers.

Setting up and Installing Ultralytics First, we open a notebook in Google Colab. Then, we write the following code to install Ultralytics:



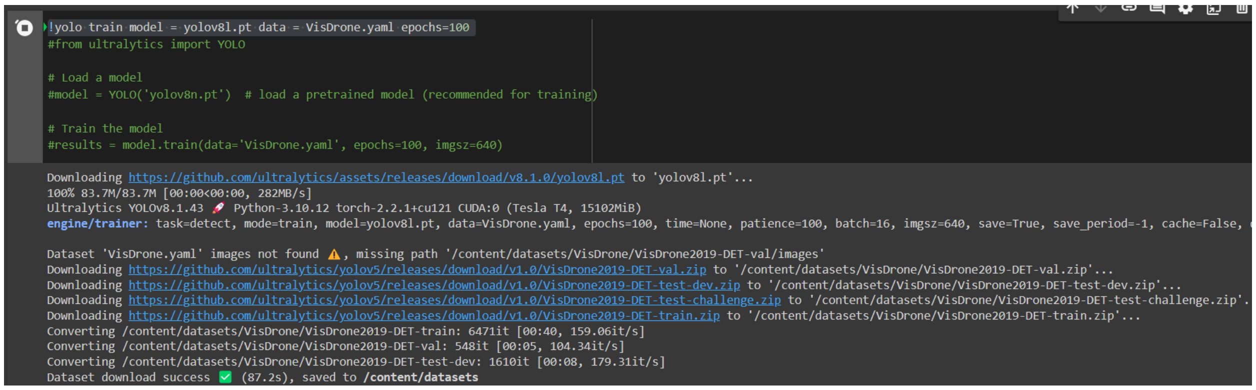
```
%pip install ultralytics
import ultralytics
ultralytics.checks()
```

After running the code, we observe that the installation is successful.



```
Ultralytics YOLOv8.1.43 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Setup complete ✅ (2 CPUs, 12.7 GB RAM, 28.9/78.2 GB disk)
```

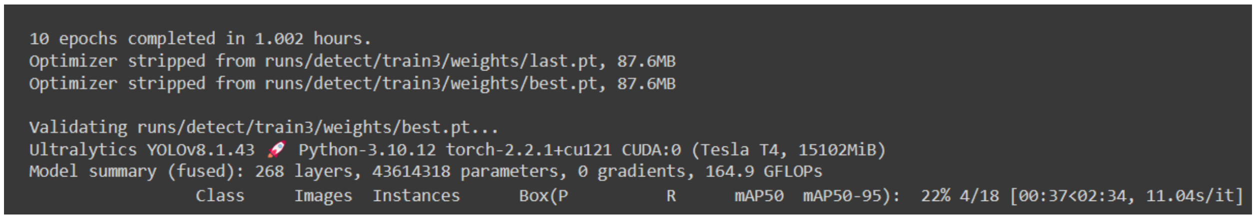
Next, we proceed to set up the required dataset.



```
!yolo train model = yolov8l.pt data = VisDrone.yaml epochs=100
#from ultralytics import YOLO

# Load a model
#model = YOLO('yolov8n.pt')  # load a pretrained model (recommended for training)

# Train the model
#results = model.train(data='VisDrone.yaml', epochs=100, imgsz=640)
```

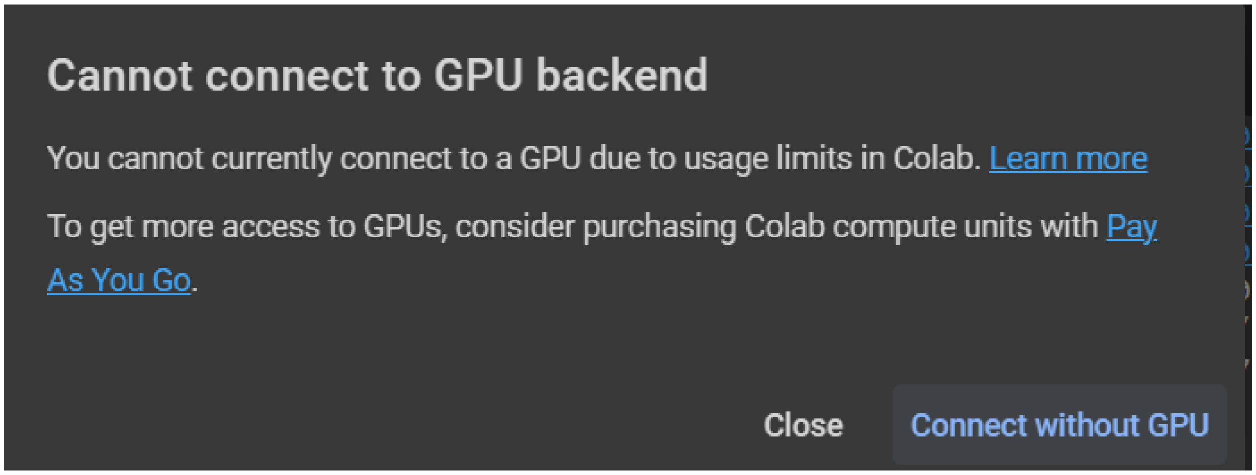
```
Downloading https://github.com/ultralytics/assets/releases/download/v8.1.0/yolov8l.pt to 'yolov8l.pt'...
100% 83.7M/83.7M [00:00<00:00, 282MB/s]
Ultralytics YOLOv8.1.43 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
engine/trainer: task=detect, mode=train, model=yolov8l.pt, data=VisDrone.yaml, epochs=100, time=None, patience=100, batch=16, imgsz=640, save=True, save_period=-1, cache=False,

Dataset 'VisDrone.yaml' images not found ⚠, missing path '/content/datasets/VisDrone/VisDrone2019-DET-val/images'
Downloading https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019-DET-val.zip to '/content/datasets/VisDrone/VisDrone2019-DET-val.zip'...
Downloading https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019-DET-test-dev.zip to '/content/datasets/VisDrone/VisDrone2019-DET-test-dev.zip'...
Downloading https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019-DET-test-challenge.zip to '/content/datasets/VisDrone/VisDrone2019-DET-test-challenge.zip'.
Downloading https://github.com/ultralytics/yolov5/releases/download/v1.0/VisDrone2019-DET-train.zip to '/content/datasets/VisDrone/VisDrone2019-DET-train.zip'...
Converting /content/datasets/VisDrone/VisDrone2019-DET-train: 6471it [00:40, 159.06it/s]
Converting /content/datasets/VisDrone/VisDrone2019-DET-val: 548it [00:05, 104.34it/s]
Converting /content/datasets/VisDrone/VisDrone2019-DET-test-dev: 1610it [00:08, 179.31it/s]
Dataset download success ✅ (87.2s), saved to /content/datasets
```

Training the Model in Google Colab The number of chosen epochs is very high and time-consuming, so I reduced it by a factor of ten. Finally, the response we received indicates that the model has been trained:



```
10 epochs completed in 1.002 hours.
Optimizer stripped from runs/detect/train3/weights/last.pt, 87.6MB
Optimizer stripped from runs/detect/train3/weights/best.pt, 87.6MB

Validating runs/detect/train3/weights/best.pt...
Ultralytics YOLOv8.1.43 🚀 Python-3.10.12 torch-2.2.1+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 268 layers, 43614318 parameters, 0 gradients, 164.9 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95):  22% 4/18 [00:37<02:34, 11.04s/it]
```

Unfortunately, due to extensive attempts to train, the free quota eventually ran out.



Cannot connect to GPU backend

You cannot currently connect to a GPU due to usage limits in Colab. Learn more

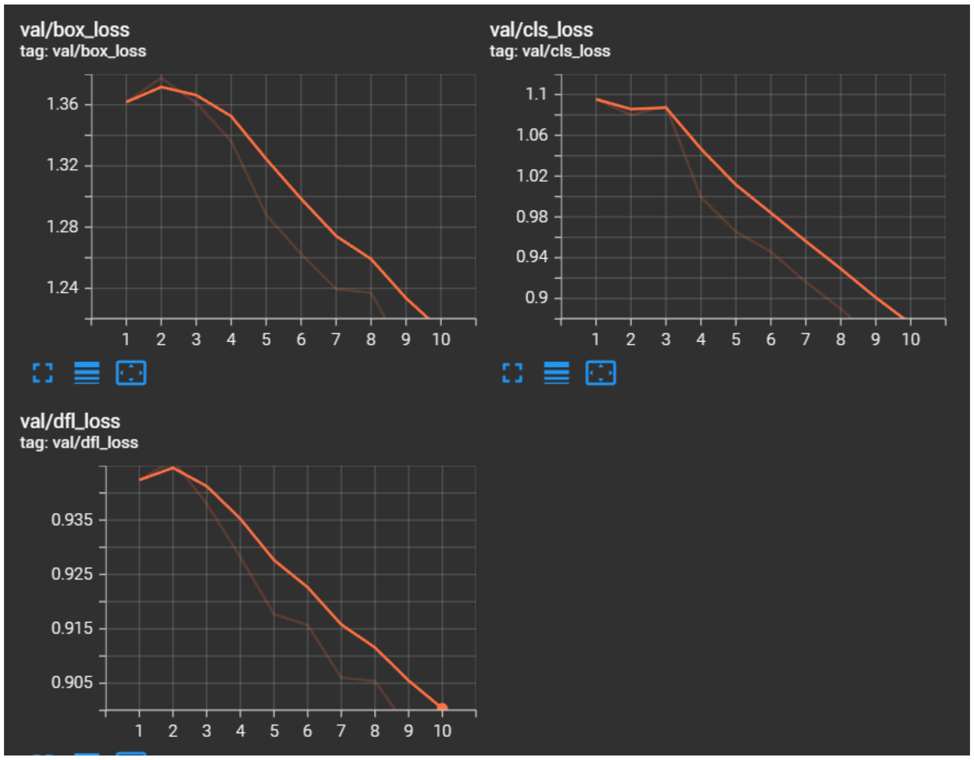
To get more access to GPUs, consider purchasing Colab compute units with Pay As You Go.

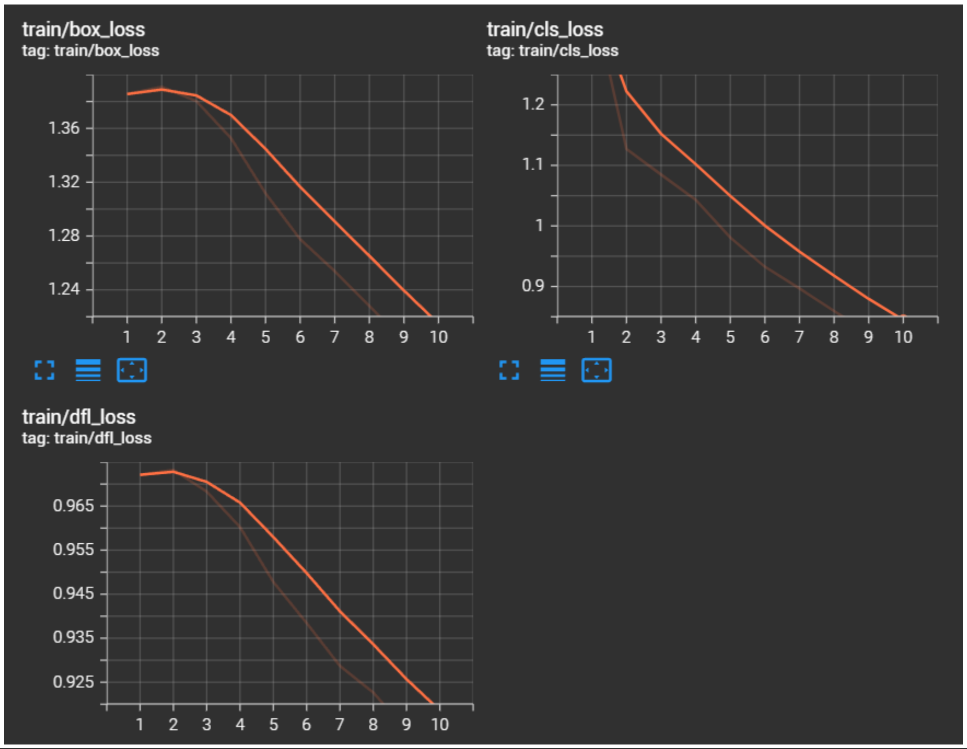
As a result, we have to perform all the steps again on another account to continue.

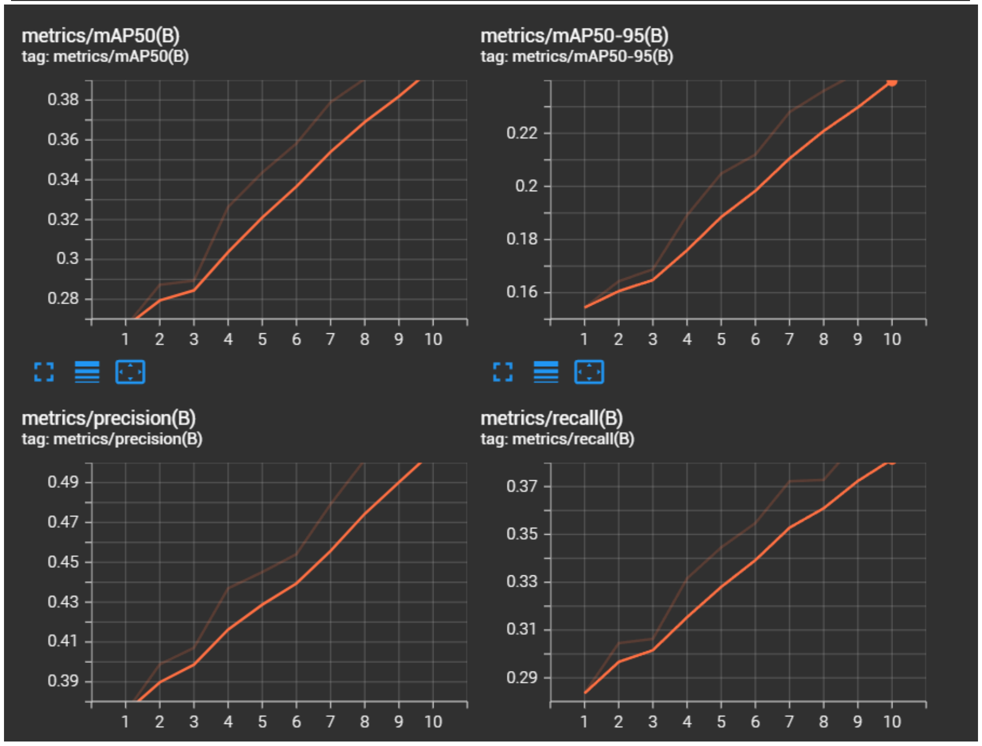
Displaying in TensorBoard

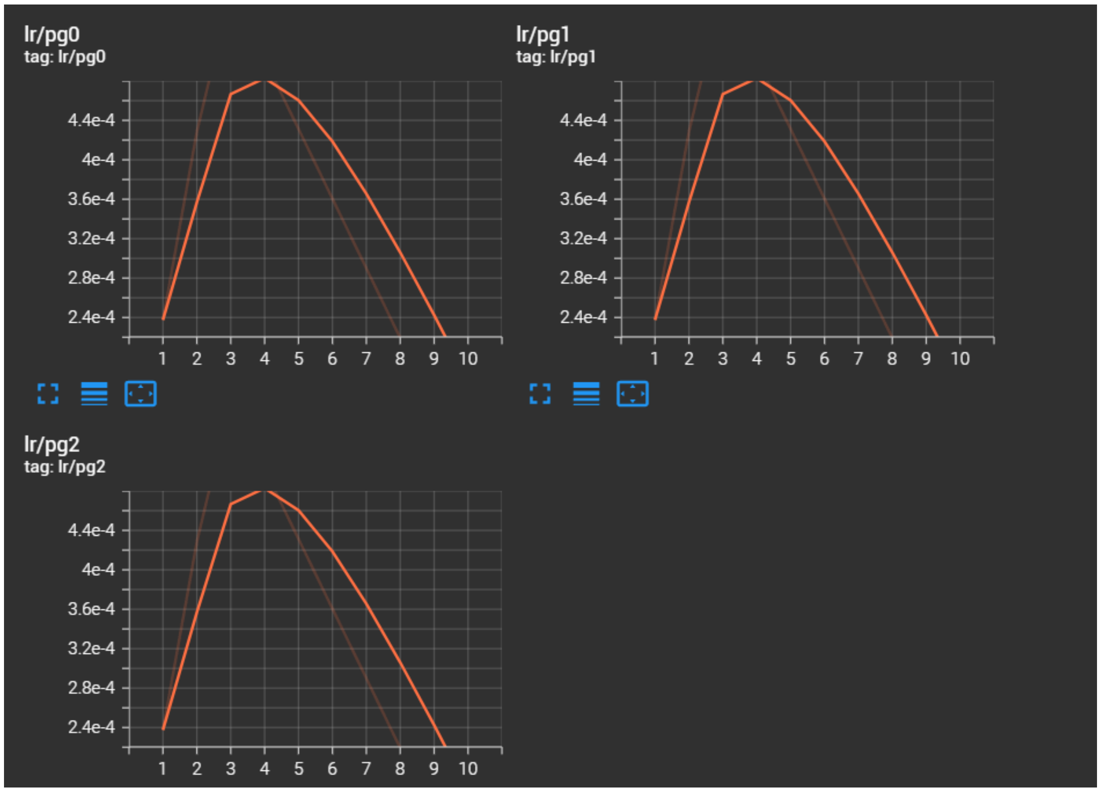
Model Prediction Analysis Well, the model is fully trained now, and we are ready to predict. All we need is an aerial video to use as a sample for prediction. Using the following command, we can perform the prediction:

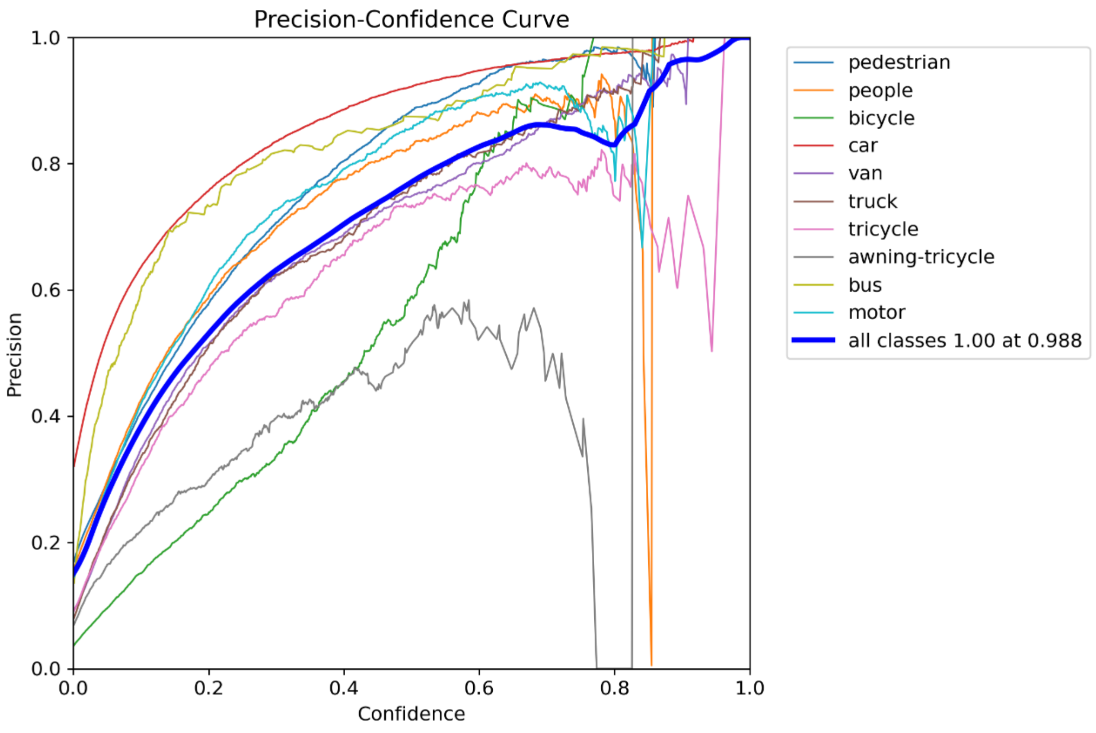
As shown, the prediction has started, and given the large number of video frames, it will take a long time to process each frame and perform the prediction.

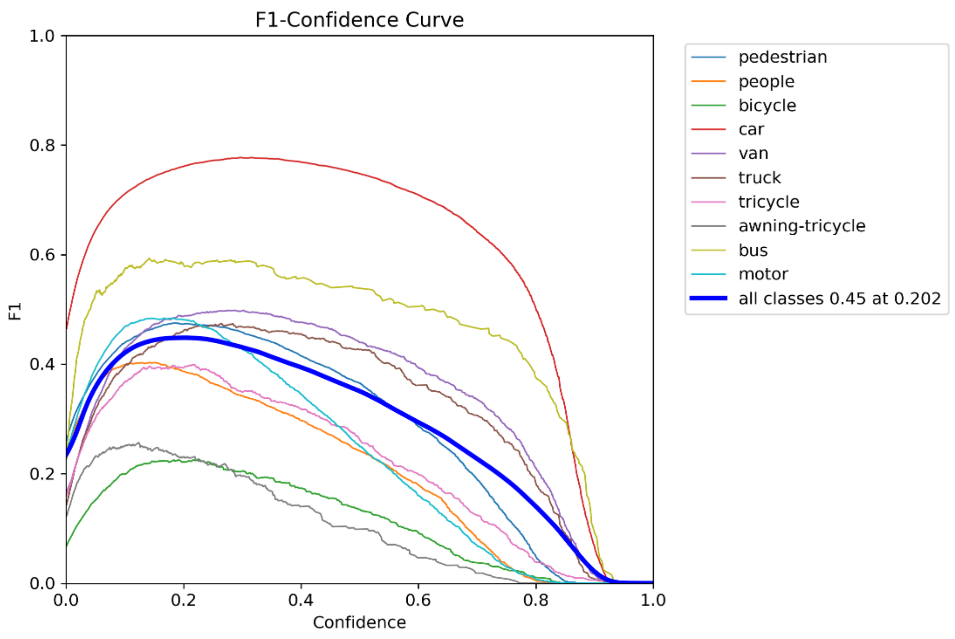


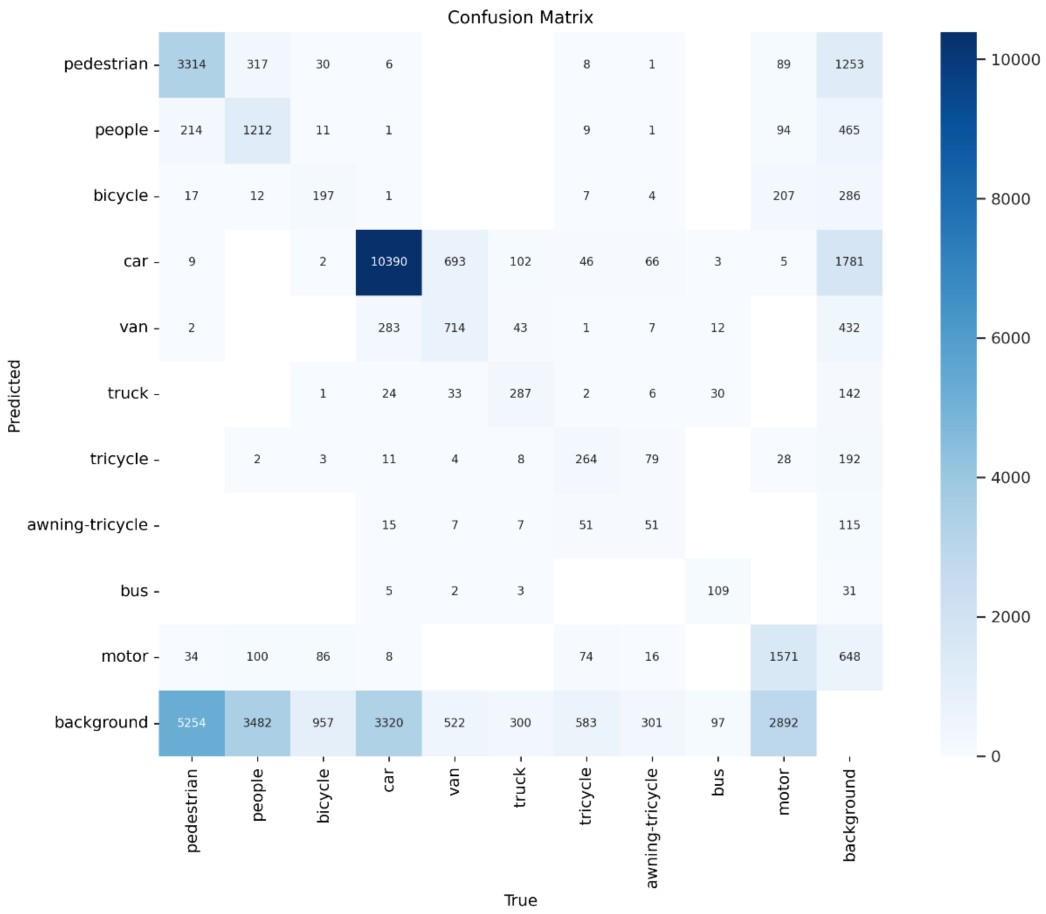

train/box_loss
tag: train/box_loss
1.36
1.32
1.28
1.24
1 2 3 4 5 6 7 8 9 10
train/cls_loss
tag: train/cls_loss
1.2
1.1
1
0.9
1 2 3 4 5 6 7 8 9 10
train/dfl_loss
tag: train/dfl_loss
0.965
0.955
0.945
0.935
0.925
1 2 3 4 5 6 7 8 9 10


metrics/mAP50(B)
tag: metrics/mAP50(B)
0.38
0.36
0.34
0.32
0.3
0.28
1 2 3 4 5 6 7 8 9 10
metrics/mAP50-95(B)
tag: metrics/mAP50-95(B)
0.22
0.2
0.18
0.16
1 2 3 4 5 6 7 8 9 10
metrics/precision(B)
tag: metrics/precision(B)
0.49
0.47
0.45
0.43
0.41
0.39
1 2 3 4 5 6 7 8 9 10
metrics/recall(B)
tag: metrics/recall(B)
0.37
0.35
0.33
0.31
0.29
1 2 3 4 5 6 7 8 9 10

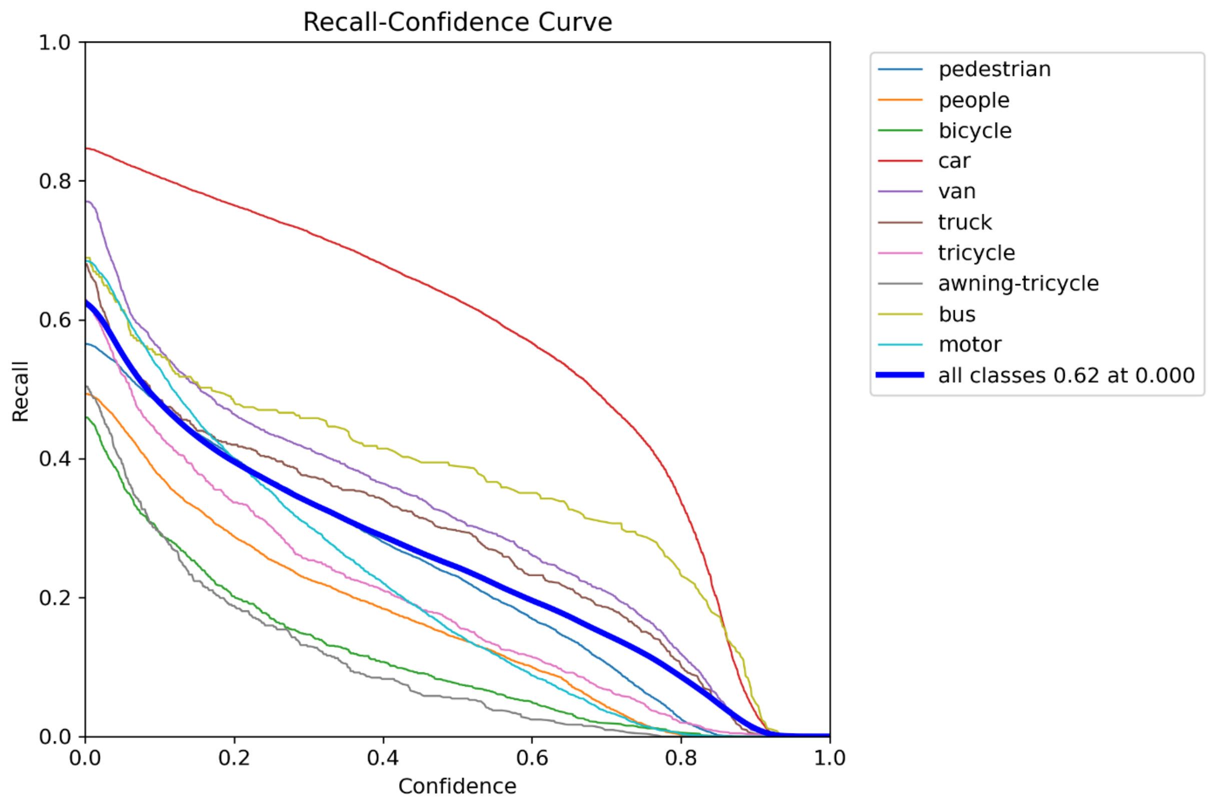

lr/pg0
tag: lr/pg0
lr/pg1
tag: lr/pg1
lr/pg2
tag: lr/pg2
4.4e-4
4e-4
3.6e-4
3.2e-4
2.8e-4
2.4e-4
1 2 3 4 5 6 7 8 9 10

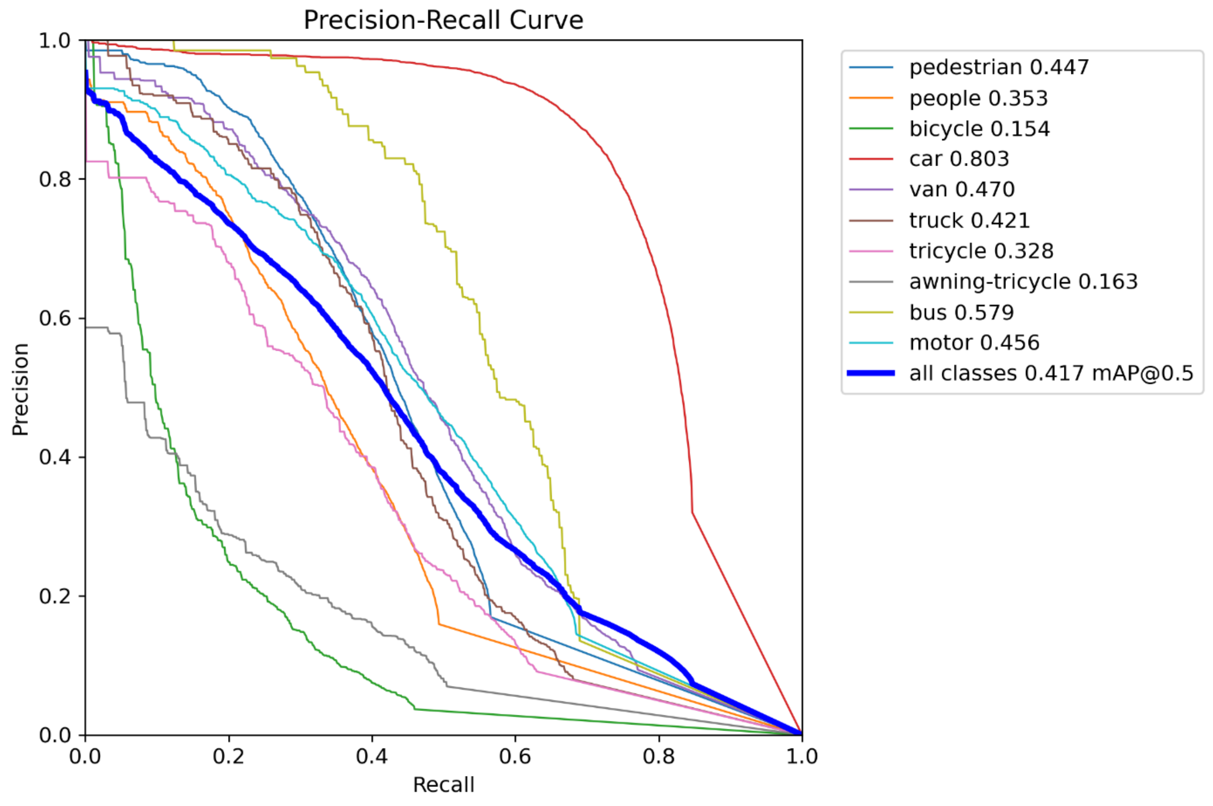

Precision-Confidence Curve
Precision
Confidence
pedestrian
people
bicycle
car
van
truck
tricycle
awning-tricycle
bus
motor
all classes 1.00 at 0.988

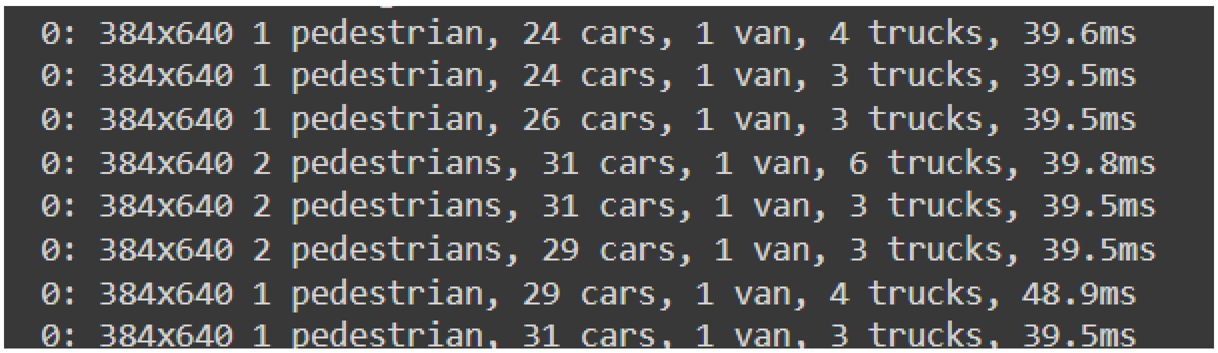




Predicted \ True	pedestrian	people	bicycle	car	van	truck	tricycle	awning-tricycle	bus	motor	background
pedestrian	3314	317	30	6			8	1		89	1253
people	214	1212	11	1			9	1		94	465
bicycle	17	12	197	1			7	4		207	286
car	9		2	10390	693	102	46	66	3	5	1781
van	2			283	714	43	1	7	12		432
truck			1	24	33	287	2	6	30		142
tricycle		2	3	11	4	8	264	79		28	192
awning-tricycle				15	7	7	51	51			115
bus				5	2	3			109		31
motor	34	100	86	8			74	16		1571	648
background	5254	3482	957	3320	522	300	583	301	97	2892	

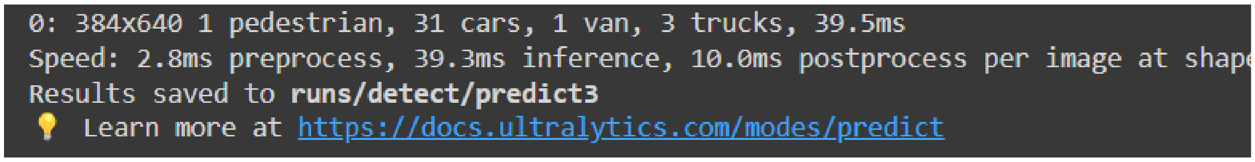

Recall-Confidence Curve
Recall
Confidence
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
pedestrian
people
bicycle
car
van
truck
tricycle
awning-tricycle
bus
motor
all classes 0.62 at 0.000


Precision-Recall Curve
Precision
Recall
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
pedestrian 0.447
people 0.353
bicycle 0.154
car 0.803
van 0.470
truck 0.421
tricycle 0.328
awning-tricycle 0.163
bus 0.579
motor 0.456
all classes 0.417 mAP@0.5



```
0: 384x640 1 pedestrian, 24 cars, 1 van, 4 trucks, 39.6ms
0: 384x640 1 pedestrian, 24 cars, 1 van, 3 trucks, 39.5ms
0: 384x640 1 pedestrian, 26 cars, 1 van, 3 trucks, 39.5ms
0: 384x640 2 pedestrians, 31 cars, 1 van, 6 trucks, 39.8ms
0: 384x640 2 pedestrians, 31 cars, 1 van, 3 trucks, 39.5ms
0: 384x640 2 pedestrians, 29 cars, 1 van, 3 trucks, 39.5ms
0: 384x640 1 pedestrian, 29 cars, 1 van, 4 trucks, 48.9ms
0: 384x640 1 pedestrian, 31 cars, 1 van, 3 trucks, 39.5ms
```

Finally, the prediction result is obtained, and it is observed that the prediction has been done very well, indicating that our model has been well-trained for object detection.



```
0: 384x640 1 pedestrian, 31 cars, 1 van, 3 trucks, 39.5ms
Speed: 2.8ms preprocess, 39.3ms inference, 10.0ms postprocess per image at shape
Results saved to runs/detect/predict3
💡 Learn more at https://docs.ultralytics.com/modes/predict
```

Here is part of the image obtained as a prediction result: