**Name : Mansi Mahabdi**

**PRN: 22420190**

**Roll No : 382073**

**Batch: B3**

**Assignment No 7**

**Object detection using YOLO and Pretrained Model**

**Problem Statement:**

Implement object detection using the YOLO (You Only Look Once) algorithm with a pretrained model to detect and classify objects in images or videos.

**Objective:**

* To understand the YOLO architecture and its real-time object detection capabilities.
* To use pretrained YOLO models for object detection tasks.
* To detect multiple objects in images and video streams.
* To evaluate the accuracy and efficiency of YOLO in comparison to traditional detection methods.

**Technical Apparatus used:**

* **Operating System:** Windows/Linux/MacOS
* **Kernel:** Python 3.x
* **Tools:** Jupyter Notebook, Anaconda, or Google Colab
* **Hardware:** CPU with minimum 4GB RAM; optional GPU for faster processing

**Libraries and Packages used:**

* **OpenCV**
* **NumPy**
* **Matplotlib**
* **YOLOv3 / YOLOv4 / YOLOv5 pretrained weights**
* **Torch (for YOLOv5)**
* **TensorFlow / Keras (optional for YOLO variants)**

**Theory:**

YOLO (You Only Look Once) is a state-of-the-art, real-time object detection algorithm. Unlike traditional object detection methods that use sliding windows or region proposals, YOLO treats detection as a single regression problem, directly predicting bounding boxes and class probabilities from the entire image in one evaluation. Pretrained YOLO models (such as YOLOv3, YOLOv4, and YOLOv5) are trained on large datasets like COCO, enabling them to detect a wide range of objects.

**Methodology:**

Step 1: Data Acquisition  
- Use sample images or video input for testing pretrained YOLO models.  
  
Step 2: Load Pretrained Model  
- Download YOLO configuration file, pretrained weights, and class labels.  
- Load using OpenCV DNN module or PyTorch Hub (for YOLOv5).  
  
Step 3: Preprocess Input  
- Resize images to 416x416 or 640x640 (depending on YOLO version).  
- Normalize pixel values.  
  
Step 4: Object Detection  
- Pass the input image/video through the YOLO model.  
- Extract bounding boxes, class labels, and confidence scores.  
  
Step 5: Visualization  
- Draw bounding boxes with class labels on detected objects.  
  
Step 6: Evaluation  
- Assess performance based on detection accuracy, confidence threshold, and speed.

**Advantages:**

* Real-time object detection.
* High accuracy with pretrained models.
* Detects multiple objects simultaneously.
* Scalable to different YOLO versions with improvements in speed and accuracy.

**Limitations:**

* Requires GPU for optimal real time performance
* Accuracy depends on the pretrained dataset.
* Struggles with detecting very small objects in images.

**Applications:**

* Autonomous vehicles (detecting pedestrians, traffic signs, vehicles).
* Security and surveillance.
* Retail analytics and inventory monitoring.
* Healthcare (detecting medical instruments or abnormalities in scans).

**Conclusion:**

In this assignment, YOLO was used with pretrained models for object detection tasks. The algorithm demonstrated real-time detection capability by identifying multiple objects in images and videos. This highlights YOLO’s effectiveness in computer vision applications requiring speed and accuracy.

**Implementation & Results:**

The YOLO object detection model was implemented using the pretrained weights (yolov3.weights). The code was executed in Jupyter Notebook (YOLO.ipynb), and detections were performed on sample images.  
  
Below are example detections:

1. Detection on horse image:



2. Detection on bus image:



## References:

- YOLOv3 Pretrained Weights (yolov3.weights)  
- Jupyter Notebook Implementation (YOLO.ipynb)  
- Test Images: horse.jpg, images.jpg