Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

Assignment - RCNN and Yolo

1. What is the main purpose of RCNN in object detection?

   ➔ The main purpose of **R-CNN (Region-based Convolutional Neural Network)** in object detection is to effectively localize and classify objects within an image. R-CNN achieves this by addressing two key tasks:

   - **Object Localization**: Identifying regions within an image that potentially contain objects (region proposals).
   - **Object Classification**: Determining the class of the objects in the proposed regions.

     R-CNN bridges the gap between traditional object detection methods (which rely on handcrafted features) and modern deep learning-based approaches by leveraging CNNs for feature extraction. Its design makes it suitable for complex object detection tasks where precise localization and classification are required.

2. What is the difference between Fast RCNN and Faster RCNN?

   ➔ The primary difference between **Fast R-CNN** and **Faster R-CNN** lies in how they handle **region proposals**, which are the candidate bounding boxes for potential objects in an image. Here's a breakdown of their differences:

   - **Region Proposal Generation:** Fast RCNN relies on an **external region proposal algorithm** such as **Selective Search** to generate region proposals whereas Faster RCNN is a fully convolutional network that generates region proposals directly from feature maps, significantly speeding up the process.
   - **Speed and Efficiency:** Fast RCNN still suffers from the bottleneck of using Selective Search for generating region proposals whereas Faster RCNN by replacing Selective Search with the RPN, Faster R-CNN eliminates the bottleneck, making it significantly faster.
     Faster R-CNN is an advancement over Fast R-CNN, providing a faster and more efficient framework for object detection.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

3. How does YOLO handle object detection in real-time?

➔ **YOLO (You Only Look Once)** handles object detection in real-time by rethinking the traditional object detection pipeline as a single, unified neural network model. Its design optimizes both speed and accuracy, enabling it to process images at high frame rates. Here's how YOLO achieves real-time performance:

- Unified Architecture
- Grid Based Prediction
- Direct Prediction
- Real Time Speed

YOLO's unified architecture, grid-based prediction, and efficient computational design enable it to handle object detection in real-time, making it ideal for applications like autonomous vehicles, surveillance, and robotics.

4. Explain the Region Proposal Networks (RPN) concept in Faster RCNN.

➔ The **Region Proposal Network (RPN)** is a key innovation in **Faster R-CNN**, designed to generate high-quality **region proposals** directly from feature maps in a deep learning model. By integrating RPN into the architecture, Faster R-CNN eliminates the need for external region proposal methods like Selective Search, significantly improving speed and accuracy.

How RPN is faster than RCNN-

- Feature map Sharing: RPN and the object detection head (classifier and regressor) share the same feature map, reducing computational redundancy.
- End-to-End Training: The RPN is trained jointly with the detection network, allowing the entire Faster R-CNN architecture to learn optimal region proposals and object detection simultaneously.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

5. How does YOLOv9 improve upon its predecessors?

➔ YOLOv9 builds upon its predecessors with significant enhancements in accuracy, computational efficiency, and versatility for different use cases. Some key improvements include:

- Advanced Architecture - Improves gradient computations for better training efficiency and accuracy, addressing the information bottleneck in deep neural networks.
- Improved Performance Metrics - Achieves a higher mean Average Precision (mAP) compared to previous versions, such as a 1.7% improvement over YOLOv8-X while reducing the number of parameters and computational complexity.
- Compact and Efficient—Smaller model sizes and lower memory requirements make it more suitable for edge devices and real-time application deployment.

6. What role does non-max suppression play in YOLO object detection?

➔ Non-max suppression (NMS) is a critical post-processing step in YOLO (You Only Look Once) object detection algorithms. It ensures that the model outputs meaningful and non-redundant detection results by eliminating duplicate bounding boxes that represent the same object.

Role of Non-Max Suppression:

- Removing Overlapping Boxes: YOLO generates multiple bounding boxes for each object in the image, with slightly different positions and confidence scores. NMS removes these redundant detections by keeping only the most confident bounding box for each object.
- Balancing Precision and Recall: The IoU threshold in NMS allows control over the trade-off between precision (avoiding duplicate detections) and recall (ensuring all objects are detected). A high threshold keeps more boxes, while a lower threshold aggressively removes overlaps.
- Speed and Real-Time Efficiency - By filtering out unnecessary bounding boxes, NMS reduces the computational burden in subsequent steps and aligns with YOLO's goal of real-time object detection.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

7. Describe the data preparation process for training YOLOv9.

➔ The data preparation process for training YOLOv9 follows a structured workflow to ensure that the model is trained effectively for accurate object detection. Below are the key steps involved in preparing the dataset:

- Dataset Collection: Gather labeled datasets from public sources like COCO, PASCAL VOC, or custom datasets specific to the use case.
- Labelling and Annotation: Use tools like **LabelImg**, **CVAT**, or **Roboflow** to create bounding box annotations for objects in the dataset.
- Dataset Splitting: Split the dataset into **training**, **validation**, and **test** sets.
- Data Augmentation: Apply transformations to increase dataset diversity and reduce overfitting.
- Normalization and Preprocessing: Normalize image pixel values and resize all images.
- Verify Dataset and Pretrained Weights

8. What is the significance of anchor boxes in object detection models like YOLOv9?

➔ Anchor boxes play a crucial role in object detection models like YOLOv9 by acting as predefined bounding boxes that help detect objects of varying shapes and sizes. They streamline the process of predicting object locations and class labels efficiently.

- Efficient Localization
- Support for multiclass Detection
- Handling varying Aspect Ratios
- Optimization of Computation
- Compatibility with Reak-Time Detection

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

9. What is the key difference between YOLO and R-CNN architectures?

➔ The key difference between YOLO (You Only Look Once) and R-CNN (Region-based Convolutional Neural Networks) architectures lies in their approach to object detection, particularly in how they handle region proposals and their computational efficiency.

a. Processing approach: YOLO takes a **single-stage** detector that directly predicts bounding boxes and class probabilities for the entire image in one pass whereas R-CNN is a **two-stage** detector. First, it generates **region proposals** using selective search. Then, these regions are classified and refined using convolutional neural networks.
b. Region Proposals: YOLO does not rely on explicit region proposal methods. Instead, it uses anchor boxes or grid-based predictions to detect objects whereas R-CNN generates explicit region proposals using selective search.

10. Why is Faster RCNN considered faster than Fast RCNN?

➔ Faster R-CNN is considered faster than Fast R-CNN due to the integration of **Region Proposal Networks (RPNs)**, which streamline the region proposal generation process. Here's a detailed comparison:

- **Region Proposal Generation:** Fast RCNN relies on an **external region proposal algorithm** such as **Selective Search** to generate region proposals whereas Faster RCNN is a fully convolutional network that generates region proposals directly from feature maps, significantly speeding up the process.
- **Speed and Efficiency:** Fast RCNN still suffers from the bottleneck of using Selective Search for generating region proposals whereas Faster RCNN by replacing Selective Search with the RPN, Faster R-CNN eliminates the bottleneck, making it significantly faster. Faster R-CNN is an advancement over Fast R-CNN, providing a faster and more efficient framework for object detection.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

11.What is the role of selective search in RCNN?

➔ Selective Search plays a critical role in **R-CNN (Region-based Convolutional Neural Networks)** by generating region proposals that are used to identify potential objects within an image. This process is essential for the R-CNN architecture because it provides the regions that the network analyzes to detect objects. Here's an in-depth look at the role of selective search in R-CNN:

- Generating Region Proposals
- Improved Detection
- High Quality Proposals

12.How does YOLOv9 handle multiple classes in object detection?

➔ YOLOv9, like its predecessors, handles multiple classes in object detection through a combination of features in its architecture and loss functions. Here's a breakdown of how it achieves multi-class detection:

- Grid-Based Prediction - YOLO models, including YOLOv9, divide the input image into an **S×SS \times SS×S grid**. Each grid cell is responsible for detecting objects whose center falls within that cell. This approach allows the model to predict multiple bounding boxes and class probabilities for each cell, effectively handling multiple objects and classes.
- Class Prediction - For each grid cell and anchor box combination, YOLOv9 outputs class probabilities, typically through a **softmax** or **sigmoid activation** function (the latter if predicting multiple labels per anchor is allowed). This enables the network to handle multiple classes by outputting scores for each class independently.

  YOLOv9 manages multiple classes by leveraging a grid-based detection approach combined with anchor boxes, class-specific probability outputs, and post-processing techniques like NMS. These elements enable the model to detect and classify objects efficiently and accurately within an image, handling a diverse range of classes in real-time.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

13.What are the key differences between YOLOv3 and YOLOv9?

➔ The key differences between **YOLOv3** and **YOLOv9** stem from advancements in the underlying architecture, model optimization, and performance improvements aimed at achieving better accuracy and speed.

- **Backbone and Feature Extraction**: **YOLOv3** uses **Darknet-53** as its backbone, which is a deep CNN architecture with 53 layers. It provides a good balance between speed and accuracy but is less efficient whereas **YOLOv9** incorporates more advanced backbone networks, such as **CSPDarknet** or custom lightweight backbones.
- **Train:** YOLOv3 uses batch normalization and leaky ReLU as its primary activation function whereas **YOLOv9** features enhanced training techniques such as **CIOU (Complete Intersection over Union) loss**, **self-adversarial training**, and **mixed precision training**.

14.How is the loss function calculated in Faster RCNN?

➔ In **Faster R-CNN**, the loss function is designed to train the model by combining two main components: **classification loss** and **bounding box regression loss**. These components help the model optimize both the accuracy of the class predictions and the precision of the object localization.

The **classification loss** is typically computed using **softmax cross-entropy**, which measures the difference between the predicted class probabilities and the true class labels for each proposal.
The **bounding box regression loss** is used to refine the predicted bounding boxes. This is achieved by minimizing the distance between the predicted box coordinates and the ground truth.
These combined losses allow Faster R-CNN to effectively learn to both classify objects and precisely predict their locations within an image.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

15. Explain how YOLOv9 improves speed compared to earlier versions.

➔ **YOLOv9** improves speed compared to earlier versions through several key optimizations and enhancements in its architecture and training processes. Here's how these changes contribute to faster performance:

- Efficient Backbone Networks - YOLOv9 incorporates lighter and more optimized backbones such as CSPDarknet, MobileNetV3, or custom streamlined architectures that reduce computational complexity and improve inference speed.
- Sparsity - **Sparsity** techniques are also applied, where non-essential connections in the network are pruned, reducing the overall number of computations needed.
- Lightweight Head Design - The detection head in YOLOv9 is optimized for speed, typically using **fewer parameters** and **simplified activation functions** compared to YOLOv3 and its predecessors, reducing the overhead during prediction.

16. What are some challenges faced in training YOLOv9?

➔ **Large and diverse Dataset -** YOLOv9, like other deep learning models, requires a vast and diverse dataset to achieve high accuracy. Collecting and annotating large amounts of high-quality data for training can be time-consuming and expensive.

➔ **Computational Resources -** Training YOLOv9 with its enhanced features and optimizations requires significant computational resources. This can mean high costs for using powerful GPUs or TPUs, especially when training on large-scale datasets.

➔ **Hyperparameter Tuning -** Fine-tuning hyperparameters (such as learning rate, batch size, and regularization terms) is crucial for achieving the best performance.

➔ **Overfitting -** The complexity of YOLOv9's model may lead to overfitting, particularly when training on smaller datasets. Regularization techniques such as dropout, data augmentation, and weight decay are necessary to mitigate this risk.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

17. How does the YOLOv9 architecture handle large and small object detection?

➔ The **YOLOv9** architecture has been designed with several improvements to effectively handle both **large and small object detection**. This capability is crucial for real-world applications where objects vary significantly in size. Below are the main ways YOLOv9 addresses this challenge:

- Enhanced Backbone Network
- Attention Mechanism
- Multi-Scale Detection Heads
- Loss Function Enhancements

YOLOv9's ability to handle large and small object detection effectively comes from its **enhanced backbone network**, **multi-scale feature integration**, **attention mechanisms**, **optimized anchor boxes**, and **advanced loss functions**. These architectural choices and training strategies make YOLOv9 a powerful model for real-time object detection with high accuracy across various object scales.

18. What is the significance of fine-tuning in YOLO?

➔ **Fine-tuning** is a crucial technique in training YOLO (You Only Look Once) models, particularly for improving the performance on specific tasks or adapting to new data distributions. Here's why fine-tuning is significant in YOLO:

- **Transfer Leaning Benefits:** Fine-tuning allows you to start with a pre-trained YOLO model that has learned general features from a large dataset (e.g., COCO or ImageNet). This pre-training saves time and computational resources since the model doesn't need to learn basic features from scratch.
- **Improving Model Performances:** By training on new data that better represents the deployment environment, fine-tuning helps the model generalize well to real-world scenarios, which can involve different lighting, occlusions, or scales.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

19.What is the concept of bounding box regression in Faster RCNN?

➔ **Bounding box regression** in **Faster R-CNN** is a crucial step for refining the localization of detected objects. The concept is designed to improve the accuracy of the bounding boxes predicted by the region proposal network (RPN) or any other object detection network component.

Bounding box regression in Faster R-CNN is a method used to refine the initial bounding box proposals generated by the RPN to match the ground-truth object locations more closely. By predicting offsets for the positions and sizes of the boxes and applying them to the initial anchor boxes, Faster R-CNN achieves improved object localization, which is essential for accurate object detection.

20.Describe how transfer learning is used in YOLO.

➔ **Transfer learning** is a powerful technique used in training YOLO models, including **YOLOF (YOLO with Fast and Efficient Object Detection)**, to leverage pre-trained weights and adapt them to specific tasks.

- Start with Pre-trained Weights - YOLOF can be initialized using weights from a pre-trained model, such as **YOLOv4** or **YOLOv5**, which have been trained on large-scale datasets like COCO or ImageNet. This approach saves time and computational resources by not having to train from scratch.
- Fine-Tuning for specific Datasets - After initializing with pre-trained weights, the YOLOF model undergoes **fine-tuning** on a task-specific dataset. Fine-tuning adjusts the pre-trained weights to better suit the distribution and characteristics of the new dataset.
- Adaptation to Smaller Datasets - Techniques like **data augmentation** can be used alongside transfer learning to further improve the model's robustness and generalization on smaller datasets.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

21. What is the role of the backbone network in object detection models like YOLOv9?

➔ The **backbone network** in object detection models like **YOLOv9** plays a crucial role in extracting meaningful features from input images. Here's an overview of its significance:

- Feature Extraction - The backbone network is responsible for processing raw image data and transforming it into feature maps that highlight important patterns and structures. This is essential for identifying objects within the image.
- Key Components in YOLO Architecture - YOLO models, including YOLOv9, use backbones such as **CSPDarknet**, **ResNet**, or modified versions of **VGG** to perform this task effectively. Each of these architectures has strengths in balancing computational efficiency and feature extraction capabilities.
- Supporting Multi-scale Detection - Object detection often involves recognizing objects of different sizes. Backbones in modern YOLO models are designed to handle multi-scale detection by including structures that capture both fine-grained and global features.

22. How does YOLO handle overlapping objects?

➔ YOLO (You Only Look Once) handles overlapping objects through a combination of design strategies that include **bounding box prediction**, **confidence scores**, and **non-maximum suppression (NMS)**. Here's how these components work together to address overlapping object scenarios:

- Bounding Box Prediction
- Non-Maximum Suppression
- Anchor Boxes and Aspect Ratios
- Improvements in Later YOLO Versions

YOLO handles overlapping objects primarily through **NMS**, which eliminates redundant bounding boxes and retains only the best ones, and **anchor boxes** that adapt to various object shapes and scales. Later versions improve this with deeper architectures and enhancements to better detect overlapping.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

23.What is the importance of data augmentation in object detection?

➔ **Data augmentation** is a crucial technique in object detection that significantly enhances model performance by artificially expanding the training dataset. This process helps create diverse variations of the input images through transformations, thus enabling the model to generalize better. Here's why data augmentation is important in object detection.

- Improving Model Generalization
- Handling Real world variations
- Addressing Class Imbalance
- Reducing the need for Large Datasets

By using data augmentation, object detection models become more capable of recognizing objects under varied and challenging conditions, enhancing their accuracy and reliability in real-world applications.

24.How is performance evaluated in YOLO-based object detection?

➔ Mean Average Precision: **mAP** is the most widely used metric for evaluating the performance of object detection models, including YOLO. It summarizes the precision-recall curve by calculating the average precision (AP) for each class and then averaging these across all classes.

➔ Intersection over Union: **IoU** measures how much the predicted bounding box overlaps with the ground truth bounding box. It is calculated as the area of the intersection divided by the area of the union of the predicted and ground truth boxes.

➔ Precision and Recall: **Precision** indicates how many of the predicted positive detections are correct, while **recall** indicates how many of the actual positive detections were found by the model.

➔ Confusion Matrix: A confusion matrix can be used to provide a summary of prediction results, showing counts of true positives, false positives, false negatives, and true negatives for each class. While this is more common for classification tasks, it can also be adapted for object detection evaluations.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

25. How do the computational requirements of Faster RCNN compare to those of YOLO?

➔ The computational requirements of **Faster R-CNN** and **YOLO** differ significantly due to their architectural designs and intended use cases. Here's a comparison highlighting the key aspects:

- Architecture Complexity – Faster RCNN model is more complex and consists of multiple stages. It includes a **Region Proposal Network (RPN)** that generates region proposals, which are then passed to a second network for object classification and bounding box regression whereas YOLO is designed as a single, unified network that performs object detection in one forward pass.
- Inference Time - Due to its multi-stage nature, Faster R-CNN typically has higher inference times compared to YOLO.
- GPU/CPU Usage – Faster RCNN is more resource-demanding and requires a more powerful GPU or multi-core CPU for efficient processing whereas YOLO is less demanding than Faster R-CNN in terms of memory and processing.

26. What role do convolutional layers play in object detection with RCNN?

➔ Convolutional layers are foundational to **object detection with RCNN** (Region-based Convolutional Neural Networks), playing a crucial role in feature extraction, which underpins the detection process.

- Feature Extraction: Convolutional layers are designed to extract features from the input image, transforming raw pixel values into higher-level feature maps that represent more abstract information (e.g., edges, textures, shapes, and patterns). These features are essential for identifying objects in an image.
- Region Proposal: The **Region Proposal Network (RPN)** in **Faster RCNN** uses convolutional layers to generate region proposals. These proposals suggest potential bounding boxes that could contain objects. The RPN scans the feature maps produced by the convolutional layers and applies sliding windows to propose regions, predicting both the objectness score and the bounding box coordinates.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

27. How does the loss function in YOLO differ from other object detection models?

➔ The loss function in **YOLO** (You Only Look Once) differs significantly from those used in other object detection models, such as **Faster RCNN** or **SSD** (Single Shot Multibox Detector). This difference arises from YOLO's unique design and its approach to predicting multiple objects in a single forward pass.

YOLO's loss function allows for faster, end-to-end training and inference because it merges multiple aspects of object detection into one streamlined process. This contributes to YOLO's ability to process images in real-time.

YOLO's unified loss function, integrating localization, confidence, and classification losses, is key to its speed and simplicity in training and inference. This contrasts with the more modular and often more complex loss structures in models like **Faster RCNN**, which separate object classification and localization tasks.

28. What are the key advantages of using YOLO for real-time object detection?

➔ YOLO (You Only Look Once) is highly regarded for real-time object detection due to its unique architecture and key advantages:

- Speed and Efficiency - YOLO's primary advantage lies in its ability to perform object detection in a single pass through the network. Unlike models such as **Faster RCNN**, which separate region proposal and classification into multiple stages, YOLO directly predicts bounding boxes and class probabilities from full images in one network evaluation.
- Practical Applications - YOLO has a strong open-source presence, which customizes and fine-tunes it for specific tasks, contributing to its widespread adoption in industries requiring quick object recognition.

YOLO's architecture enables it to be one of the fastest and most efficient objects durable, offering real-time performance with a good balance of accuracy. Its ability to process frames at high speed, combined with improvements in subsequent versions.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

29.How does Faster RCNN handle the trade-off between accuracy and speed?

➔ **Faster RCNN** handles the trade-off between accuracy and speed by integrating different strategies that balance these two competing aspects.

- **Two Stage Detection Framework -** Faster RCNN introduces an RPN that generates region proposals directly from the feature maps, eliminating the need for external region proposal algorithms like Selective Search used in the original RCNN. This change speeds up the proposal generation phase and improves overall efficiency.
- **Backbone Network Selection:** The backbone network (e.g., ResNet, VGG) is a crucial component of Faster RCNN. The choice of backbone affects both the accuracy and the speed of the model. More complex backbones (like ResNet with deeper layers) provide higher accuracy due to better feature extraction but come with increased computational cost.

30.What is the role of the backbone network in both YOLO and Faster RCNN, and how do they differ?

➔ The **backbone network** in both **YOLO** and **Faster RCNN** plays a critical role in feature extraction, which forms the foundation for object detection.

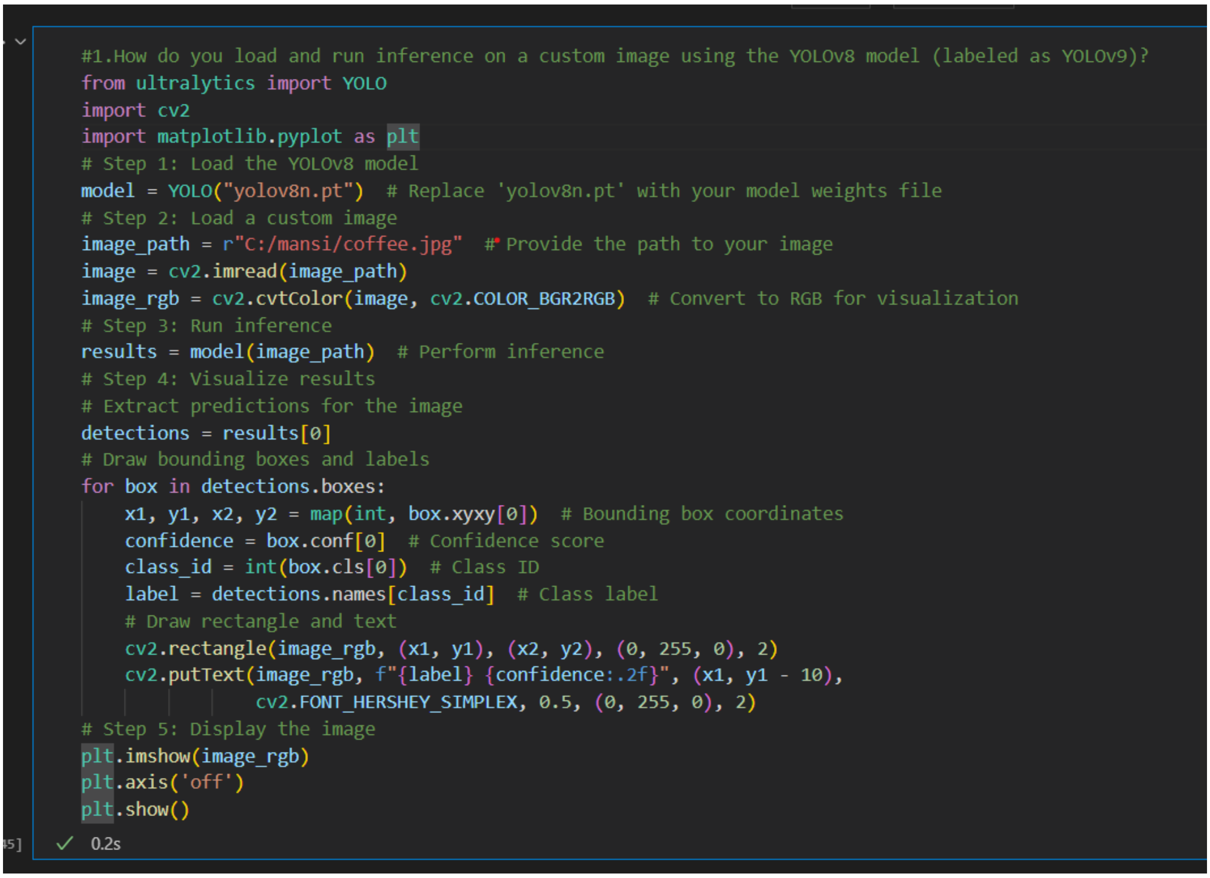
- Roles of backbone network - The backbone network extracts hierarchical features from the input image, which are essential for detecting and classifying objects. In both YOLO and Faster RCNN, this feature map serves as the input for subsequent processing stages.
- Backbone in YOLO - YOLO's design emphasizes speed and efficiency. The backbone network needs to balance the depth and complexity to extract useful features without significantly slowing down inference.
- Backbone in Faster RCNN - Faster RCNN separates the region proposal and classification stages, which can lead to higher accuracy but at the cost of speed.

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

PRACTICAL

1. How do you load and run inference on a custom image using the YOLOv8 model (labeled as YOLOv9)?

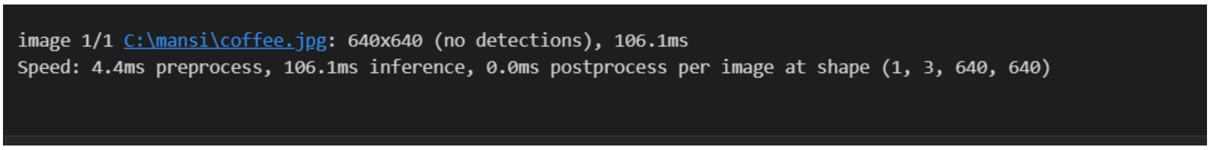
➔ CODE:



```
#1.How do you load and run inference on a custom image using the YOLOv8 model (labeled as YOLOv9)?
from ultralytics import YOLO
import cv2
import matplotlib.pyplot as plt
# Step 1: Load the YOLOv8 model
model = YOLO("yolov8n.pt")  # Replace 'yolov8n.pt' with your model weights file
# Step 2: Load a custom image
image_path = r"C:/mansi/coffee.jpg"  # Provide the path to your image
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB for visualization
# Step 3: Run inference
results = model(image_path)  # Perform inference
# Step 4: Visualize results
# Extract predictions for the image
detections = results[0]
# Draw bounding boxes and labels
for box in detections.boxes:
    x1, y1, x2, y2 = map(int, box.xyxy[0])  # Bounding box coordinates
    confidence = box.conf[0]  # Confidence score
    class_id = int(box.cls[0])  # Class ID
    label = detections.names[class_id]  # Class label
    # Draw rectangle and text
    cv2.rectangle(image_rgb, (x1, y1), (x2, y2), (0, 255, 0), 2)
    cv2.putText(image_rgb, f"{label} {confidence:.2f}", (x1, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
# Step 5: Display the image
plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```

✓ 0.2s

OUTPUT:

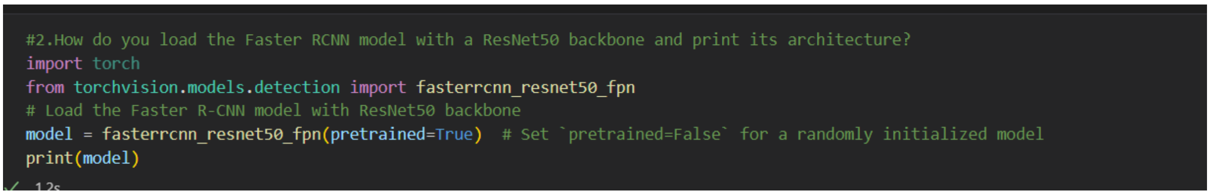


```
image 1/1 C:\mansi\coffee.jpg: 640x640 (no detections), 106.1ms
Speed: 4.4ms preprocess, 106.1ms inference, 0.0ms postprocess per image at shape (1, 3, 640, 640)
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

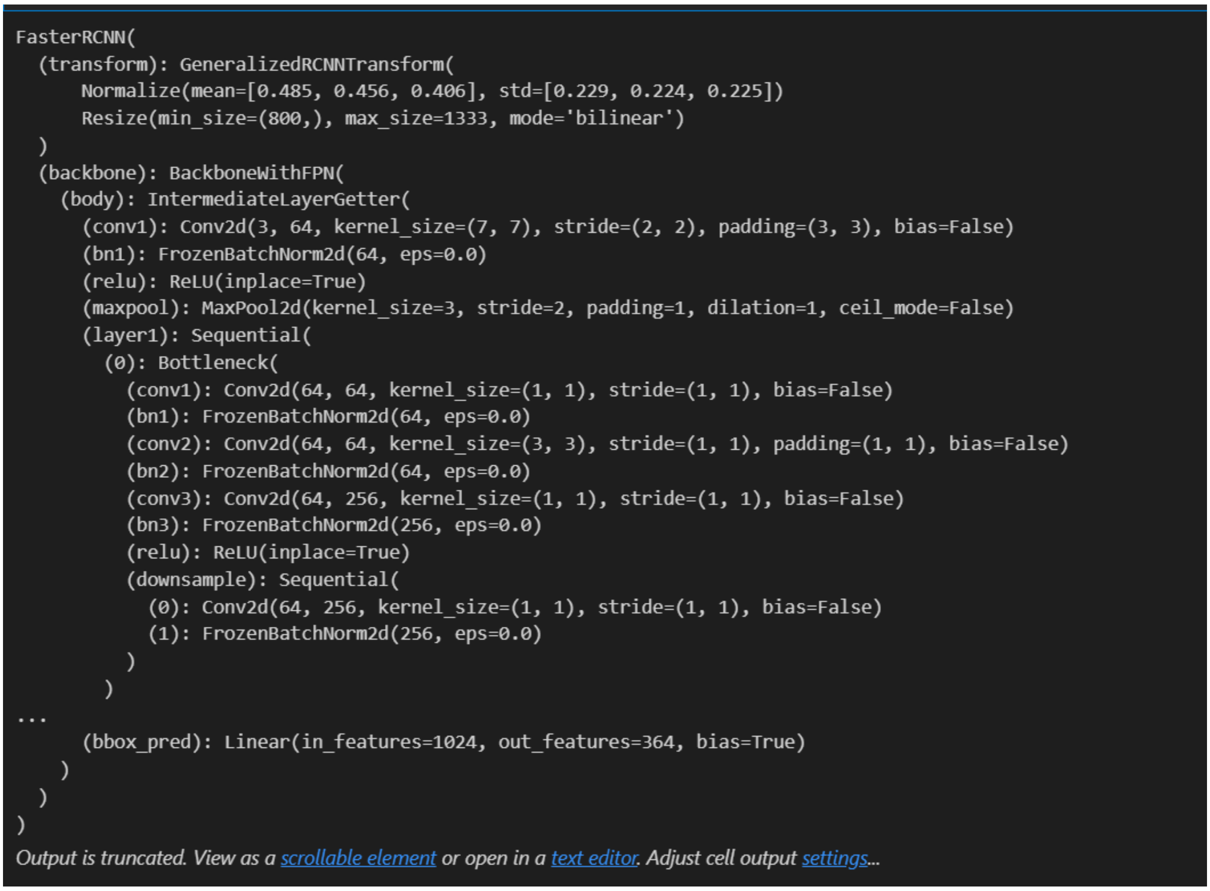
2. How do you load the Faster RCNN model with a ResNet50 backbone and print its architecture?

➔ CODE:



```
#2.How do you load the Faster RCNN model with a ResNet50 backbone and print its architecture?
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
# Load the Faster R-CNN model with ResNet50 backbone
model = fasterrcnn_resnet50_fpn(pretrained=True)  # Set `pretrained=False` for a randomly initialized model
print(model)
```

OUTPUT:



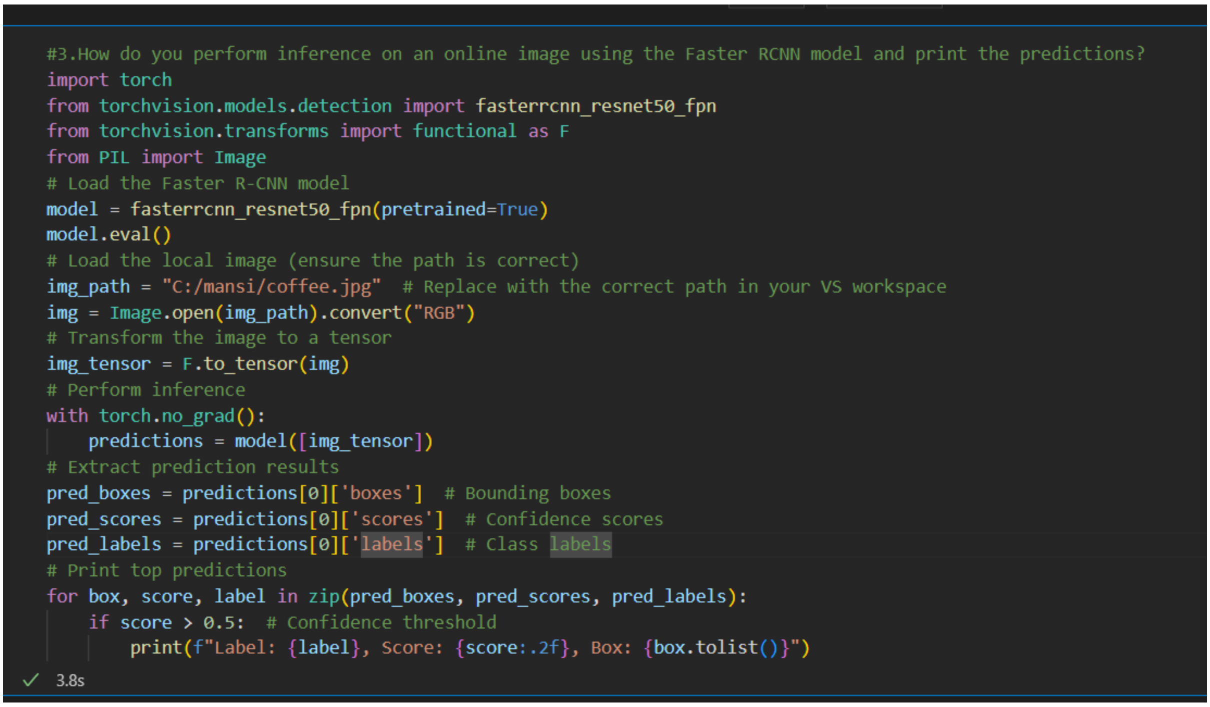
```
FasterRCNN(
  (transform): GeneralizedRCNNTransform(
      Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
      Resize(min_size=(800,), max_size=1333, mode='bilinear')
  )
  (backbone): BackboneWithFPN(
    (body): IntermediateLayerGetter(
      (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
      (bn1): FrozenBatchNorm2d(64, eps=0.0)
      (relu): ReLU(inplace=True)
      (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
      (layer1): Sequential(
        (0): Bottleneck(
          (conv1): Conv2d(64, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn1): FrozenBatchNorm2d(64, eps=0.0)
          (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
          (bn2): FrozenBatchNorm2d(64, eps=0.0)
          (conv3): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (bn3): FrozenBatchNorm2d(256, eps=0.0)
          (relu): ReLU(inplace=True)
          (downsample): Sequential(
            (0): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): FrozenBatchNorm2d(256, eps=0.0)
          )
        )
...
      (bbox_pred): Linear(in_features=1024, out_features=364, bias=True)
    )
  )
)
```

Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

3. How do you perform inference on an online image using the Faster RCNN model and print the predictions?

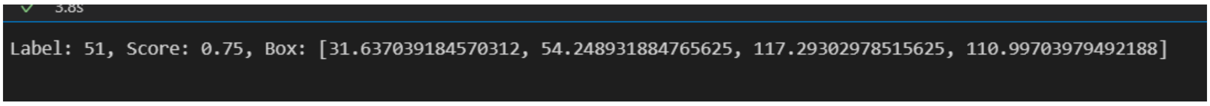
➔ CODE:



```
#3.How do you perform inference on an online image using the Faster RCNN model and print the predictions?
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.transforms import functional as F
from PIL import Image
# Load the Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()
# Load the local image (ensure the path is correct)
img_path = "C:/mansi/coffee.jpg"  # Replace with the correct path in your VS workspace
img = Image.open(img_path).convert("RGB")
# Transform the image to a tensor
img_tensor = F.to_tensor(img)
# Perform inference
with torch.no_grad():
    predictions = model([img_tensor])
# Extract prediction results
pred_boxes = predictions[0]['boxes']  # Bounding boxes
pred_scores = predictions[0]['scores']  # Confidence scores
pred_labels = predictions[0]['labels']  # Class labels
# Print top predictions
for box, score, label in zip(pred_boxes, pred_scores, pred_labels):
    if score > 0.5:  # Confidence threshold
        print(f"Label: {label}, Score: {score:.2f}, Box: {box.tolist()}")
```

✓ 3.8s

OUTPUT:

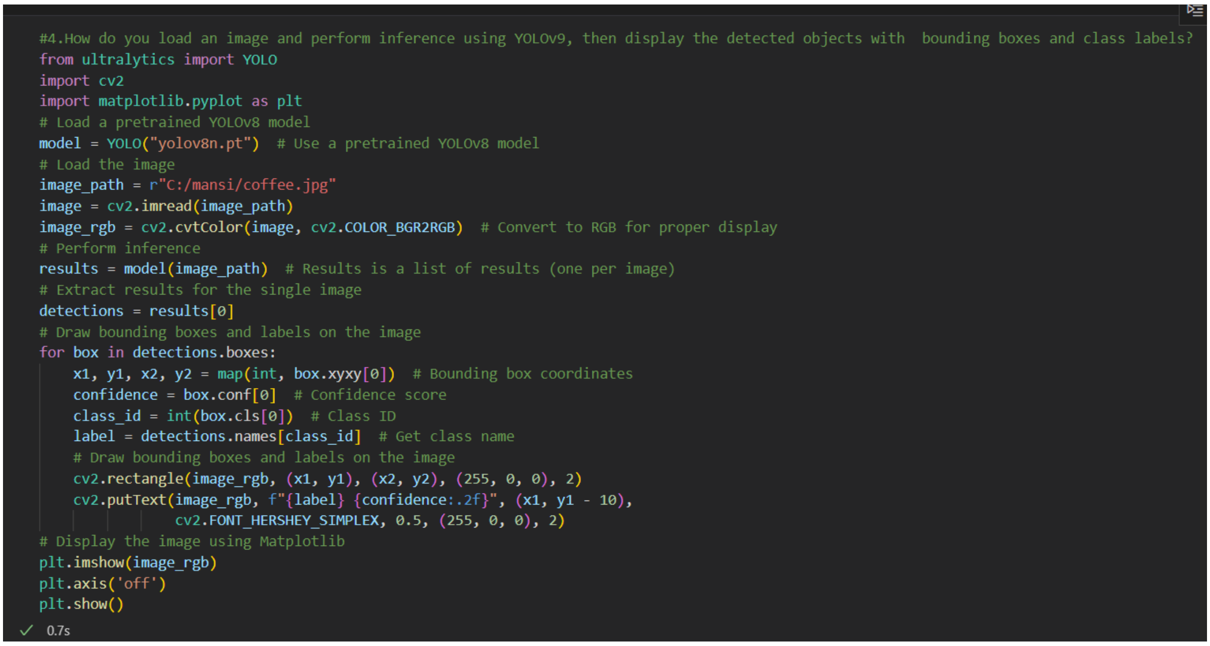


```
Label: 51, Score: 0.75, Box: [31.637039184570312, 54.248931884765625, 117.29302978515625, 110.99703979492188]
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

4. How do you load an image and perform inference using YOLOv9, then display the detected objects with bounding boxes and class labels?

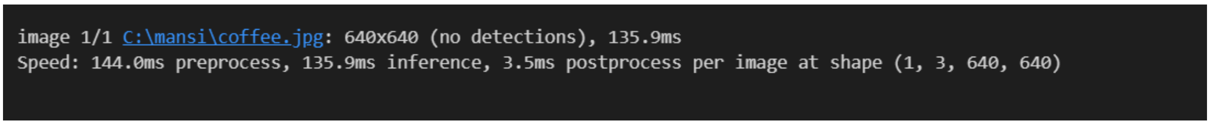
   ➔ CODE:



```
#4.How do you load an image and perform inference using YOLOv9, then display the detected objects with  bounding boxes and class labels?
from ultralytics import YOLO
import cv2
import matplotlib.pyplot as plt
# Load a pretrained YOLOv8 model
model = YOLO("yolov8n.pt")  # Use a pretrained YOLOv8 model
# Load the image
image_path = r"C:/mansi/coffee.jpg"
image = cv2.imread(image_path)
image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)  # Convert to RGB for proper display
# Perform inference
results = model(image_path)  # Results is a list of results (one per image)
# Extract results for the single image
detections = results[0]
# Draw bounding boxes and labels on the image
for box in detections.boxes:
    x1, y1, x2, y2 = map(int, box.xyxy[0])  # Bounding box coordinates
    confidence = box.conf[0]  # Confidence score
    class_id = int(box.cls[0])  # Class ID
    label = detections.names[class_id]  # Get class name
    # Draw bounding boxes and labels on the image
    cv2.rectangle(image_rgb, (x1, y1), (x2, y2), (255, 0, 0), 2)
    cv2.putText(image_rgb, f"{label} {confidence:.2f}", (x1, y1 - 10),
                cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
# Display the image using Matplotlib
plt.imshow(image_rgb)
plt.axis('off')
plt.show()
```

✓ 0.7s

OUTPUT:

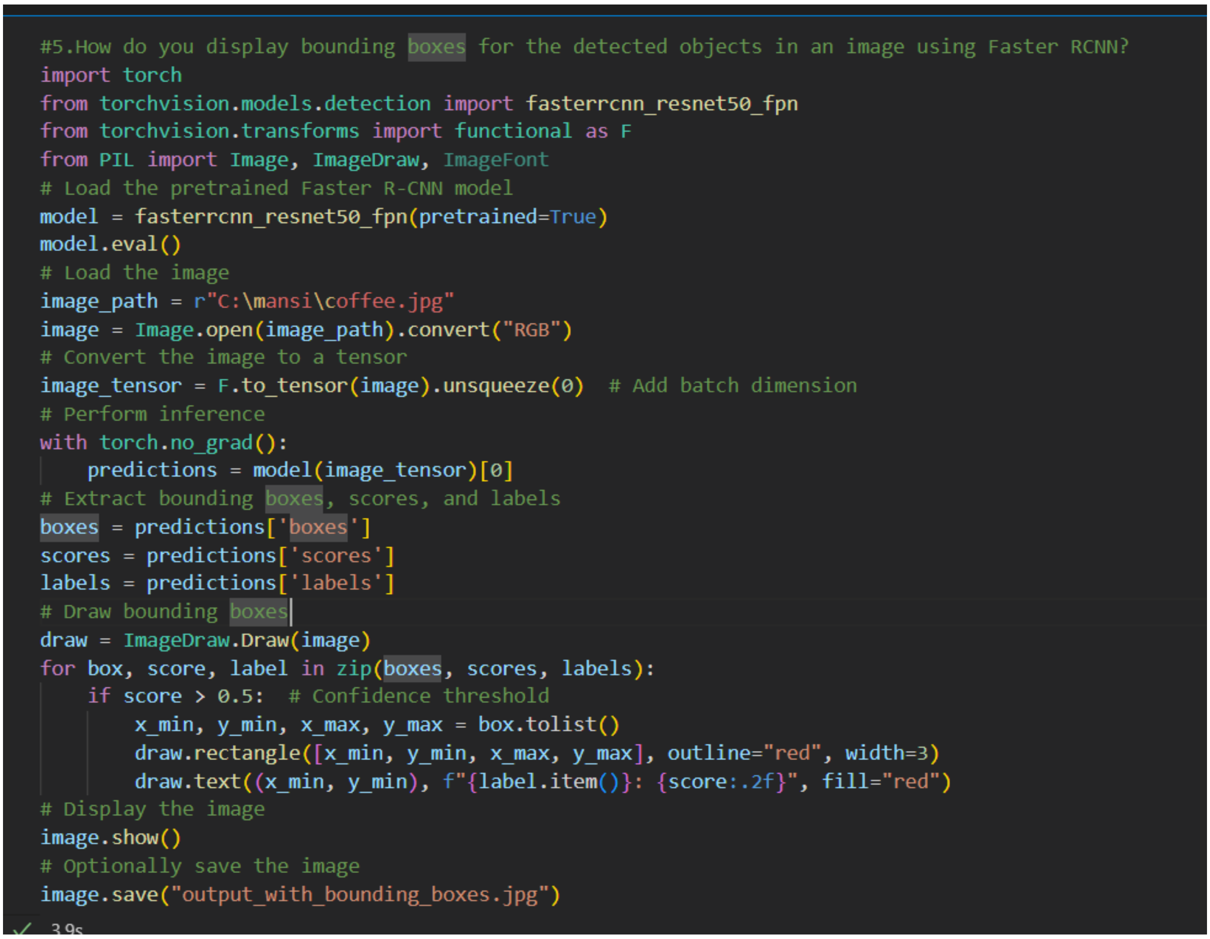


```
image 1/1 C:\mansi\coffee.jpg: 640x640 (no detections), 135.9ms
Speed: 144.0ms preprocess, 135.9ms inference, 3.5ms postprocess per image at shape (1, 3, 640, 640)
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

5. How do you display bounding boxes for the detected objects in an image using Faster RCNN?

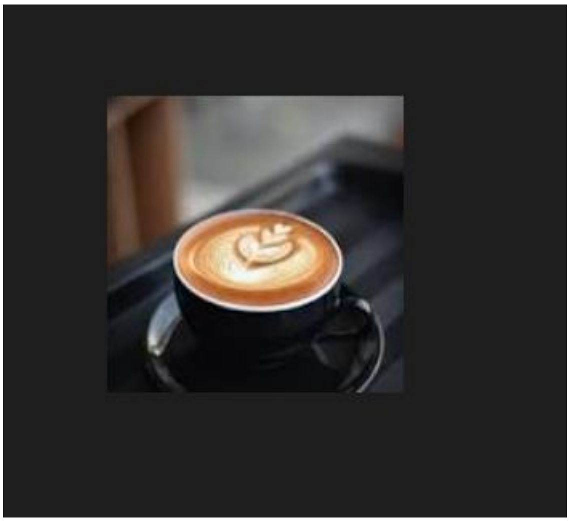
➔ CODE:



```
#5.How do you display bounding boxes for the detected objects in an image using Faster RCNN?
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.transforms import functional as F
from PIL import Image, ImageDraw, ImageFont
# Load the pretrained Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()
# Load the image
image_path = r"C:\mansi\coffee.jpg"
image = Image.open(image_path).convert("RGB")
# Convert the image to a tensor
image_tensor = F.to_tensor(image).unsqueeze(0)  # Add batch dimension
# Perform inference
with torch.no_grad():
    predictions = model(image_tensor)[0]
# Extract bounding boxes, scores, and labels
boxes = predictions['boxes']
scores = predictions['scores']
labels = predictions['labels']
# Draw bounding boxes
draw = ImageDraw.Draw(image)
for box, score, label in zip(boxes, scores, labels):
    if score > 0.5:  # Confidence threshold
        x_min, y_min, x_max, y_max = box.tolist()
        draw.rectangle([x_min, y_min, x_max, y_max], outline="red", width=3)
        draw.text((x_min, y_min), f"{label.item()}: {score:.2f}", fill="red")
# Display the image
image.show()
# Optionally save the image
image.save("output_with_bounding_boxes.jpg")
```

✓ 3.9s

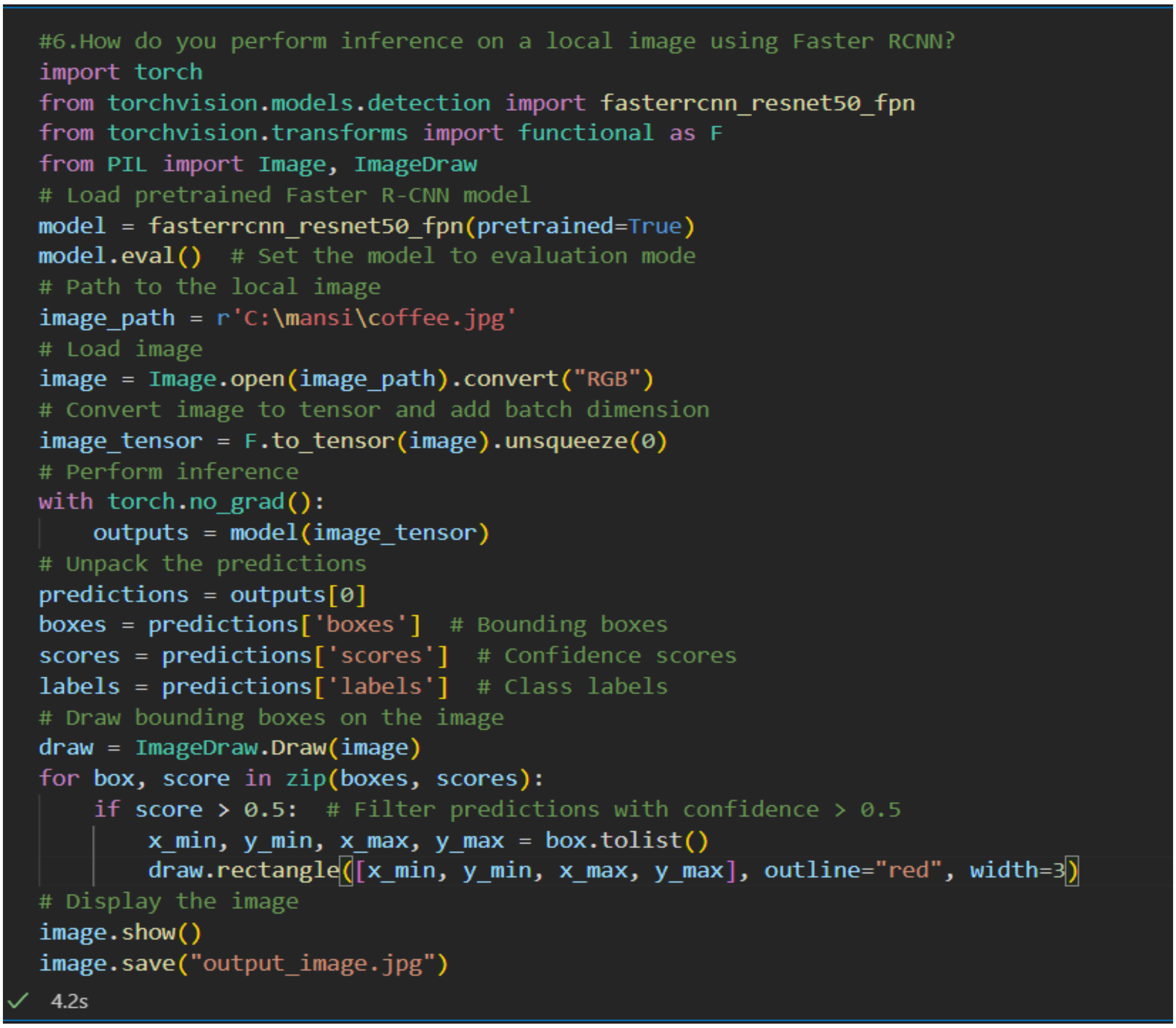
OUTPUT:



Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

6. How do you perform inference on a local image using Faster RCNN?

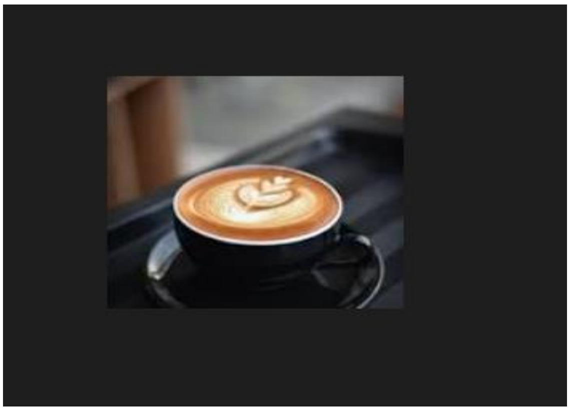
➔ CODE:



```
#6.How do you perform inference on a local image using Faster RCNN?
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
from torchvision.transforms import functional as F
from PIL import Image, ImageDraw
# Load pretrained Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()  # Set the model to evaluation mode
# Path to the local image
image_path = r'C:\mansi\coffee.jpg'
# Load image
image = Image.open(image_path).convert("RGB")
# Convert image to tensor and add batch dimension
image_tensor = F.to_tensor(image).unsqueeze(0)
# Perform inference
with torch.no_grad():
    outputs = model(image_tensor)
# Unpack the predictions
predictions = outputs[0]
boxes = predictions['boxes']  # Bounding boxes
scores = predictions['scores']  # Confidence scores
labels = predictions['labels']  # Class labels
# Draw bounding boxes on the image
draw = ImageDraw.Draw(image)
for box, score in zip(boxes, scores):
    if score > 0.5:  # Filter predictions with confidence > 0.5
        x_min, y_min, x_max, y_max = box.tolist()
        draw.rectangle([x_min, y_min, x_max, y_max], outline="red", width=3)
# Display the image
image.show()
image.save("output_image.jpg")
```

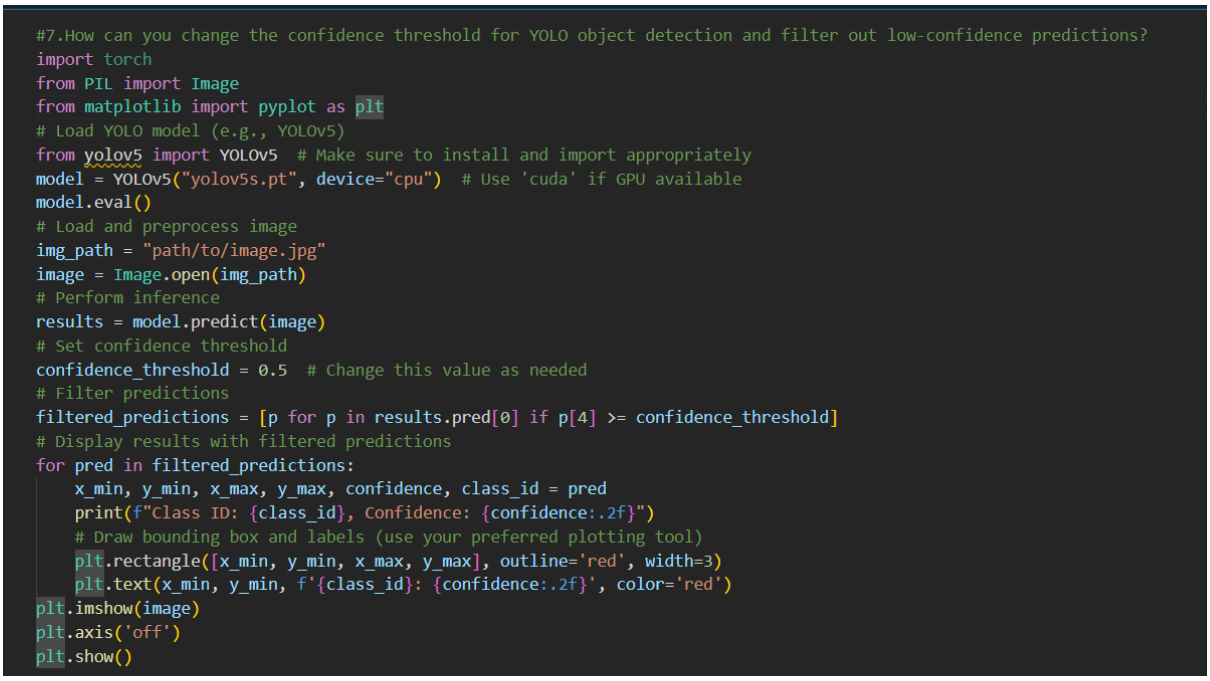
✓ 4.2s

OUTPUT:



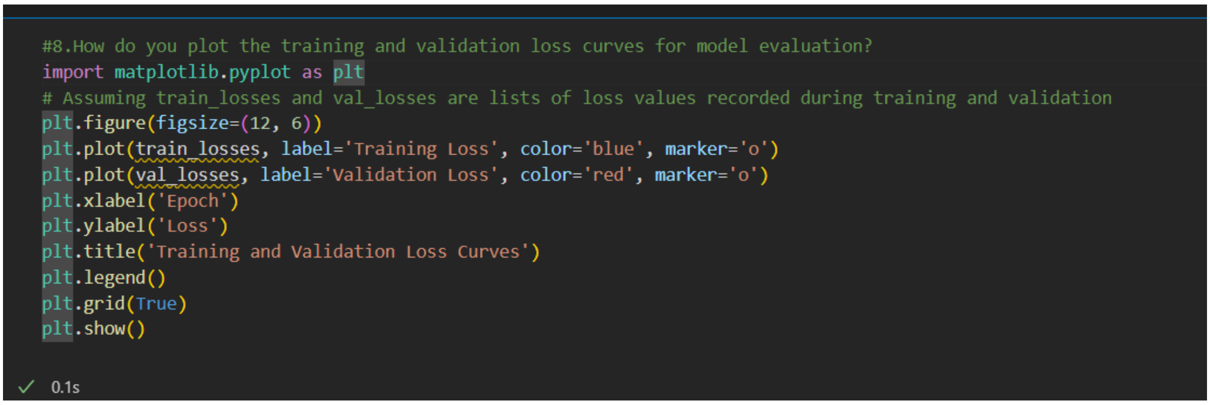
Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

7. How can you change the confidence threshold for YOLO object detection and filter out low-confidence predictions?



```
#7.How can you change the confidence threshold for YOLO object detection and filter out low-confidence predictions?
import torch
from PIL import Image
from matplotlib import pyplot as plt
# Load YOLO model (e.g., YOLOv5)
from yolov5 import YOLOv5  # Make sure to install and import appropriately
model = YOLOv5("yolov5s.pt", device="cpu")  # Use 'cuda' if GPU available
model.eval()
# Load and preprocess image
img_path = "path/to/image.jpg"
image = Image.open(img_path)
# Perform inference
results = model.predict(image)
# Set confidence threshold
confidence_threshold = 0.5  # Change this value as needed
# Filter predictions
filtered_predictions = [p for p in results.pred[0] if p[4] >= confidence_threshold]
# Display results with filtered predictions
for pred in filtered_predictions:
    x_min, y_min, x_max, y_max, confidence, class_id = pred
    print(f"Class ID: {class_id}, Confidence: {confidence:.2f}")
    # Draw bounding box and labels (use your preferred plotting tool)
    plt.rectangle([x_min, y_min, x_max, y_max], outline='red', width=3)
    plt.text(x_min, y_min, f'{class_id}: {confidence:.2f}', color='red')
plt.imshow(image)
plt.axis('off')
plt.show()
```

8. How do you plot the training and validation loss curves for model evaluation?

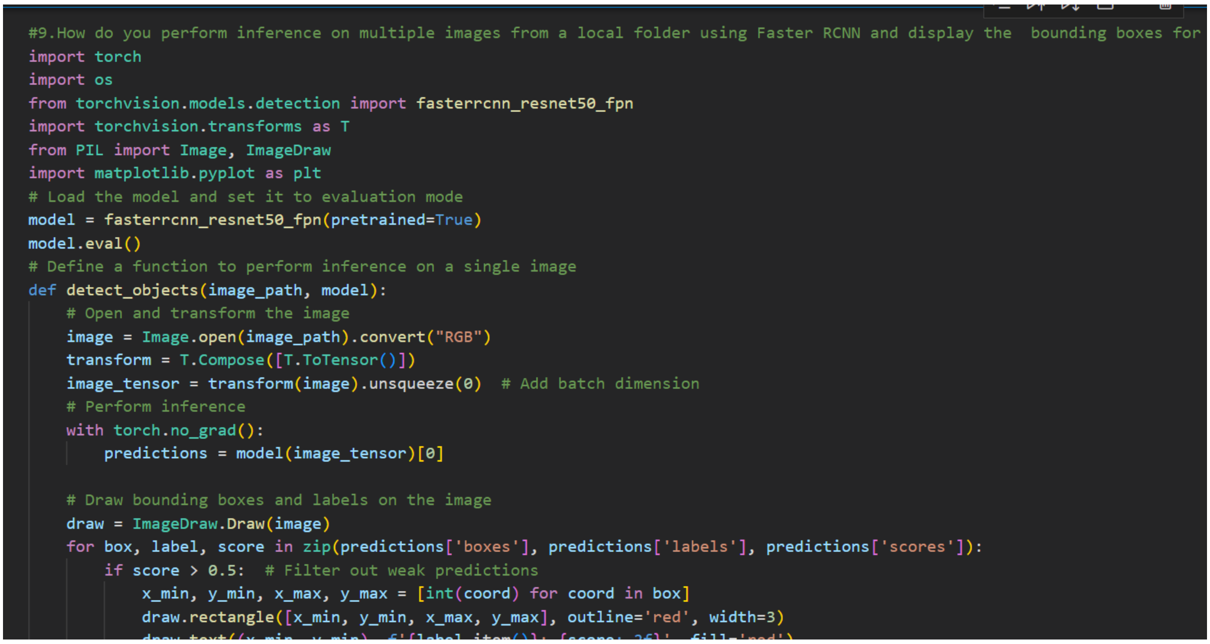


```
#8.How do you plot the training and validation loss curves for model evaluation?
import matplotlib.pyplot as plt
# Assuming train_losses and val_losses are lists of loss values recorded during training and validation
plt.figure(figsize=(12, 6))
plt.plot(train_losses, label='Training Loss', color='blue', marker='o')
plt.plot(val_losses, label='Validation Loss', color='red', marker='o')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss Curves')
plt.legend()
plt.grid(True)
plt.show()
```

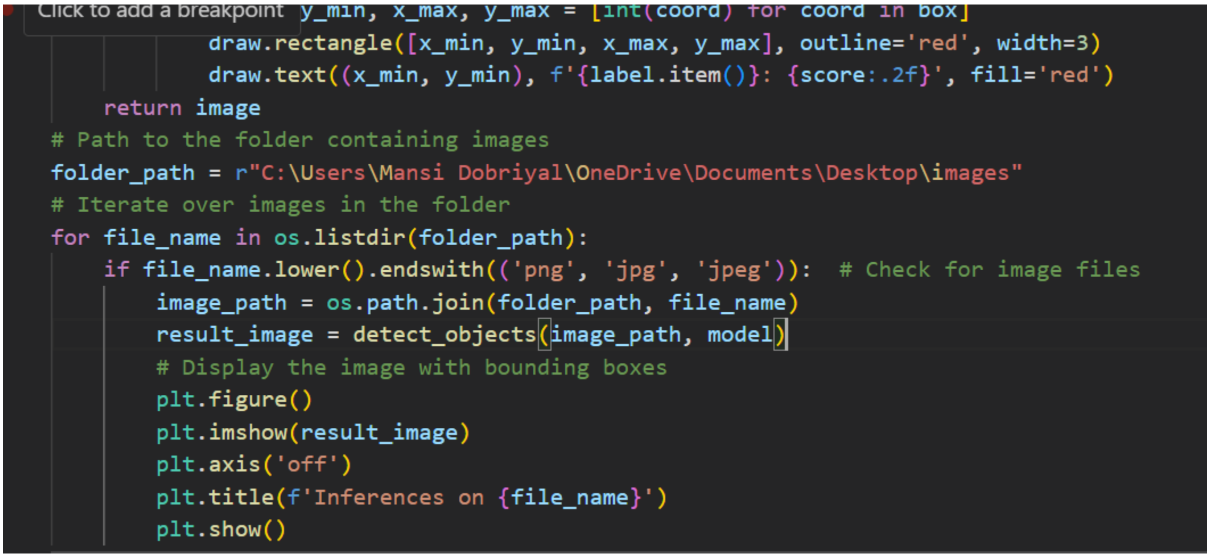
✓ 0.1s

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

9. How do you perform inference on multiple images from a local folder using Faster RCNN and display the bounding boxes for each?



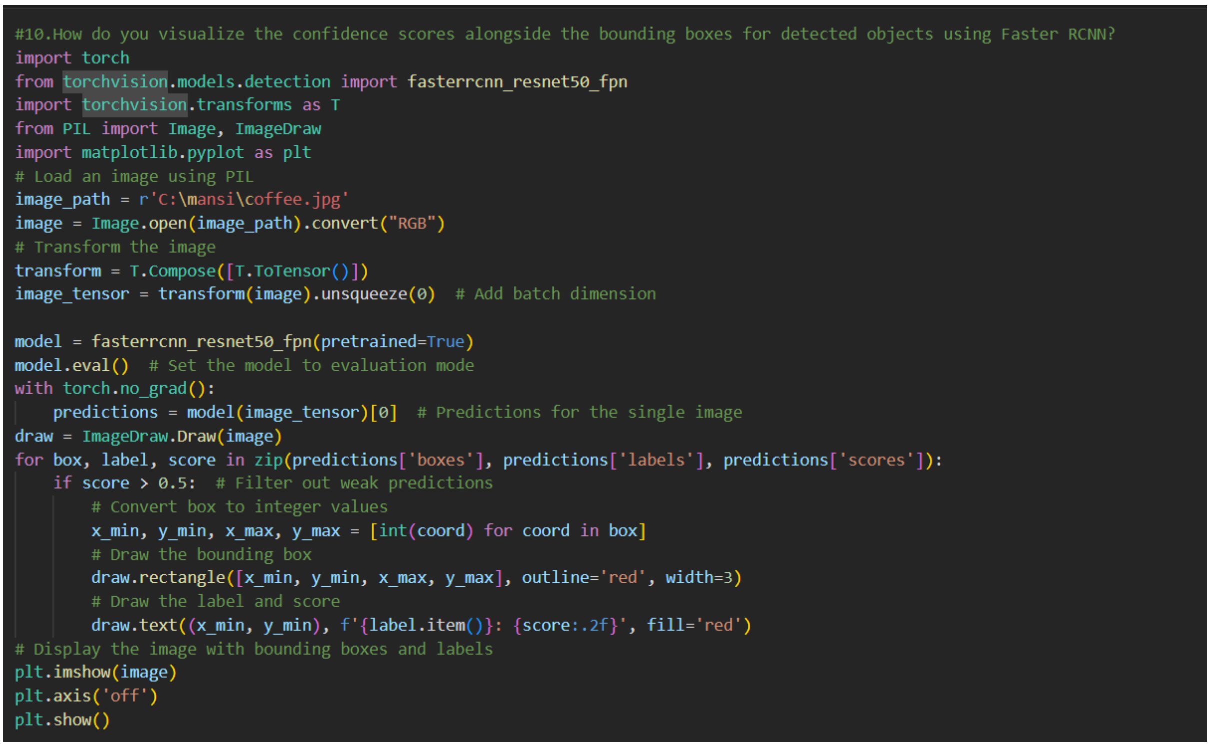
```python
#9.How do you perform inference on multiple images from a local folder using Faster RCNN and display the  bounding boxes for
import torch
import os
from torchvision.models.detection import fasterrcnn_resnet50_fpn
import torchvision.transforms as T
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
# Load the model and set it to evaluation mode
model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()
# Define a function to perform inference on a single image
def detect_objects(image_path, model):
    # Open and transform the image
    image = Image.open(image_path).convert("RGB")
    transform = T.Compose([T.ToTensor()])
    image_tensor = transform(image).unsqueeze(0)  # Add batch dimension
    # Perform inference
    with torch.no_grad():
        predictions = model(image_tensor)[0]

    # Draw bounding boxes and labels on the image
    draw = ImageDraw.Draw(image)
    for box, label, score in zip(predictions['boxes'], predictions['labels'], predictions['scores']):
        if score > 0.5:  # Filter out weak predictions
            x_min, y_min, x_max, y_max = [int(coord) for coord in box]
            draw.rectangle([x_min, y_min, x_max, y_max], outline='red', width=3)
            draw.text((x_min, y_min), f'{label.item()}: {score:.2f}', fill='red')
```



```python
            y_min, x_max, y_max = [int(coord) for coord in box]
            draw.rectangle([x_min, y_min, x_max, y_max], outline='red', width=3)
            draw.text((x_min, y_min), f'{label.item()}: {score:.2f}', fill='red')
    return image
# Path to the folder containing images
folder_path = r"C:\Users\Mansi Dobriyal\OneDrive\Documents\Desktop\images"
# Iterate over images in the folder
for file_name in os.listdir(folder_path):
    if file_name.lower().endswith(('png', 'jpg', 'jpeg')):  # Check for image files
        image_path = os.path.join(folder_path, file_name)
        result_image = detect_objects(image_path, model)
        # Display the image with bounding boxes
        plt.figure()
        plt.imshow(result_image)
        plt.axis('off')
        plt.title(f'Inferences on {file_name}')
        plt.show()
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

10. How do you visualize the confidence scores alongside the bounding boxes for detected objects using Faster RCNN?

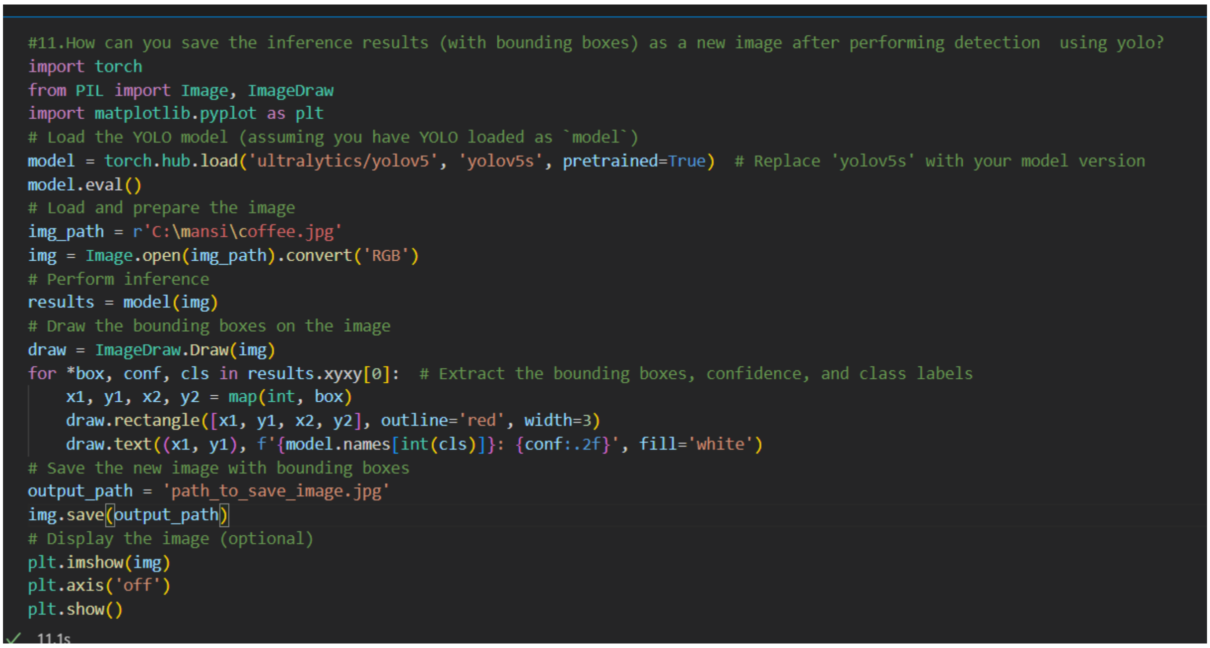


```
#10.How do you visualize the confidence scores alongside the bounding boxes for detected objects using Faster RCNN?
import torch
from torchvision.models.detection import fasterrcnn_resnet50_fpn
import torchvision.transforms as T
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
# Load an image using PIL
image_path = r'C:\mansi\coffee.jpg'
image = Image.open(image_path).convert("RGB")
# Transform the image
transform = T.Compose([T.ToTensor()])
image_tensor = transform(image).unsqueeze(0)  # Add batch dimension

model = fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()  # Set the model to evaluation mode
with torch.no_grad():
    predictions = model(image_tensor)[0]  # Predictions for the single image
draw = ImageDraw.Draw(image)
for box, label, score in zip(predictions['boxes'], predictions['labels'], predictions['scores']):
    if score > 0.5:  # Filter out weak predictions
        # Convert box to integer values
        x_min, y_min, x_max, y_max = [int(coord) for coord in box]
        # Draw the bounding box
        draw.rectangle([x_min, y_min, x_max, y_max], outline='red', width=3)
        # Draw the label and score
        draw.text((x_min, y_min), f'{label.item()}: {score:.2f}', fill='red')
# Display the image with bounding boxes and labels
plt.imshow(image)
plt.axis('off')
plt.show()
```

Name – Mansi
Course – Data Science with Generative AI
E-mail – mansidobriyal50@gmail.com
Phone no. - 9311453903

11. How can you save the interference results(with bounding boxes) as a new image after performing detection using yolo?

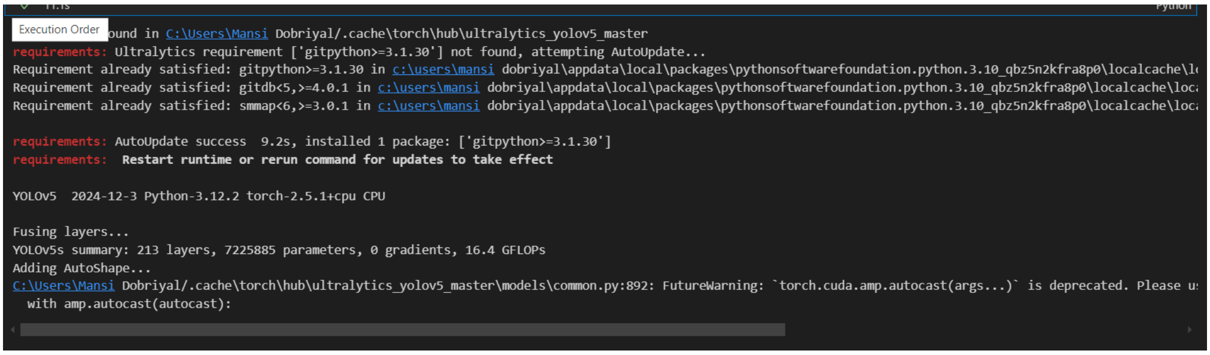
→ CODE:



```python
#11.How can you save the inference results (with bounding boxes) as a new image after performing detection  using yolo?
import torch
from PIL import Image, ImageDraw
import matplotlib.pyplot as plt
# Load the YOLO model (assuming you have YOLO loaded as `model`)
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)  # Replace 'yolov5s' with your model version
model.eval()
# Load and prepare the image
img_path = r'C:\mansi\coffee.jpg'
img = Image.open(img_path).convert('RGB')
# Perform inference
results = model(img)
# Draw the bounding boxes on the image
draw = ImageDraw.Draw(img)
for *box, conf, cls in results.xyxy[0]:  # Extract the bounding boxes, confidence, and class labels
    x1, y1, x2, y2 = map(int, box)
    draw.rectangle([x1, y1, x2, y2], outline='red', width=3)
    draw.text((x1, y1), f'{model.names[int(cls)]}: {conf:.2f}', fill='white')
# Save the new image with bounding boxes
output_path = 'path_to_save_image.jpg'
img.save(output_path)
# Display the image (optional)
plt.imshow(img)
plt.axis('off')
plt.show()
```

✓ 11.1s

OUTPUT:



```
Execution Order ound in C:\Users\Mansi Dobriyal/.cache\torch\hub\ultralytics_yolov5_master
requirements: Ultralytics requirement ['gitpython>=3.1.30'] not found, attempting AutoUpdate...
Requirement already satisfied: gitpython>=3.1.30 in c:\users\mansi dobriyal\appdata\local\packages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\l
Requirement already satisfied: gitdb<5,>=4.0.1 in c:\users\mansi dobriyal\appdata\local\packages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\loc
Requirement already satisfied: smmap<6,>=3.0.1 in c:\users\mansi dobriyal\appdata\local\packages\pythonsoftwarefoundation.python.3.10_qbz5n2kfra8p0\localcache\loc

requirements: AutoUpdate success  9.2s, installed 1 package: ['gitpython>=3.1.30']
requirements:  Restart runtime or rerun command for updates to take effect

YOLOv5  2024-12-3 Python-3.12.2 torch-2.5.1+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
Adding AutoShape...
C:\Users\Mansi Dobriyal/.cache\torch\hub\ultralytics_yolov5_master\models\common.py:892: FutureWarning: `torch.cuda.amp.autocast(args...)` is deprecated. Please u
  with amp.autocast(autocast):
```