School of Mathematics Computer Science and Engineering

Data Science and Analytics

(COMM022AZ2024/5)

Machine Learning

LIVERPOOL HOPE UNIVERSITY

OBJECT DETECTION USING YOLOV10 MODEL ON A LIVE WEB CAMERA

SHANAWAZ MEMON (24012311)

MSc Data Science

Table of Contents

Abstract

This report presents the development and implementation of a YOLOv10 object detection model aimed at enhancing real-time object recognition capabilities. Object detection plays a critical role in various applications, from autonomous vehicles to surveillance systems. In this project, the YOLOv10 architecture was selected due to its superior balance between speed and accuracy, making it suitable for real-time applications.

The dataset was curated using Roboflow, encompassing diverse images that were meticulously annotated to facilitate effective training. The implementation utilized several key libraries, including Ultralytics for model training and OpenCV for video processing and real-time object detection.

Extensive evaluations were conducted, utilizing metrics such as precision, recall, and mean average precision (mAP) to assess model performance. Results showed promising accuracy with a high degree of reliability in bounding box predictions on sample images and in live webcam applications.

The findings emphasize the potential of YOLOv10 for practical applications and outline certain limitations encountered during the model's development. Future enhancements will focus on refining classification accuracy and exploring advanced architectures, indicating avenues for subsequent research. This project demonstrates the critical role of modern object detection models in advancing intelligent systems.

Introduction

1.1. Overview of Object Detection

Object detection is a pivotal technology in the field of computer vision that empowers machines to identify and locate objects within images or video streams. Imagine having a system that mimics human perception, accurately recognizing items like cars, pedestrians, or animals in real-time. This capability not only enhances automated surveillance, autonomous vehicles, and security systems, but also plays a crucial role in industries like healthcare, robotics, and retail. By combining deep learning with imaging techniques, object detection has transformed the way we interact with technology, making it smarter and more intuitive.

1.2. Project Goals and Objectives

1. 🆕 Use YOLOv10 which is the fastest model in the YOLO family.
2. 📷 Train and test the model on a sample image.
3. 📦 Use RoboFlow for easy image labeling and annotation.
4. 📊 Fine-tune the YOLOv10 model in Google Colab.
5. 🎥 Run live inference on webcam footage to test the model.
6. 🚀 YOLOv10 offers faster inference speed and improved accuracy.
7. 📈 Visualising the results.

1.3. Why YOLOv10?

YOLOv10 demonstrates superior inference speed and accuracy compared to previous versions, making it ideal for real-time applications.

The decision to utilize the YOLOv10 architecture stems from its impressive balance between performance and speed. Unlike traditional object detection models that require multiple passes through the image, YOLO—"You Only Look Once"—processes images in a single pass, making it exceptionally fast. YOLOv10, in particular, integrates state-of-the-art techniques that have enhanced its accuracy and efficiency even further.

Its ability to simultaneously detect and classify multiple objects in a single frame is vital for applications requiring real-time feedback. For instance, in a live surveillance scenario, immediate recognition of events can be crucial. Additionally, YOLOv10 supports a range of object detection tasks, adapting seamlessly to various environments and conditions. This versatility, combined with its ease of use, makes it the ideal choice for our project, ensuring that we leverage the latest innovations in the field while achieving reliable performance for practical applications.

1.4. Applications

Here are some real-world applications of the YOLO (You Only Look Once) object detection model across various domains:

1. Autonomous Vehicles

YOLO plays a crucial role in the development of autonomous vehicles by enabling real-time recognition of other vehicles, pedestrians, traffic signs, and obstacles. This capability is essential for safe navigation in dynamic environments, helping cars make split-second decisions to avoid collisions.

2. Surveillance and Security
In security applications, YOLO is utilized for video surveillance systems that need to monitor public areas or properties. It can detect suspicious activities, recognize faces, and identify lost objects, enhancing security measures by providing timely alerts and real-time video analytics.

3. Retail and Inventory Management
In retail environments, YOLO is employed to manage inventory efficiently. Smart shelves equipped with cameras can detect when items are running low or when products are misplaced. Additionally, it can analyze customer interactions with products, providing insights into shopping behaviors.

4. Industrial Automation
In manufacturing, YOLO is used for quality control by detecting defects in products or components on assembly lines. It helps ensure that only high-quality items proceed through production, reducing waste and improving overall efficiency.

5. Healthcare
YOLO has applications in healthcare, particularly in medical imaging. It can assist radiologists by identifying anomalies in X-rays or MRIs, enabling faster diagnosis of conditions like tumors or fractures. This capability enhances diagnostic accuracy and supports timely treatment decisions.

6. Agriculture
In agriculture, YOLO is used for monitoring crop health and identifying pests or diseases. Drones equipped with cameras can survey fields, detect unhealthy plants, and assess crop yield potential, helping farmers make informed decisions about resource allocation and treatment.

7. Sports Analytics
In sports, YOLO can analyze player movements and strategies during games. By tracking players and the ball in real-time, coaches and analysts can gather insights into performance metrics, helping teams refine strategies and improve training sessions.

2. Background

To understand the significance of our object detection project, it’s essential to delve into the foundational concepts and techniques that drive this technology. The journey of object detection spans various methodologies, each contributing uniquely to the capabilities we see today.

2.1. Object Detection Techniques

Object detection has evolved through numerous techniques over the years, transitioning from classical methods to sophisticated deep-learning approaches. Initially, traditional algorithms relied on handcrafted features and templates to identify objects. Techniques such as Haar cascades and HOG (Histogram of Oriented Gradients) laid the groundwork, but they often struggled with variability in object appearance and environment.

The breakthrough came with the advent of deep learning. Today, convolutional neural networks (CNNs) dominate the field. These models automatically learn hierarchical representations from the data, yielding not just higher accuracy but also the capacity to recognize objects in complex scenes. Frameworks like YOLO, Faster R-CNN, and SSD (Single Shot Detector) exemplify this approach, each with unique strengths in terms of speed and accuracy. While YOLO focuses on real-time performance, others might emphasize higher precision in challenging conditions.

2.2. YOLO Architecture

YOLO, short for "You Only Look Once," revolutionized the way we approach object detection by treating it as a single regression problem, rather than breaking it down into multiple stages. This innovative architecture allows YOLO to process images quickly, yielding bounding boxes and class probabilities in a single evaluation.

At its core, YOLO divides an image into a grid and assigns bounding boxes to each grid cell based on object presence. The model predicts both class probabilities and bounding box coordinates simultaneously, enabling real-time predictions that are essential for applications requiring immediate feedback. This unique efficiency has positioned YOLO as a go-to choice for various real-world scenarios, from autonomous driving to live surveillance systems.

2.3. Key Concepts

Understanding object detection requires familiarity with several fundamental concepts:

A. Bounding Boxes

Bounding boxes are the cornerstone of object localization. They define the position of detected objects in an image by creating rectangular outlines. Each box is typically described by its coordinates—specifying the center point and dimensions—enabling systems to highlight where particular objects reside.

B. Label Annotation

Label annotation is critical for supervised learning in object detection. It involves marking images with specific labels corresponding to the objects within them. This process not only facilitates model training but also ensures that the model learns to distinguish between different classes accurately. Quality annotations are indispensable for achieving high performance in detection tasks.

C. Loss Functions

Loss functions are essential in training machine learning models, guiding the optimization process by quantifying the difference between predicted and actual values. In object detection, several loss functions are employed—addressing localization, classification, and the overall performance of the network. By minimizing these loss values during training, the model learns to improve its accuracy in detecting and classifying objects.

D. Data augmentation

Augmentation significantly enhances the performance of YOLO object detection models. By artificially increasing the size and diversity of the training dataset through techniques like image flipping, rotation, scaling, cropping, color jittering, and mosaic augmentation, the model's robustness and generalization capabilities are improved. These augmentations expose the model to a wider range of object appearances and viewpoints, mitigating overfitting and improving its ability to accurately detect objects in real-world scenarios [2]. Strategies like mosaic augmentation, which combines multiple images, are particularly effective for improving detection accuracy in complex or cluttered scenes [36]. The careful selection and application of data augmentation techniques are crucial for optimizing model performance without introducing unrealistic artifacts that could negatively impact generalization [2]. Moreover, advanced augmentation methods employing generative AI are emerging, promising even greater enhancements in dataset diversity and the ability to synthesize realistic variations of existing data.

3. Implementation

3.1 Dataset Collection and Annotation

For our object detection project utilizing YOLOv10, the first crucial step was the collection and annotation of the dataset. We aimed to create a diverse set of images that accurately represented the objects of interest—whether these were vehicles, animals, or everyday items. We sourced our images from multiple platforms, including public datasets and real-world captures, ensuring that we captured various angles, backgrounds, and lighting conditions.

Annotation was done meticulously, where each object was labeled within the images using bounding boxes. This step was facilitated by tools like Roboflow, which allowed us to efficiently manage our annotations and ensure consistency across the dataset. The quality of our labels directly impacts the model's performance, so we paid close attention to detail, ensuring that each bounding box was tightly fitted around the objects. This process transformed raw images into a structured dataset, ready for training our model.

3.2 Software and Hardware Environment

Creating an effective object detection model requires the right technology stack. For our setup, we chose to work on a machine equipped with a powerful NVIDIA GPU, which significantly sped up the training process due to its ability to handle parallel computations. Our environment consisted of Google Colab, along with Python as the primary programming language.

We utilized virtual environments to manage dependencies, making it easier to ensure that all necessary packages were compatible. This setup allowed us to run large-scale experiments without worrying about system performance degradation. Additionally, we made sure to keep our drivers and libraries updated to leverage the latest optimizations available for deep learning frameworks.

3.3 Libraries and Frameworks

In developing our object detection model, we leaned on several libraries that streamlined the process and enhanced functionality.

A. Ultralytics

Ultralytics YOLOv10 served as the backbone of our object detection pipeline. This library is renowned for its efficiency and ease of use, making it accessible even for those new to deep learning. The framework provided pre-trained models and flexible configuration options, allowing us to fine-tune our model according to the specifics of our dataset. Its comprehensive documentation helped us quickly implement the necessary training routines and inference processes without getting bogged down in technical details.

B. Supervision

Supervision played a pivotal role in our project, offering tools that helped visualize our data and results effectively. This library streamlined the evaluation of our models by automatically generating metrics and visualizations, which made it easier to gauge performance and make necessary adjustments. Its user-friendly interface allowed for smooth integration with our existing workflow, ensuring that we could quickly iterate on changes and understand their impact.

C. OpenCV (cv2)

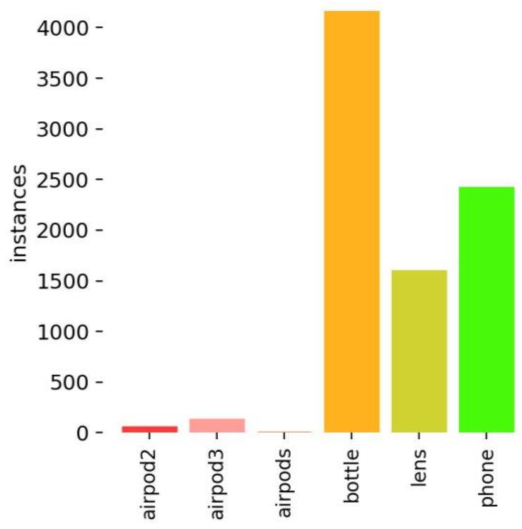
OpenCV, or Open Source Computer Vision Library, was indispensable for image processing tasks throughout our project. We utilized it for preprocessing our data, such as resizing images and applying transformations. Additionally, during the real-time inference phase, OpenCV enabled us to capture video streams from webcams and display the detected objects with bounding boxes in real-time. Its extensive functionalities allowed us to efficiently handle various computer vision tasks, making it a foundational component of our implementation.

D. Model Training

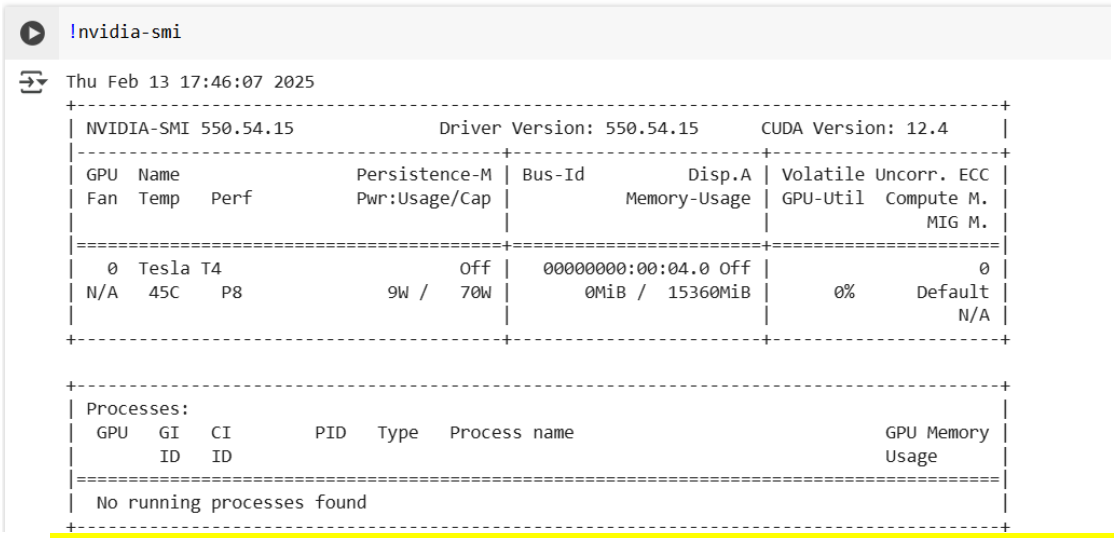
Training our YOLOv10 model involved a series of carefully planned steps. After preparing our annotated dataset, we set the training parameters, including learning rates, batch sizes, and the number of epochs—ensuring that our choices were informed by the specific characteristics of our data. We started with transfer learning, using weights from a pre-trained model, which allowed us to benefit from robust feature extraction capabilities right from the start.

As the training progressed, we monitored performance metrics such as loss and accuracy to determine how well the model was learning. We made periodic adjustments to the training schedule based on these metrics, stopping the training when we observed diminishing returns in accuracy improvements. Ultimately, our goal was to achieve a model that generalized well to unseen images, providing accurate detection results in real-world situations.

Our instance for the dataset to be trained on the model.



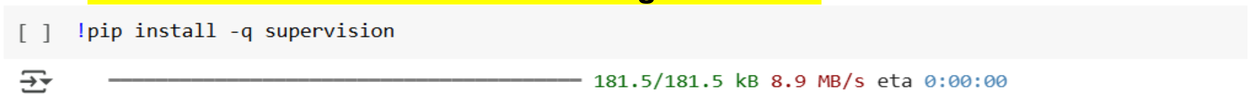
3.4 Final Implementation



```
!nvidia-smi
```

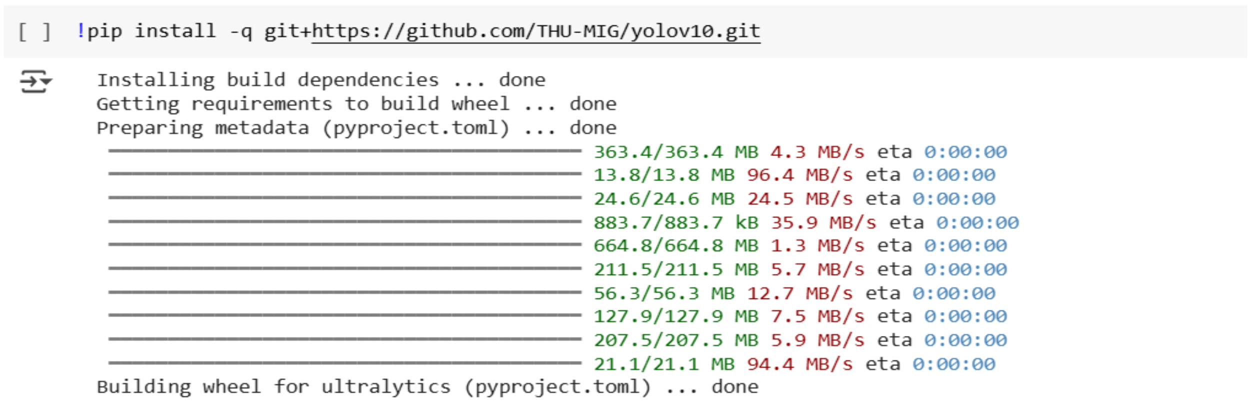
```
Thu Feb 13 17:46:07 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.54.15              Driver Version: 550.54.15      CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  Tesla T4                       Off |   00000000:00:04.0 Off |                    0 |
| N/A   45C    P8              9W /   70W |       0MiB /  15360MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

1. **Using Nvidia GPU runtime from Colab to speed up the training process via parallel computations.**
2. **Importing the github repository on the colab file system. So that we can access the YOLOV10 model and weights from it.**



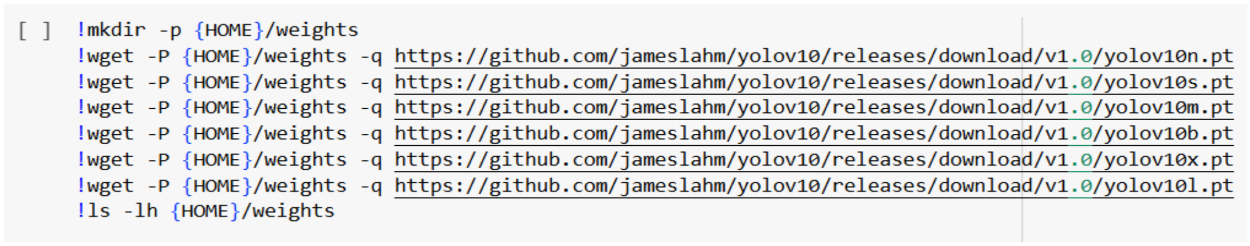
```
[ ]  !pip install -q supervision
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 181.5/181.5 kB 8.9 MB/s eta 0:00:00
```

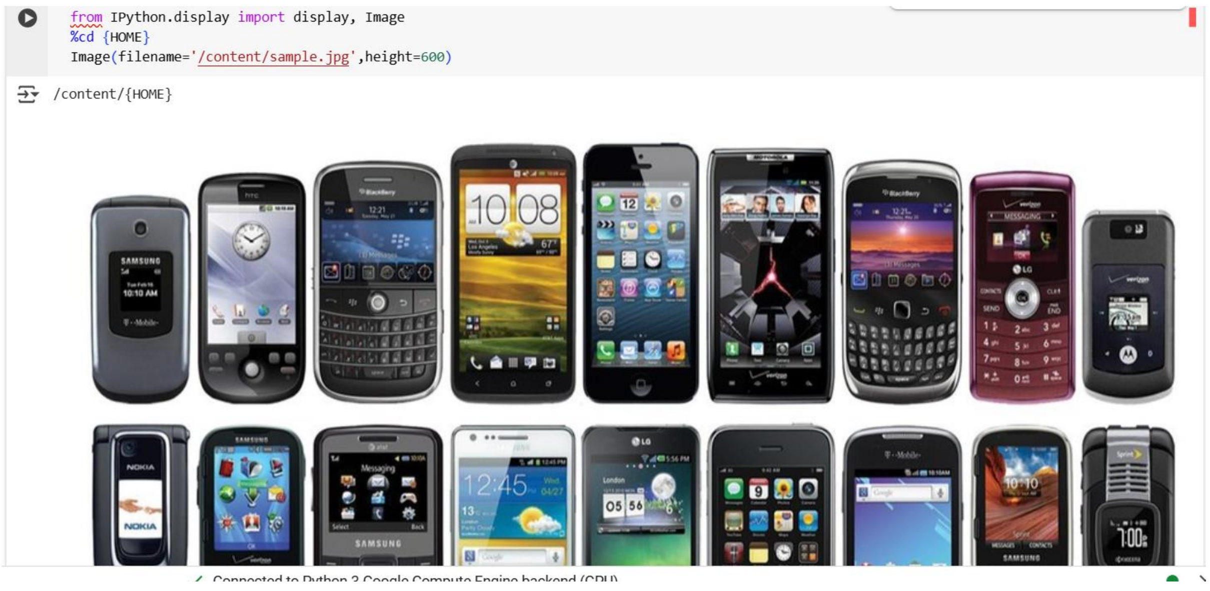


```
[ ]  !pip install -q git+https://github.com/THU-MIG/yolov10.git
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 96.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 24.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 35.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 1.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 12.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 7.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.9 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 94.4 MB/s eta 0:00:00
  Building wheel for ultralytics (pyproject.toml) ... done
```



```
[ ]  !mkdir -p {HOME}/weights
     !wget -P {HOME}/weights -q https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10n.pt
     !wget -P {HOME}/weights -q https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10s.pt
     !wget -P {HOME}/weights -q https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10m.pt
     !wget -P {HOME}/weights -q https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10b.pt
     !wget -P {HOME}/weights -q https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10x.pt
     !wget -P {HOME}/weights -q https://github.com/jameslahm/yolov10/releases/download/v1.0/yolov10l.pt
     !ls -lh {HOME}/weights
```

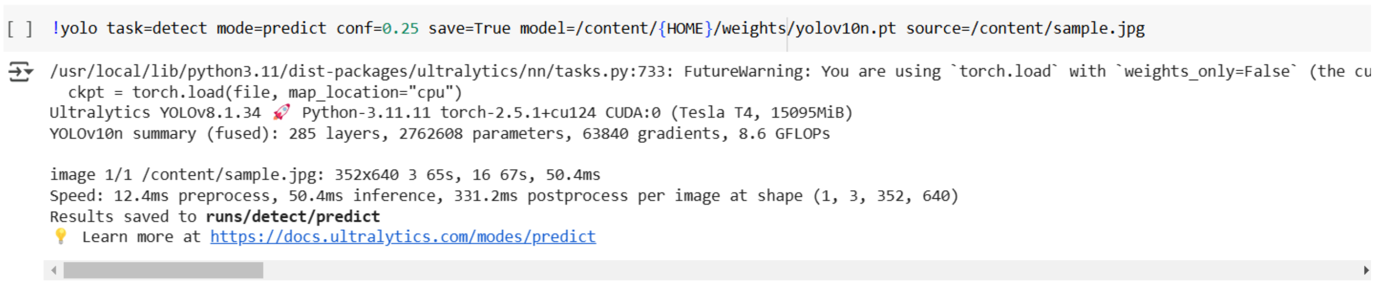


```
from IPython.display import display, Image
%cd {HOME}
Image(filename='/content/sample.jpg',height=600)
```

```
/content/{HOME}
```

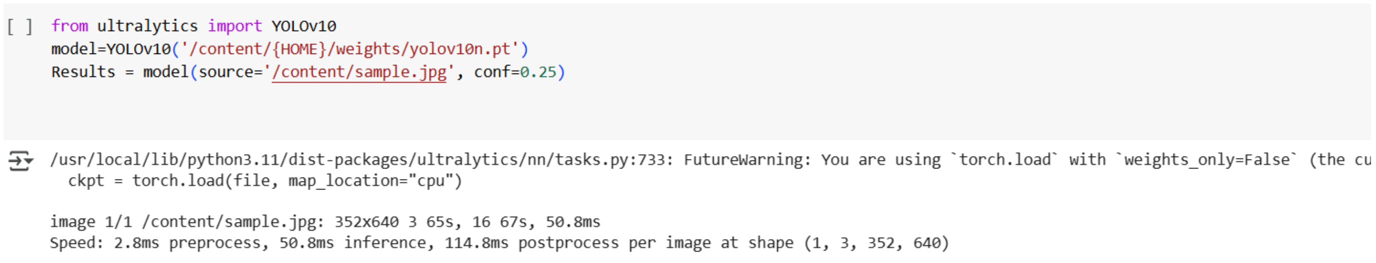
Connected to Python 3 Google Compute Engine backend (CPU)

3. **Uploading and displaying a sample image which will be used for training the model before performing the webcam inference to make sure it works fine.**
4. **Training the yolov10n model on our sample image with a confidence score of 25 or higher prediction.**
5. **Using the Ultralytics library from Python, we create an instance of the model trained and assign it to the variable named 'model'.**
6. **Performing model detection on the sample image.**



```
[ ] !yolo task=detect mode=predict conf=0.25 save=True model=/content/{HOME}/weights/yolov10n.pt source=/content/sample.jpg
```

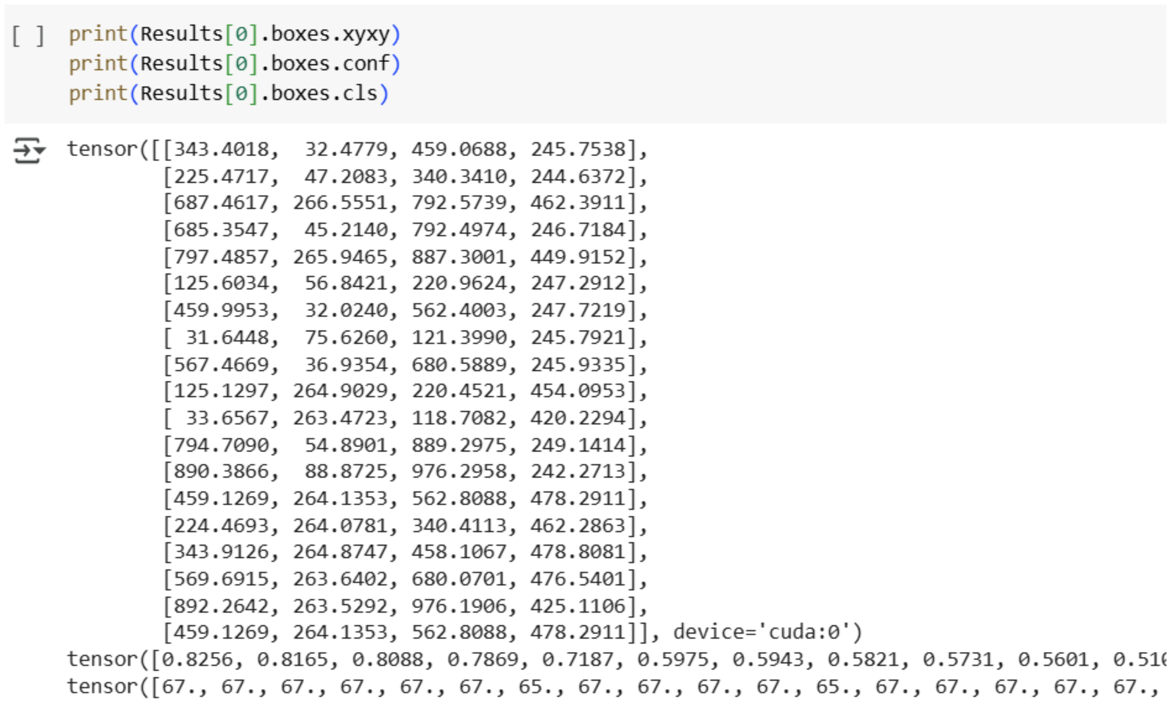
```
/usr/local/lib/python3.11/dist-packages/ultralytics/nn/tasks.py:733: FutureWarning: You are using `torch.load` with `weights_only=False` (the cu
  ckpt = torch.load(file, map_location="cpu")
Ultralytics YOLOv8.1.34 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)
YOLOv10n summary (fused): 285 layers, 2762608 parameters, 63840 gradients, 8.6 GFLOPs

image 1/1 /content/sample.jpg: 352x640 3 65s, 16 67s, 50.4ms
Speed: 12.4ms preprocess, 50.4ms inference, 331.2ms postprocess per image at shape (1, 3, 352, 640)
Results saved to runs/detect/predict
💡 Learn more at https://docs.ultralytics.com/modes/predict
```



```
[ ] from ultralytics import YOLOv10
    model=YOLOv10('/content/{HOME}/weights/yolov10n.pt')
    Results = model(source='/content/sample.jpg', conf=0.25)
```

```
/usr/local/lib/python3.11/dist-packages/ultralytics/nn/tasks.py:733: FutureWarning: You are using `torch.load` with `weights_only=False` (the cu
  ckpt = torch.load(file, map_location="cpu")

image 1/1 /content/sample.jpg: 352x640 3 65s, 16 67s, 50.8ms
Speed: 2.8ms preprocess, 50.8ms inference, 114.8ms postprocess per image at shape (1, 3, 352, 640)
```

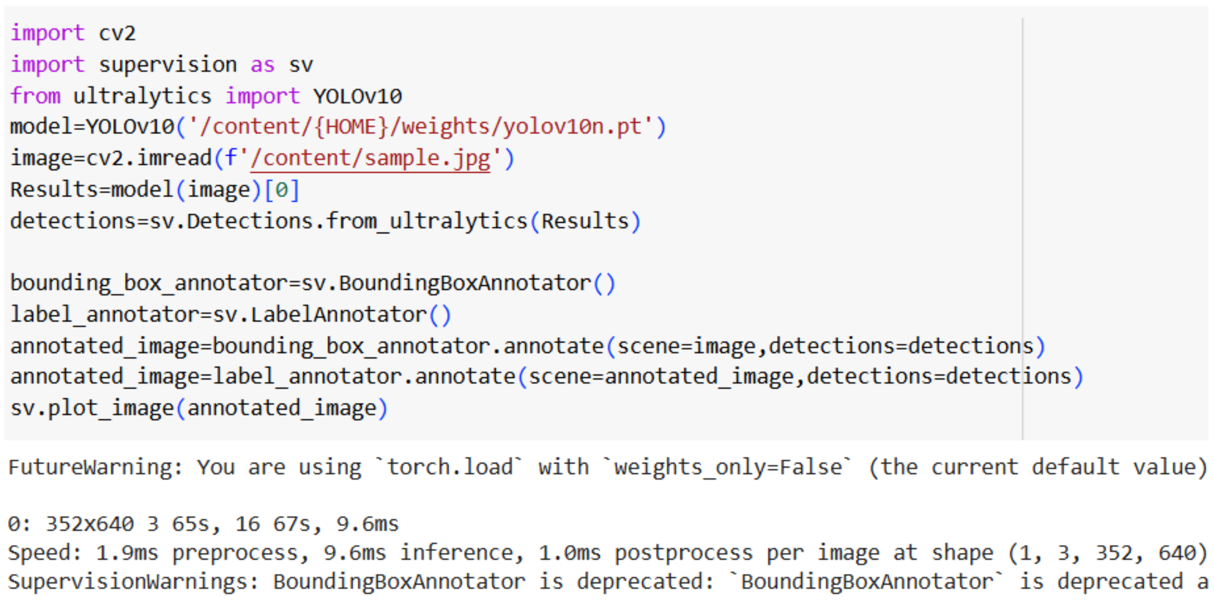
7. **Obtaining the coordinates for the bounding box, Model confidence in the detection, and predicted class of each object.**



```
[ ]  print(Results[0].boxes.xyxy)
     print(Results[0].boxes.conf)
     print(Results[0].boxes.cls)
```

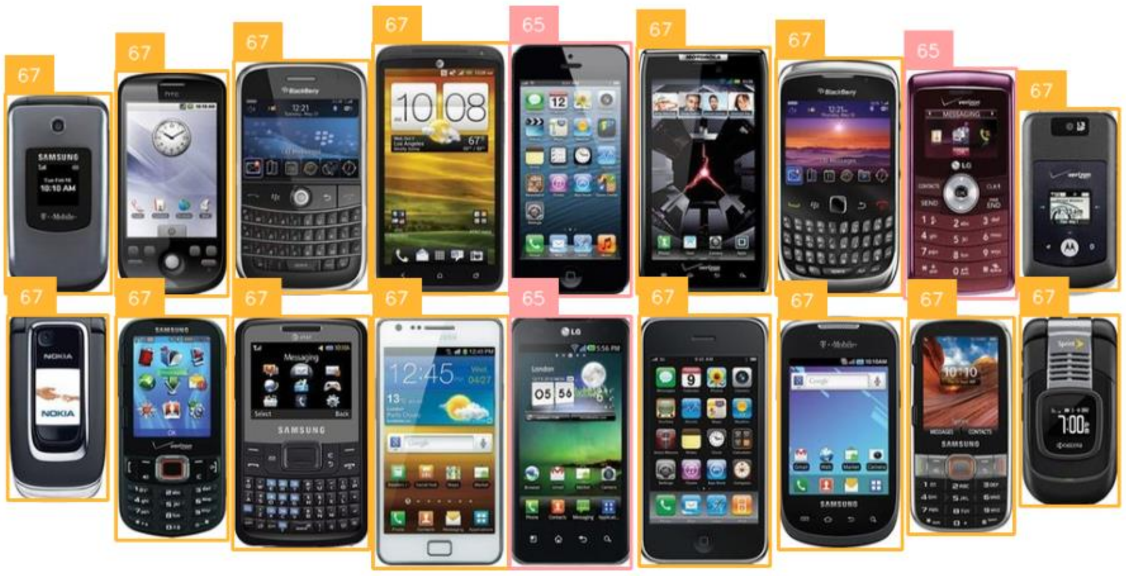
```
tensor([[343.4018,  32.4779, 459.0688, 245.7538],
        [225.4717,  47.2083, 340.3410, 244.6372],
        [687.4617, 266.5551, 792.5739, 462.3911],
        [685.3547,  45.2140, 792.4974, 246.7184],
        [797.4857, 265.9465, 887.3001, 449.9152],
        [125.6034,  56.8421, 220.9624, 247.2912],
        [459.9953,  32.0240, 562.4003, 247.7219],
        [ 31.6448,  75.6260, 121.3990, 245.7921],
        [567.4669,  36.9354, 680.5889, 245.9335],
        [125.1297, 264.9029, 220.4521, 454.0953],
        [ 33.6567, 263.4723, 118.7082, 420.2294],
        [794.7090,  54.8901, 889.2975, 249.1414],
        [890.3866,  88.8725, 976.2958, 242.2713],
        [459.1269, 264.1353, 562.8088, 478.2911],
        [224.4693, 264.0781, 340.4113, 462.2863],
        [343.9126, 264.8747, 458.1067, 478.8081],
        [569.6915, 263.6402, 680.0701, 476.5401],
        [892.2642, 263.5292, 976.1906, 425.1106],
        [459.1269, 264.1353, 562.8088, 478.2911]], device='cuda:0')
tensor([0.8256, 0.8165, 0.8088, 0.7869, 0.7187, 0.5975, 0.5943, 0.5821, 0.5731, 0.5601, 0.51
tensor([67., 67., 67., 67., 67., 67., 65., 67., 67., 67., 67., 65., 67., 67., 67., 67., 67.,
```

8. **Using Open cv to module to manipulate and read data from images, Supervision for visualization and ultralytics to import the yolov10 model.**
9. **Creating bounding_box_annotator and label annotator for creating a bounding box & label annotator around the detected objects.**
10. **Plot the result.**

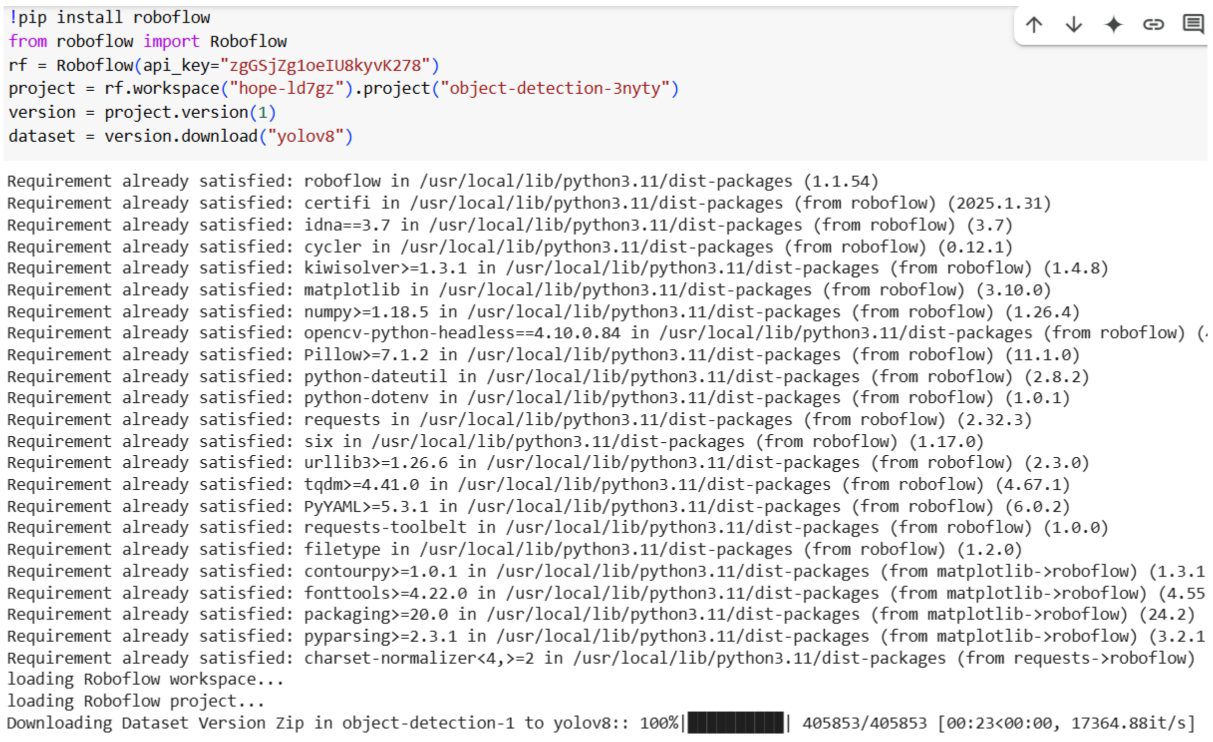


```
import cv2
import supervision as sv
from ultralytics import YOLOv10
model=YOLOv10('/content/{HOME}/weights/yolov10n.pt')
image=cv2.imread(f'/content/sample.jpg')
Results=model(image)[0]
detections=sv.Detections.from_ultralytics(Results)

bounding_box_annotator=sv.BoundingBoxAnnotator()
label_annotator=sv.LabelAnnotator()
annotated_image=bounding_box_annotator.annotate(scene=image,detections=detections)
annotated_image=label_annotator.annotate(scene=annotated_image,detections=detections)
sv.plot_image(annotated_image)
```

```
FutureWarning: You are using `torch.load` with `weights_only=False` (the current default value)

0: 352x640 3 65s, 16 67s, 9.6ms
Speed: 1.9ms preprocess, 9.6ms inference, 1.0ms postprocess per image at shape (1, 3, 352, 640)
SupervisionWarnings: BoundingBoxAnnotator is deprecated: `BoundingBoxAnnotator` is deprecated a
```



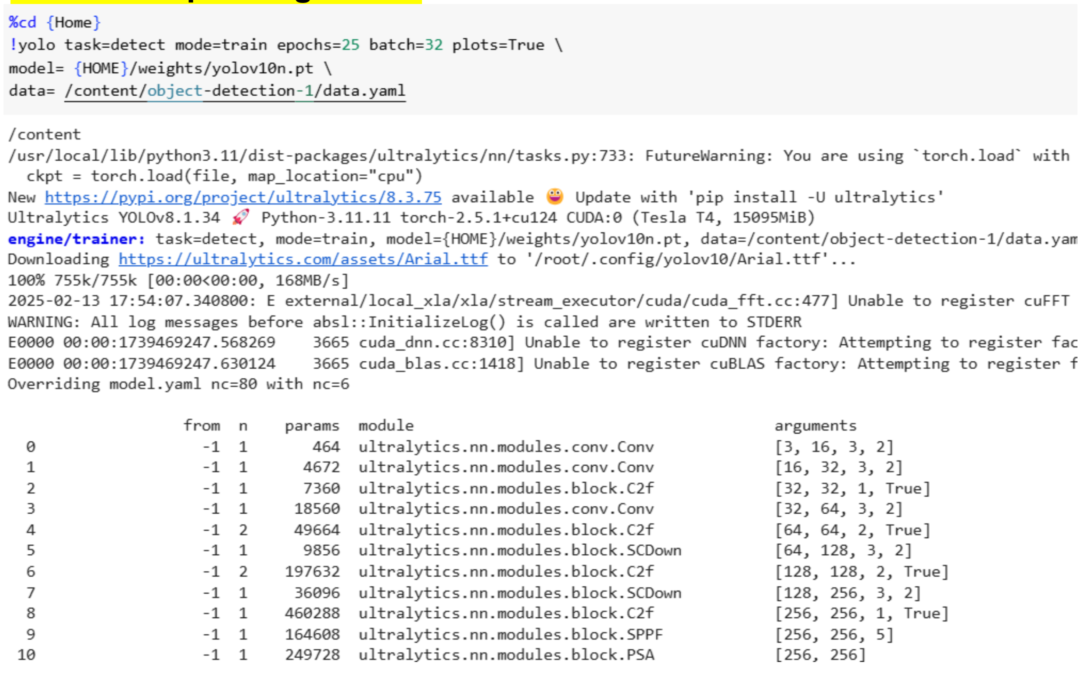
11. **We use roboflow to import the trained dataset using the API key of your account.**
12. **Download the dataset in the yolov8 version.**



```
!pip install roboflow
from roboflow import Roboflow
rf = Roboflow(api_key="zgGSjZg1oeIU8kyvK278")
project = rf.workspace("hope-ld7gz").project("object-detection-3nyty")
version = project.version(1)
dataset = version.download("yolov8")
```

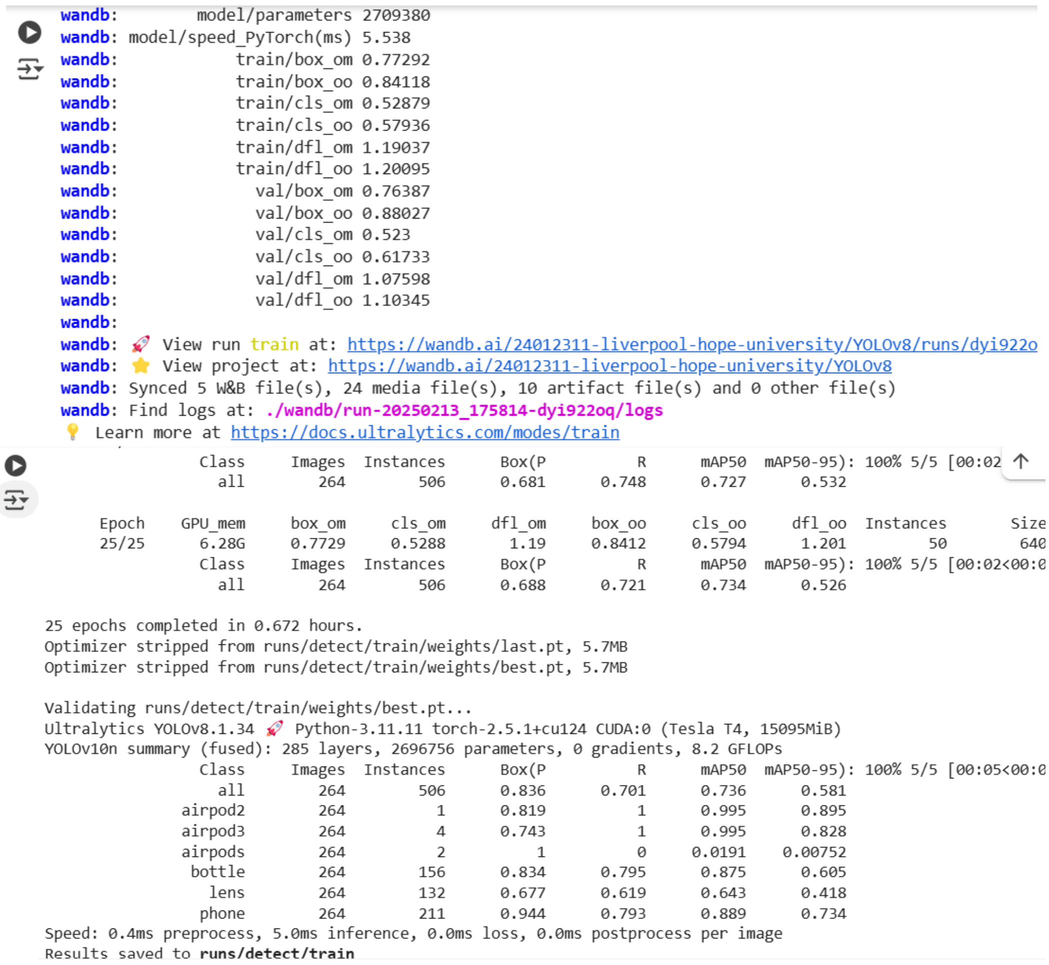
```
Requirement already satisfied: roboflow in /usr/local/lib/python3.11/dist-packages (1.1.54)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from roboflow) (2025.1.31)
Requirement already satisfied: idna==3.7 in /usr/local/lib/python3.11/dist-packages (from roboflow) (3.7)
Requirement already satisfied: cycler in /usr/local/lib/python3.11/dist-packages (from roboflow) (0.12.1)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.4.8)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.11/dist-packages (from roboflow) (3.10.0)
Requirement already satisfied: numpy>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.26.4)
Requirement already satisfied: opencv-python-headless==4.10.0.84 in /usr/local/lib/python3.11/dist-packages (from roboflow) (
Requirement already satisfied: Pillow>=7.1.2 in /usr/local/lib/python3.11/dist-packages (from roboflow) (11.1.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.8.2)
Requirement already satisfied: python-dotenv in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.0.1)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.32.3)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.17.0)
Requirement already satisfied: urllib3>=1.26.6 in /usr/local/lib/python3.11/dist-packages (from roboflow) (2.3.0)
Requirement already satisfied: tqdm>=4.41.0 in /usr/local/lib/python3.11/dist-packages (from roboflow) (4.67.1)
Requirement already satisfied: PyYAML>=5.3.1 in /usr/local/lib/python3.11/dist-packages (from roboflow) (6.0.2)
Requirement already satisfied: requests-toolbelt in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.0.0)
Requirement already satisfied: filetype in /usr/local/lib/python3.11/dist-packages (from roboflow) (1.2.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (1.3.1
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (4.55
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib->roboflow) (3.2.1
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->roboflow)
loading Roboflow workspace...
loading Roboflow project...
Downloading Dataset Version Zip in object-detection-1 to yolov8:: 100%|██████████| 405853/405853 [00:23<00:00, 17364.88it/s]
```

13. **Now we train the dataset consisting of airpods, bottles, phones, and lenses on yolov10n model.**
14. **Data.yaml file likely describes the location of the training images and their corresponding labels.**



```
%cd {Home}
!yolo task=detect mode=train epochs=25 batch=32 plots=True \
model= {HOME}/weights/yolov10n.pt \
data= /content/object-detection-1/data.yaml
```

```
/content
/usr/local/lib/python3.11/dist-packages/ultralytics/nn/tasks.py:733: FutureWarning: You are using `torch.load` with
  ckpt = torch.load(file, map_location="cpu")
New https://pypi.org/project/ultralytics/8.3.75 available 😃 Update with 'pip install -U ultralytics'
Ultralytics YOLOv8.1.34 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)
engine/trainer: task=detect, mode=train, model={HOME}/weights/yolov10n.pt, data=/content/object-detection-1/data.yam
Downloading https://ultralytics.com/assets/Arial.ttf to '/root/.config/yolov10/Arial.ttf'...
100% 755k/755k [00:00<00:00, 168MB/s]
2025-02-13 17:54:07.340800: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:477] Unable to register cuFFT
WARNING: All log messages before absl::InitializeLog() is called are written to STDERR
E0000 00:00:1739469247.568269    3665 cuda_dnn.cc:8310] Unable to register cuDNN factory: Attempting to register fac
E0000 00:00:1739469247.630124    3665 cuda_blas.cc:1418] Unable to register cuBLAS factory: Attempting to register f
Overriding model.yaml nc=80 with nc=6

                   from  n    params  module                                       arguments
  0                  -1  1       464  ultralytics.nn.modules.conv.Conv             [3, 16, 3, 2]
  1                  -1  1      4672  ultralytics.nn.modules.conv.Conv             [16, 32, 3, 2]
  2                  -1  1      7360  ultralytics.nn.modules.block.C2f             [32, 32, 1, True]
  3                  -1  1     18560  ultralytics.nn.modules.conv.Conv             [32, 64, 3, 2]
  4                  -1  2     49664  ultralytics.nn.modules.block.C2f             [64, 64, 2, True]
  5                  -1  1      9856  ultralytics.nn.modules.block.SCDown          [64, 128, 3, 2]
  6                  -1  2    197632  ultralytics.nn.modules.block.C2f             [128, 128, 2, True]
  7                  -1  1     36096  ultralytics.nn.modules.block.SCDown          [128, 256, 3, 2]
  8                  -1  1    460288  ultralytics.nn.modules.block.C2f             [256, 256, 1, True]
  9                  -1  1    164608  ultralytics.nn.modules.block.SPPF            [256, 256, 5]
 10                  -1  1    249728  ultralytics.nn.modules.block.PSA             [256, 256]
```

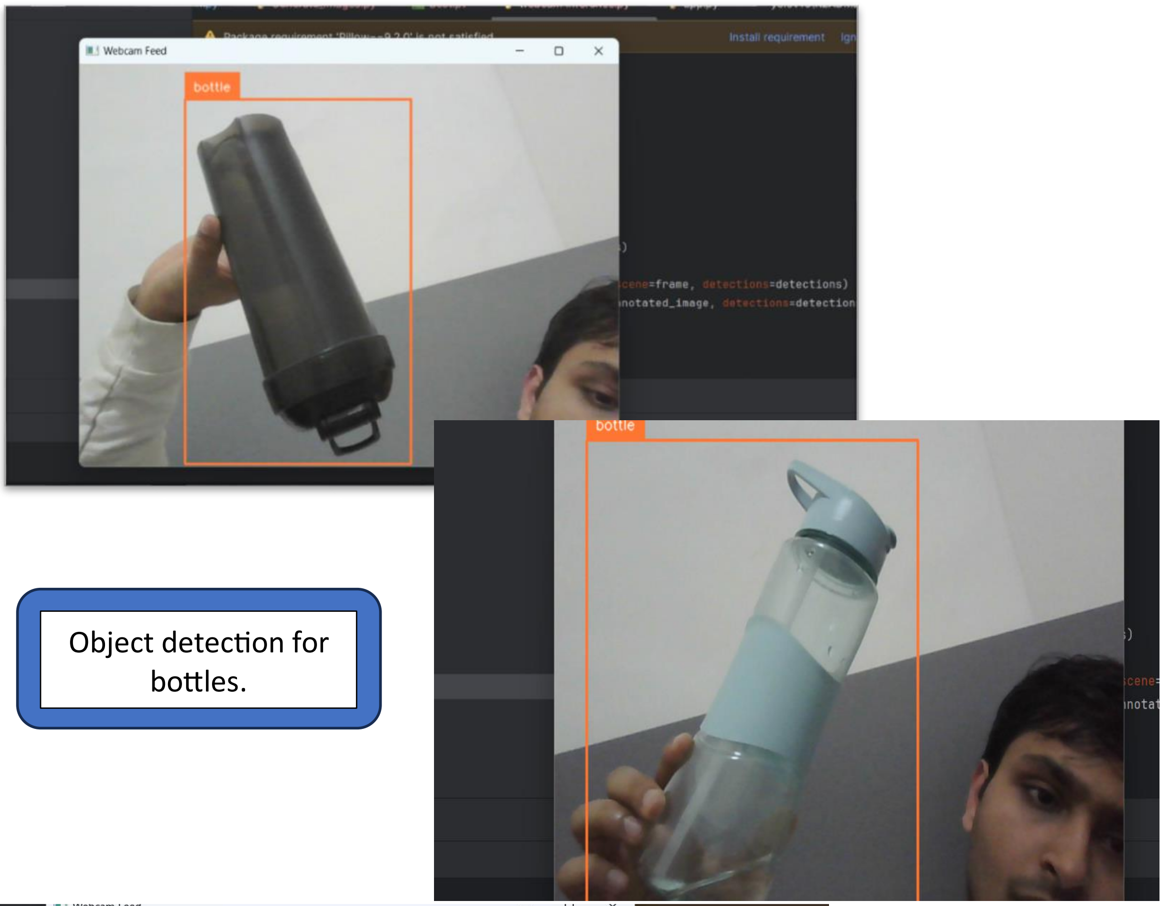


```
wandb:       model/parameters 2709380
wandb: model/speed_PyTorch(ms) 5.538
wandb:           train/box_om 0.77292
wandb:           train/box_oo 0.84118
wandb:           train/cls_om 0.52879
wandb:           train/cls_oo 0.57936
wandb:           train/dfl_om 1.19037
wandb:           train/dfl_oo 1.20095
wandb:             val/box_om 0.76387
wandb:             val/box_oo 0.88027
wandb:             val/cls_om 0.523
wandb:             val/cls_oo 0.61733
wandb:             val/dfl_om 1.07598
wandb:             val/dfl_oo 1.10345
wandb: 
wandb: 🚀 View run train at: https://wandb.ai/24012311-liverpool-hope-university/YOLOv8/runs/dyi922o
wandb: ⭐️ View project at: https://wandb.ai/24012311-liverpool-hope-university/YOLOv8
wandb: Synced 5 W&B file(s), 24 media file(s), 10 artifact file(s) and 0 other file(s)
wandb: Find logs at: ./wandb/run-20250213_175814-dyi922oq/logs
 💡 Learn more at https://docs.ultralytics.com/modes/train
```

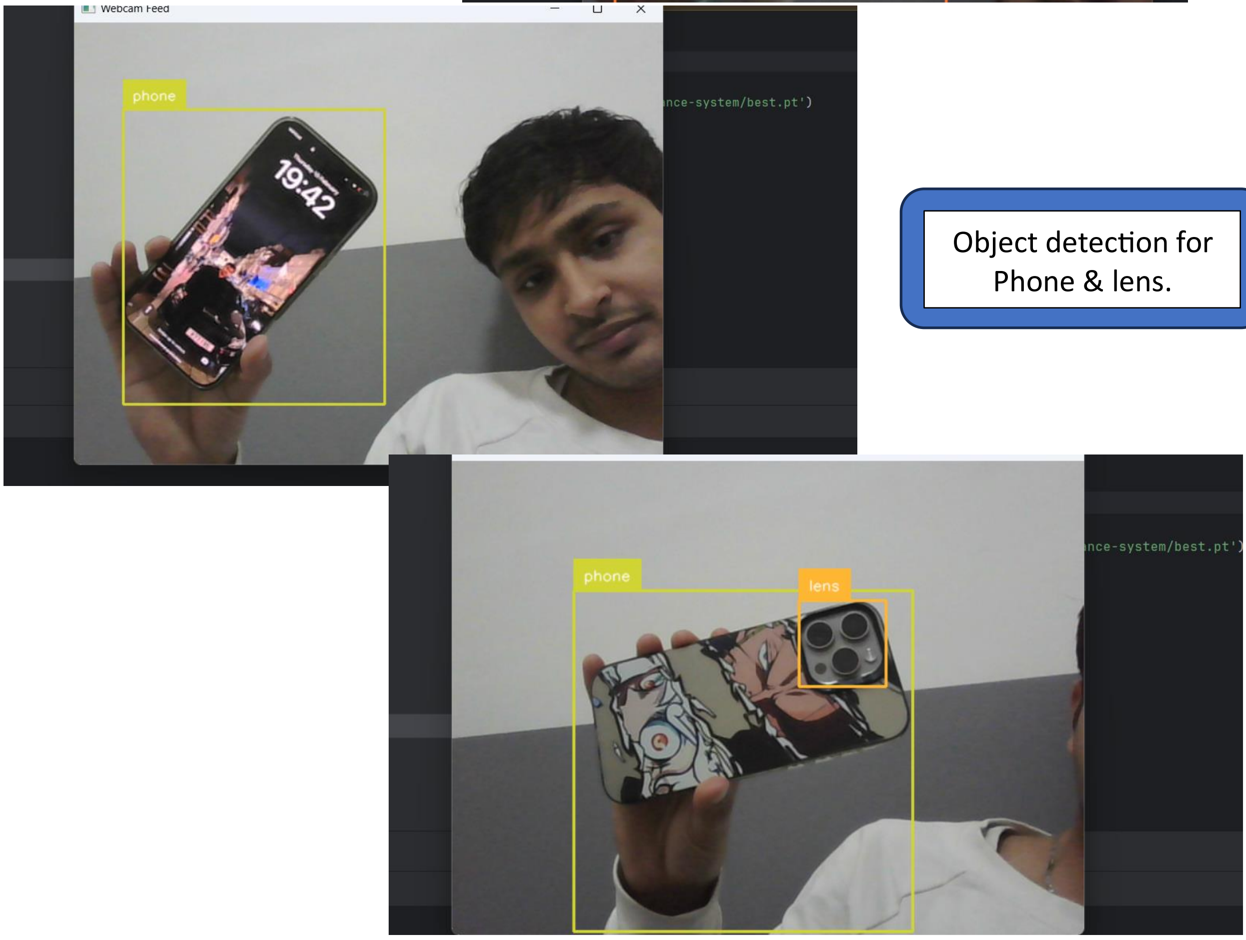
```
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02
                   all        264        506      0.681      0.748      0.727      0.532

      Epoch    GPU_mem     box_om     cls_om     dfl_om     box_oo     cls_oo     dfl_oo  Instances       Size
      25/25      6.28G     0.7729     0.5288       1.19     0.8412     0.5794      1.201         50        640
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:02<00:0
                   all        264        506      0.688      0.721      0.734      0.526

25 epochs completed in 0.672 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 5.7MB
Optimizer stripped from runs/detect/train/weights/best.pt, 5.7MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.1.34 🚀 Python-3.11.11 torch-2.5.1+cu124 CUDA:0 (Tesla T4, 15095MiB)
YOLOv10n summary (fused): 285 layers, 2696756 parameters, 0 gradients, 8.2 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% 5/5 [00:05<00:0
                   all        264        506      0.836      0.701      0.736      0.581
               airpod2        264          1      0.819          1      0.995      0.895
               airpod3        264          4      0.743          1      0.995      0.828
               airpods        264          2          1          0     0.0191    0.00752
                bottle        264        156      0.834      0.795      0.875      0.605
                  lens        264        132      0.677      0.619      0.643      0.418
                 phone        264        211      0.944      0.793      0.889      0.734
Speed: 0.4ms preprocess, 5.0ms inference, 0.0ms loss, 0.0ms postprocess per image
Results saved to runs/detect/train
```

4. Results and Model Evaluation

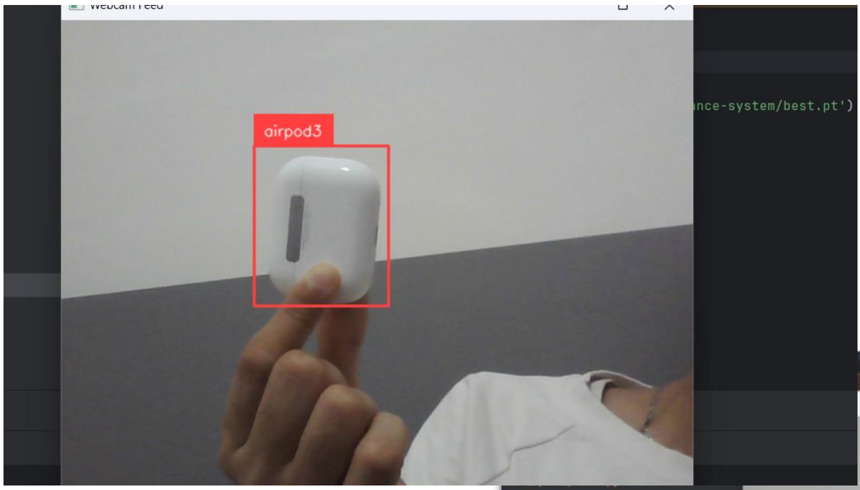
4.1. Real-time Webcam Inference Results



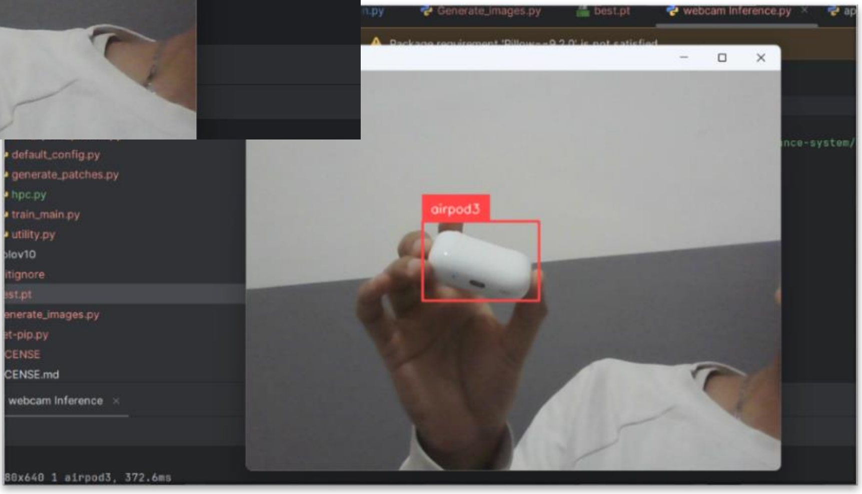
Object detection for bottles.



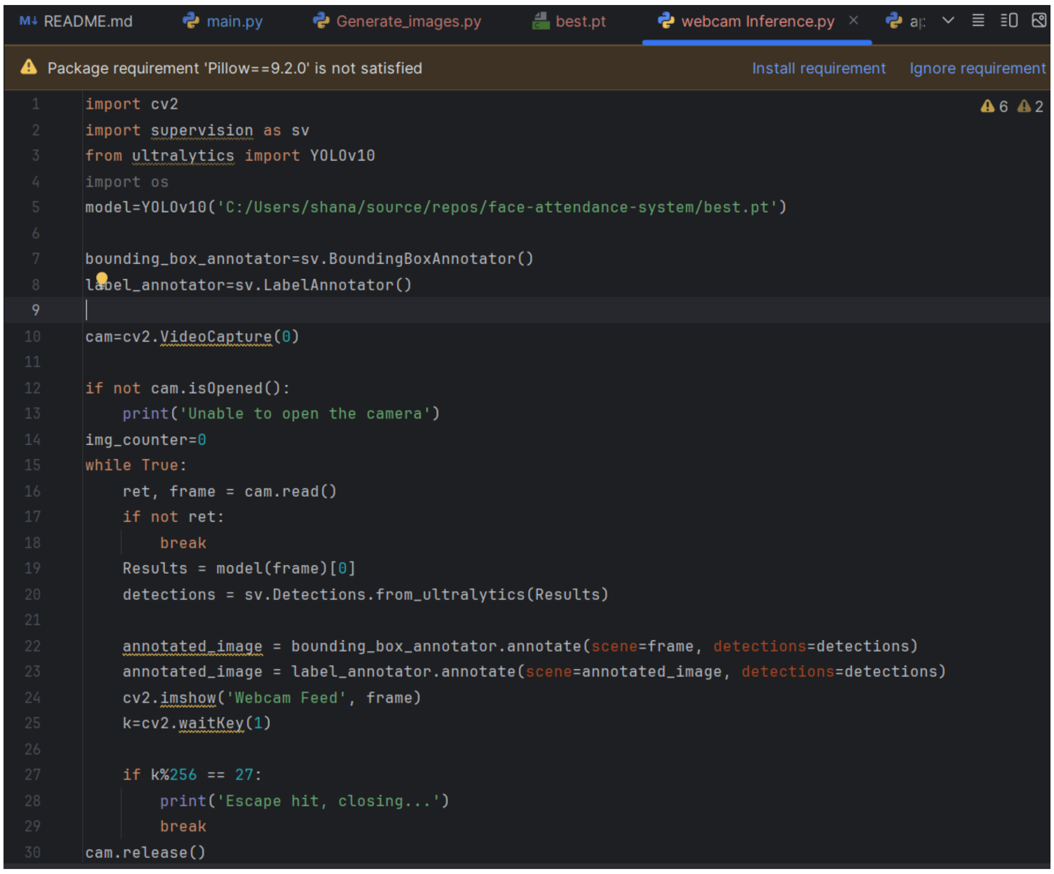
Object detection for Phone & lens.



Object detection for Airpods.



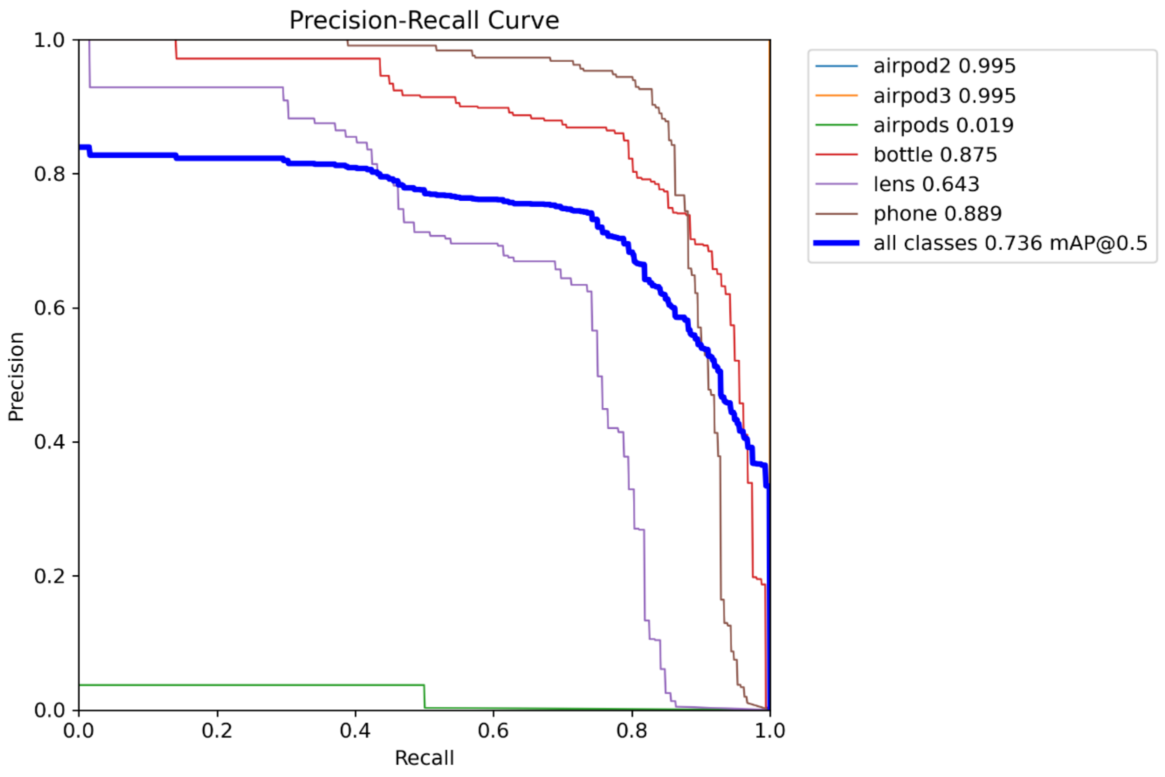
Webcam-inference.py for running the trained dataset on the model and running it live on web camera



Package requirement 'Pillow==9.2.0' is not satisfied

```python
import cv2
import supervision as sv
from ultralytics import YOLOv10
import os
model=YOLOv10('C:/Users/shana/source/repos/face-attendance-system/best.pt')

bounding_box_annotator=sv.BoundingBoxAnnotator()
label_annotator=sv.LabelAnnotator()


cam=cv2.VideoCapture(0)

if not cam.isOpened():
    print('Unable to open the camera')
img_counter=0
while True:
    ret, frame = cam.read()
    if not ret:
        break
    Results = model(frame)[0]
    detections = sv.Detections.from_ultralytics(Results)

    annotated_image = bounding_box_annotator.annotate(scene=frame, detections=detections)
    annotated_image = label_annotator.annotate(scene=annotated_image, detections=detections)
    cv2.imshow('Webcam Feed', frame)
    k=cv2.waitKey(1)

    if k%256 == 27:
        print('Escape hit, closing...')
        break
cam.release()
```

4.2 Model Evaluation

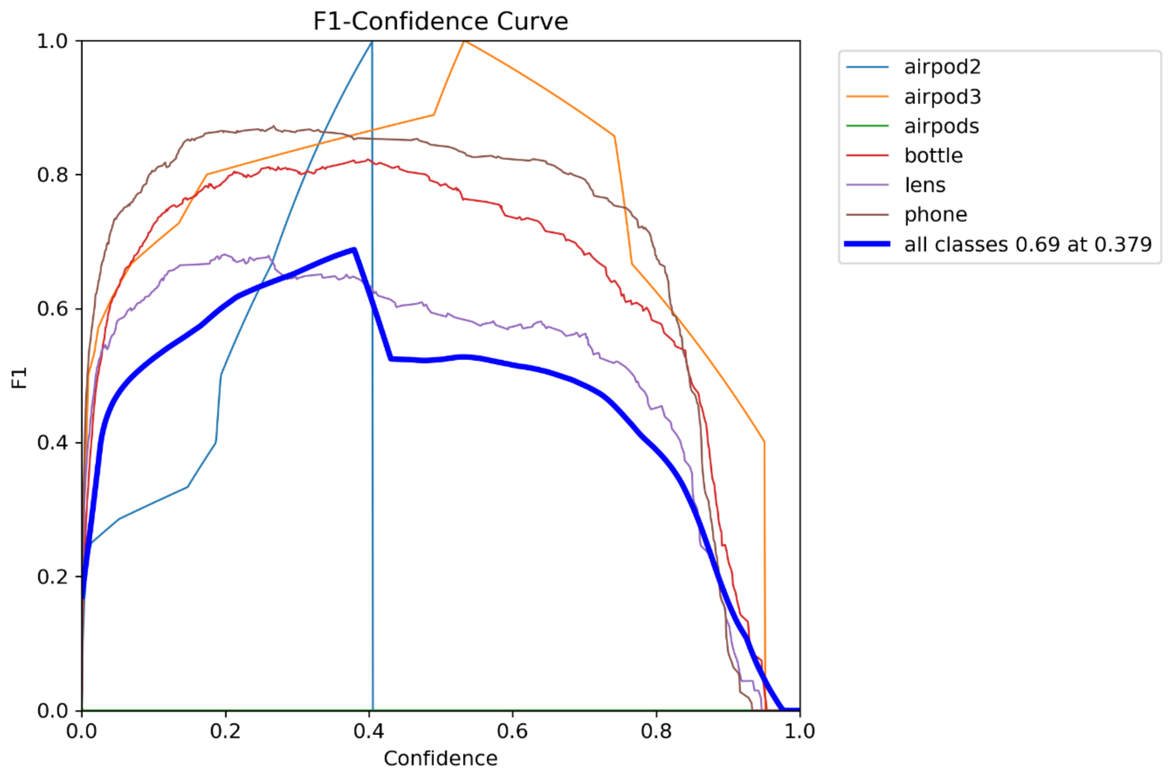


A. Precision-Recall Curve:

This curve plots precision against recall for each class and overall. High precision indicates few false positives (correct predictions among all positive predictions), while high recall indicates few false negatives (correct identification of all actual positives). The curve's shape shows the trade-off between precision and recall—as one increases, the other often decreases.

- Individual Classes: Airpod2 and Airpod3 demonstrate near-perfect precision and recall, indicating excellent performance. Airpods have significantly lower recall, suggesting many misses. Bottle, lens, and phone show moderate performance with a steeper decline in precision as recall increases.
- Overall Performance: The overall curve (blue) shows a reasonable mAP@0.5 (mean Average Precision) of 73.6%, suggesting a fair balance between precision and recall across all classes. However, the individual class performance variations highlight the need for improvement in some areas.

Conclusion: The model excels at detecting Airpod2 and Airpod3 but needs improvement in detecting airpods, and some improvement in detecting bottles and lenses.

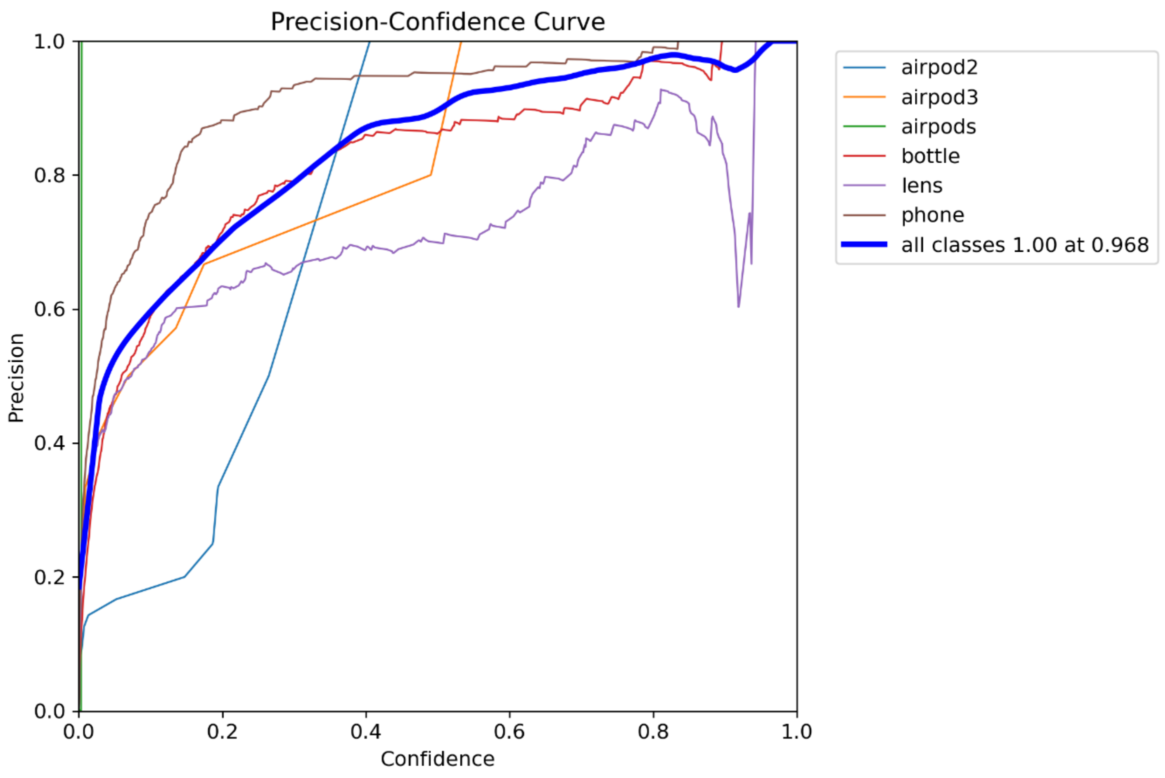


B. F1-Confidence Curve

This curve shows the F1-score (harmonic mean of precision and recall) as a function of confidence threshold. A higher F1-score represents a better balance between precision and recall. The optimal confidence threshold is the point where the F1-score peaks.

- Individual Classes: Again, Airpod2 and Airpod3 exhibit consistently high F1-scores, demonstrating strong performance across varying confidence levels. Airpods shows lower F1 scores while other classes have moderate scores.
- Overall Performance: The overall F1-score reaches a peak of 0.69 at a confidence threshold of 0.379. This indicates a good overall balance between precision and recall at that confidence threshold but is still lower than the peak of Airpod2 and Airpod3.

Conclusion: The model shows good overall performance but class-wise results are less uniform as some classes reach higher F1-scores and optimal confidence thresholds differ. The optimal confidence level for the entire model differs significantly from that of some individual classes.

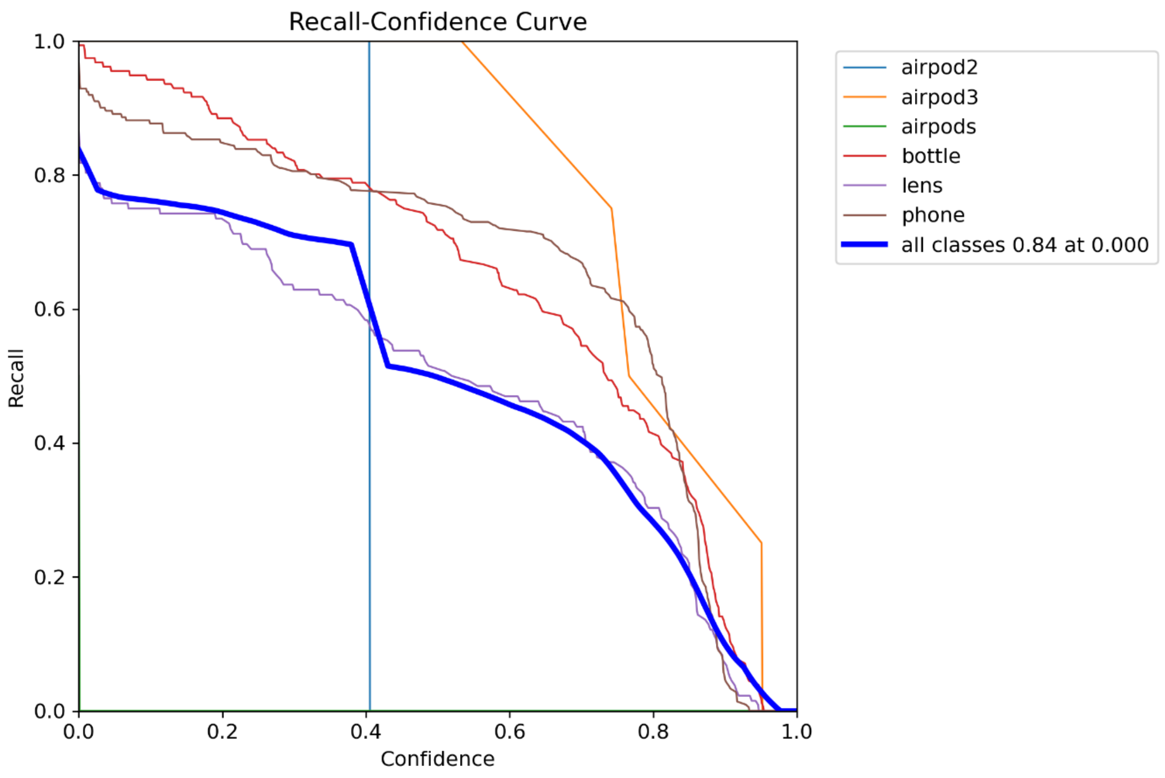


C. Precision-Confidence Curve:

This curve illustrates precision as a function of confidence. High precision at high confidence is desirable.

- Individual Classes: Airpod2 and Airpod3 maintain very high precision even at high confidence levels. Other classes show a decrease in precision as confidence increases, suggesting a greater proportion of false positives at higher confidence levels.
- Overall Performance: The overall precision reaches 1.00 at a confidence threshold of 0.968. This implies that at this confidence threshold the model's positive predictions are almost always correct.

Conclusion: The model's precision is quite high at very high confidence levels but varies significantly across classes, with some classes (Airpod2 and Airpod3) consistently demonstrating high precision even at lower confidence thresholds.

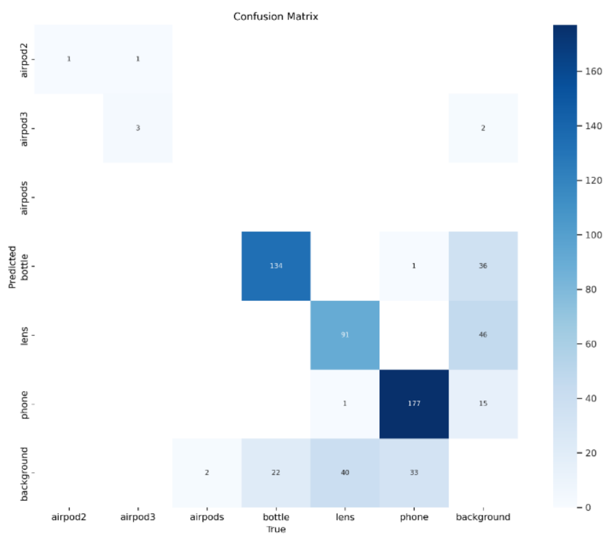


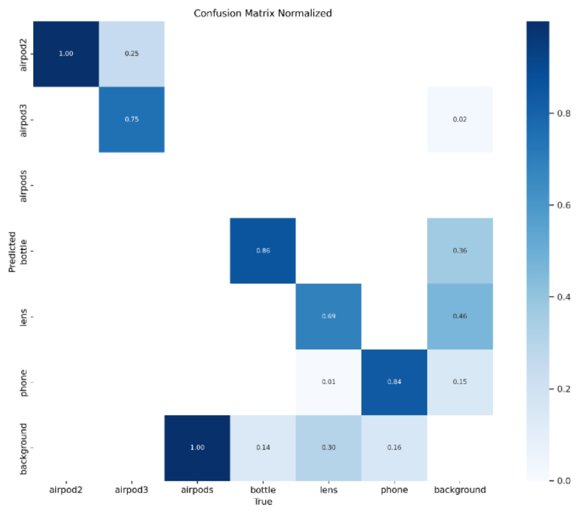
D. Recall-Confidence Curve

This curve displays recall as a function of confidence. High recall at lower confidence thresholds is desirable.

- Individual Classes: Airpod2 and Airpod3 show high recall across a range of confidence levels. The recall of other classes sharply decreases as confidence increases, indicating the model misses a considerable portion of these objects when aiming for higher confidence.
- Overall Performance: Overall recall reaches 0.84 at a confidence threshold of 0.00.

Conclusion: The model exhibits good recall for Airpod2 and Airpod3, but recall drops significantly for other classes at higher confidence levels. This indicates a tradeoff between confidence and the ability to identify all instances of these objects. The model might be overly cautious, leading to missed detections in the pursuit of high confidence predictions.





E. Confusion Matrix

- High Accuracy for Airpod2 and Phone: The model shows high accuracy in identifying Airpod2 and Phone objects, with most instances correctly classified.
- Significant Misclassifications for Airpods: A large number of Airpods are misclassified, primarily as Bottle and Background. This confirms the previously observed low recall for Airpods.
- Moderate Performance for Bottle and Lens: Bottle and Lens show moderate accuracy, with a considerable number of misclassifications between themselves and Background.
- Background Misclassifications: A noticeable number of background instances are incorrectly identified as other objects, particularly Airpods and Bottle.

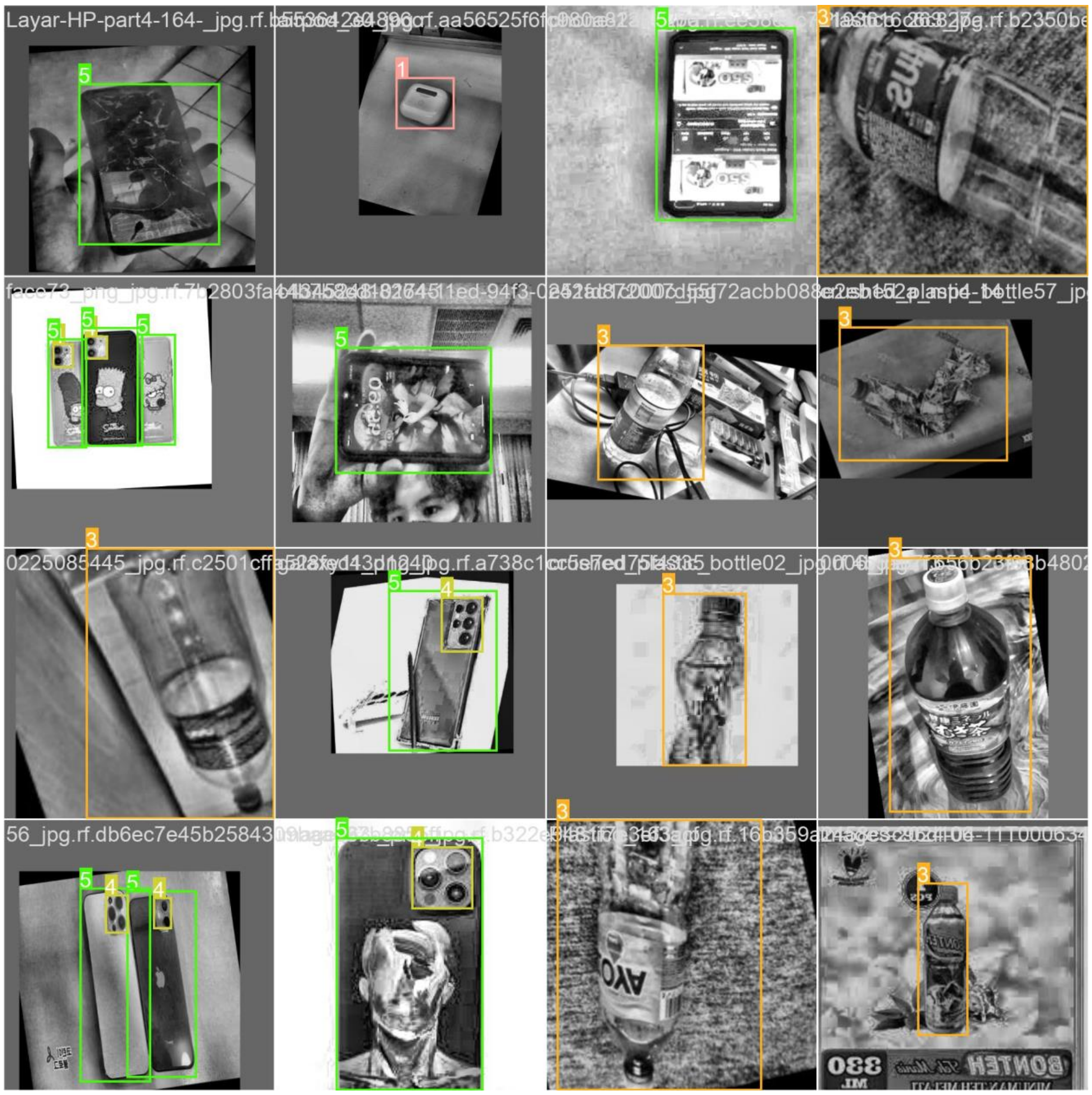
F. Normalized Confusion Matrix (Percentages):

- Airpod2 and Phone Dominate: Airpod2 and Phone show a strong diagonal dominance, indicating a high proportion of correctly classified instances.
- Airpods Misclassification: A significant portion of Airpods instances are misclassified, largely into Bottle and Background. This highlights the model's struggle with this class.
- Bottle and Lens Interchangeability: Bottle and Lens show significant cross-classification, suggesting the model has difficulty distinguishing between these two classes.
- Background Confusion: The Background class also exhibits cross-classification, especially with Airpods and Bottle. This indicates difficulty in differentiating between objects and the background.

Overall Comparison:

The normalized matrix provides a clearer picture of the model's relative performance across classes, confirming the observations from the raw confusion matrix: Airpod2 and Phone perform well, while Airpods struggle significantly. The normalized matrix also emphasizes the model's confusion between Bottle, Lens, and Background, further supporting the need for model refinement. Both matrices strongly suggest a focus on improving the model's ability to distinguish Airpods from other objects and improving the overall class separation of Bottle, Lens, and background.

G. Testing on Sample Images



5. Discussion

5.1. Conclusion

The object detection model demonstrates a mixed performance across different object classes. Airpod2 and Airpod3 achieve excellent results, exhibiting high precision and recall across all evaluation metrics. This indicates the model accurately identifies these objects with minimal errors. In contrast, the model's performance on Airpods is substantially lower. While precision remains relatively high, suggesting accurate identification when a detection is made, recall is significantly weaker, indicating a high rate of missed detections. This disparity highlights a crucial area for improvement.

The Bottle and Lens classes show moderate performance. While precision and recall are acceptable, particularly at higher confidence thresholds, the lower recall at these higher thresholds suggests that the model tends to miss instances of these objects even when it is confident in its predictions. The Phone class, on the other hand, demonstrates a good balance between precision and recall, exhibiting relatively strong performance compared to Airpods, Bottle, and Lens.

Overall, the model achieves a reasonable average performance across all classes, as indicated by a mean Average Precision (mAP@0.5) of 73.6% and a high F1-score. However, the significant variation in performance across individual classes underscores the need for further refinement.

5.2. Comparison with Other Models

YOLOv10 distinguishes itself from other object detection models through its exceptional speed and efficiency, particularly its YOLOv10-N variant which boasts a 70% reduction in latency and computational overhead [1]. While models like YOLOX achieve high accuracy [3], YOLOv10 prioritizes speed, making it ideal for real-time applications [3]. Compared to Faster R-CNN, YOLOv10 sacrifices some precision for significantly faster inference speeds [13], and surpasses Retina Net in both speed and accuracy [1]. The NMS-free approach in YOLOv10 further differentiates it by streamlining the detection process [1]. Compared to SSD, YOLOv10 offers superior accuracy and speed, particularly on embedded devices [1]. The architectural innovations in YOLOv10, including its enhanced backbone and neck, contribute to its overall performance advantage [1]. Finally, while YOLOv10's performance on imbalanced datasets requires optimization [1], its scalability across different model sizes addresses diverse computational needs [1].

5.3. Challenges and Limitations

The model's inconsistent performance, particularly the low recall for Airpods and the lower-than-desired recall for Bottle and Lens at higher confidence thresholds, necessitates improvements to ensure a more robust and reliable object detection system.

5.4. Future Work

Focusing on enhancing the model's ability to detect these underperforming classes will be crucial for achieving consistent accuracy across all target objects. Which in turn will help to improve the overall accuracy of the model.

6. References

https://www.sciencedirect.com/science/article/pii/S0010482522008289
https://github.com/THU-MIG/yolov10
https://www.labelvisor.com/comparing-yolov10-with-other-object-detection-models/
https://arxiv.org/abs/2405.14458
https://github.com/rlggyp/YOLOv10-OpenVINO-CPP-Inference
https://github.com/NielsRogge
https://github.com/Seeed-Projects/jetson-examples/blob/main/reComputer/scripts/yolov10/README.md
https://www.ultralytics.com/blog/the-ultimate-guide-to-data-augmentation-in-2025
https://www.restack.io/p/data-augmentation-knowledge-yolo-object-detection-cat-ai