Student name: Neha Sawant **PRN:** 22210877

Assignment No. 7

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.
- Loading and fine-tuning a pretrained YOLO model.
- Evaluating its performance on different images or video streams for real-time object detection.

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
  - Web camera or external camera (for real-time object detection)

4. Libraries Used

- ultralytics: For implementing YOLO models.
- NumPy: Array processing for numerical operations.
- OpenCV: Image and video processing.

- torch and torchvision: For deep learning model handling.

- 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.
- Real-time Detection: YOLO can process images at high speeds, making it ideal for applications requiring real-time analysis.

6. Methodology

Data Preparation:

- Image/Video Collection: Gather images or videos containing objects that you want to detect.
- Preprocessing: Resize images to the input size required by the YOLO model (e.g., 640x640). Convert images to a format suitable for the model.

Model Loading:

- Pretrained Model: Load a pretrained YOLO model, YOLOv8, from the ultralytics library. YOLO models are typically pretrained on large datasets like COCO.

Inference:

- Input Image to Model: Pass the input image to the YOLO model. The model processes the image and outputs predictions.
- Bounding Box Prediction: YOLO divides the image into a grid (e.g., 13x13, 26x26). Each cell in the grid predicts a set number of bounding boxes, object confidence scores, and class probabilities.
- Object Confidence Score: Represents how confident the model is that an object is present in a particular bounding box.
- Class Probabilities: Likelihood that a detected object belongs to a specific class (e.g., person, car, dog).

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.

Visualization:

- Draw Bounding Boxes: Use OpenCV to draw bounding boxes around detected objects.
- Display Results: Show the annotated image or video stream with detected objects.

Evaluation:

- Metrics: Calculate accuracy, precision, recall, and F1-score based on the predictions and ground truth labels.
- Qualitative Analysis: Evaluate how well the model performs on different types of images and adjust parameters as needed.

7. Algorithm/Working

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.

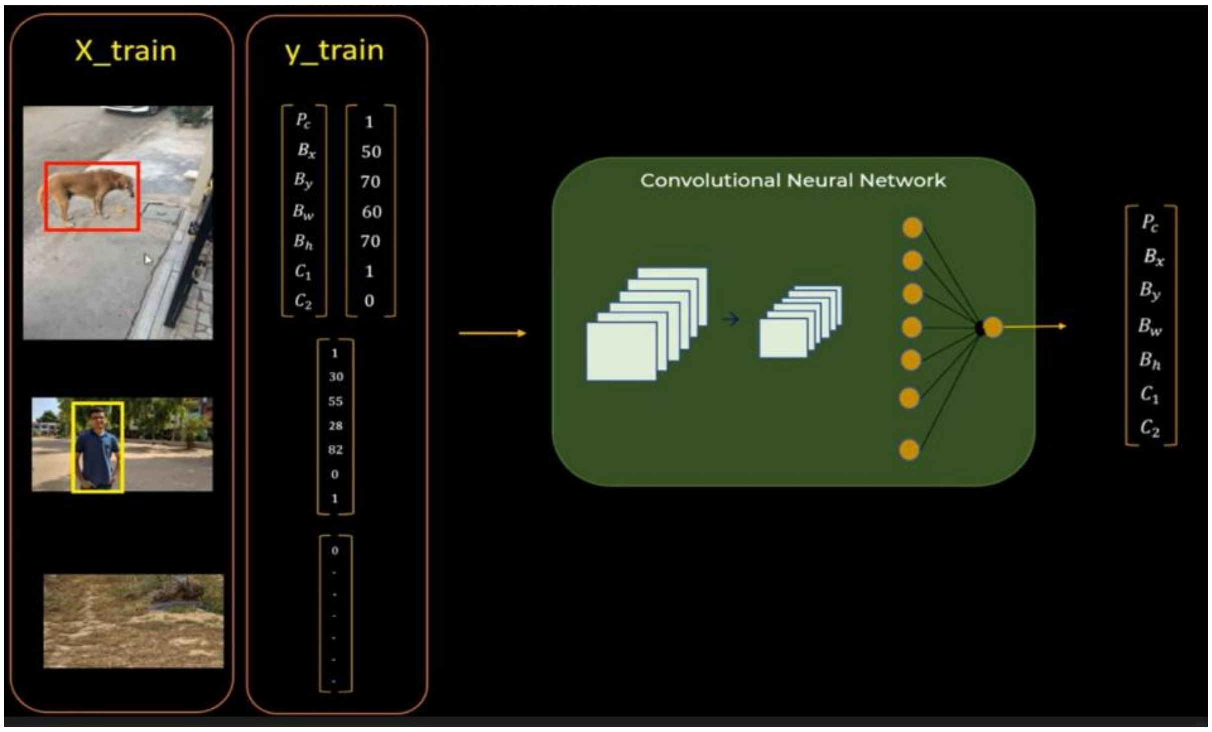
Drawing and Visualization:

- 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.

8. Diagram



9. Advantages

- Real-time Detection: Capable of processing images quickly, making it suitable for video 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.

10. 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.

11. 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.

12. Conclusion

Object detection using YOLO and pretrained models allows for efficient and effective identification of objects in images and videos. By leveraging the speed and accuracy of YOLO, various real-time applications are possible. This practical assignment demonstrates the implementation of YOLO for detecting multiple objects with high accuracy. Despite some limitations in detecting small or overlapping objects, YOLO remains a popular choice for object detection tasks in diverse fields.