**Assignment No. 7**

**YOLO Object Detection**

**1. Problem Statement:**

Object detection using YOLO and Pretrained Model.

**2. Objective:**

The objective of this assignment is to implement object detection using the YOLO model. We will use a pretrained model to detect objects in images or video streams. This involves:

* Understanding the architecture of YOLO.
* Load a pretrained YOLOv8 model from the Ultralytics library.
* Train and evaluate the model on the **COCO8 sample dataset**.
* Perform inference on a test image

**3. Software and Hardware Packages Used:**

* **Software Packages:**
* Python 3.10 or later
* Jupyter Notebook or Google Colab
* Anaconda for environment management
* YOLOv8 pretrained model weights
* **Hardware Packages:**
* GPU-enabled machine for faster training and inference (e.g., NVIDIA CUDA GPU)
* At least 8 GB RAM for processing

**4. Libraries Used:**

* ultralytics: For implementing YOLO models.
* NumPy: Array processing for numerical operations.
* OpenCV: Image and video processing.
* Matplotlib: Visualization of detected objects.
* PIL (Python Imaging Library): For handling image data.

**5. Theory:**

* YOLO (You Only Look Once) is a state-of-the-art object detection model. Unlike traditional models that process an image in a sliding window manner, YOLO applies a single neural network to the entire image, dividing it into grids. Each grid predicts bounding boxes and the probability of classes within those boxes. Key concepts include:
* YOLO Architecture: It uses a convolutional neural network (CNN) to detect objects and predict their bounding boxes.
* Pretrained Models: Models trained on large datasets like COCO (Common Objects in Context) to detect a variety of objects.

**6. Methodology:**

1. **Model Loading:**

* The pretrained YOLOv8-Nano (yolov8n.pt) model was loaded using the Ultralytics library.
* Preprocessing: Resize images to the input size required by the YOLO model (e.g., 640x640). Convert images to a format suitable for the model.

1. **Dataset:**

* The experiment used the **COCO8 built-in dataset** provided with Ultralytics. It is a small subset of the COCO dataset designed for quick demonstrations

1. **Training:**

* The model was trained on the dataset for 3 epochs with an image size of 640×640.

1. **Post-Processing:**

* Non-Maximum Suppression (NMS): Removes redundant bounding boxes with lower confidence scores. NMS ensures that only the most relevant boxes are retained for each detected object.
* Thresholding: Set a confidence threshold (e.g., 0.5) to filter out weak detections and focus only on objects with high confidence scores.

1. **Evaluation:**

* After training, the model was validated using model.val(), which provided metrics such as precision, recall, mAP@0.5, and mAP@0.5:0.95.

**7. Advantages:**

* **Real-time Detection:** Capable of processing images quickly, making it suitable for image feeds.
* **High Accuracy:** Even with a single forward pass, YOLO can detect multiple objects with good precision.
* **Pretrained Models:** Leverages large datasets, allowing users to use out-of-the-box detection without needing extensive training.

**8. Inference**

* A pretrained model was used for object detection on a test image (bus.jpg). The original test image was visualized using Matplotlib, and predicted images with bounding boxes were saved in the results directory.

**8. Limitations:**

* **Struggles with Small Objects:** YOLO’s grid-based approach can sometimes miss smaller objects due to spatial constraints.
* **Trade-off Between Speed and Accuracy:** While faster than many detection models, YOLO might compromise slightly on precision.
* **Complex Objects:** It can be less effective when detecting complex or overlapping objects.

**9. Applications:**

* **Autonomous Vehicles:** Detecting pedestrians, vehicles, and obstacles in real-time.
* **Surveillance:** Monitoring objects and people in security systems.
* **Healthcare:** Detecting abnormalities in medical imaging (e.g., X-rays, MRIs).
* **Retail:** Product detection and inventory management using cameras.
* **Gaming and AR/VR:** Real-time interaction with virtual environments through object tracking.

**10. Working/Algorithm:**

**Initialization:**

* Load the YOLO model weights (e.g., yolov8s.pt).
* Define the input image size (e.g., 640x640 pixels).

**Image Preprocessing:**

* Convert the input image to a tensor format required by the YOLO model.
* Normalize pixel values to [0, 1].
* Resize the image to the YOLO input size (e.g., 640x640).

**Prediction**:

* Pass the preprocessed image through the YOLO model.
* YOLO divides the image into an S×SS \times SS×S grid (e.g., 13x13).
* Each cell in the grid predicts multiple bounding boxes (e.g., 3) and object confidence scores.
* **Bounding Box Details**:
  + YOLO predicts 5 values for each bounding box: x,y,w,h,x, y, w, h,x,y,w,h, and confidence.
  + (x,y)(x, y)(x,y) represents the center coordinates of the box.
  + www and hhh represent the width and height of the box relative to the cell.
  + Confidence score represents the probability of an object being present in the box.

**Class Prediction:**

* YOLO predicts class probabilities for each bounding box.
* Multiply the object confidence score by the class probability to get the final score for each class.
* **Non-Maximum Suppression (NMS):**
* Apply NMS to reduce overlapping bounding boxes and retain only the most confident one.
* Steps of NMS:

1. Sort all bounding boxes by their confidence scores.
2. Select the box with the highest confidence score.
3. Compute the Intersection over Union (IoU) between this box and other boxes.
4. Remove boxes with IoU greater than a defined threshold (e.g., 0.5).
5. Repeat until all boxes are processed.

* **Post-Processing:**
* Filter out detections below a certain confidence threshold (e.g., 0.5).
* Convert relative bounding box coordinates back to absolute values (pixel values) to draw them on the original image.

**Drawing and Visualization:**

* Loop through the retained bounding boxes.
* Draw rectangles on the image using OpenCV for each bounding box.
* Display class labels and confidence scores on top of each detected object.
* Display the final image or video frame with annotations.

**Output:**

* The final output is an annotated image or video stream showing detected objects with their corresponding labels and bounding boxes.

**12. Conclusion:**

* The experiment successfully demonstrated object detection using the YOLOv8 model and the COCO8 built-in dataset. The pretrained YOLOv8 model achieved reasonable accuracy within only a few epochs of training. The results confirm that YOLOv8 is highly effective for **real-time object detection**, offering a good balance between speed and accuracy. Although performance on small datasets is limited, YOLOv8 remains a strong choice for practical applications such as surveillance, autonomous driving, and real-time video analytics.