Machine Problem 5 - Object Detection and Recognition using YOLO

Objective:

To implement real-time object detection using the YOLO (You Only Look Once) model and gain hands-on experience in loading pre-trained models, processing images, and visualizing results.

Tasks:

1. Model Loading: Use TensorFlow to load a pre-trained YOLO model.
2. Image Input: Select an image that contains multiple objects.
3. Object Detection: Feed the selected image to the YOLO model to detect various objects within it.
4. Visualization: Display the detected objects using bounding boxes and class labels.
5. Testing: Test the model on at least three different images to compare its performance and observe its accuracy.
6. Performance Analysis: Document your observations on the model's speed and accuracy, and discuss how YOLO's single-pass detection impacts its real-time capabilities.

Code:

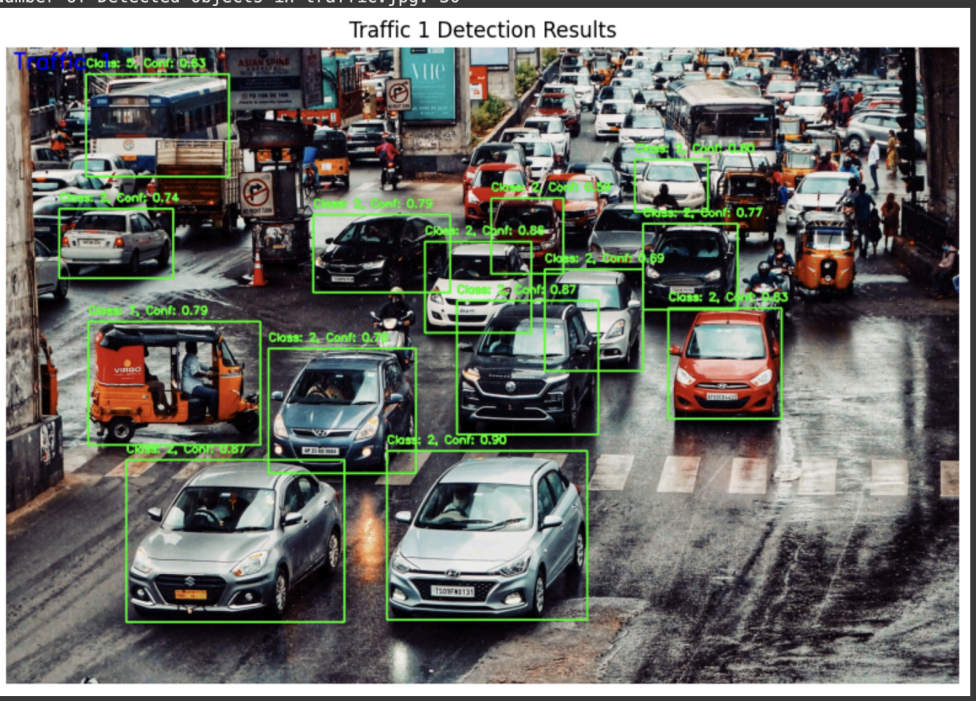
```
!pip install ultralytics
import cv2  # OpenCV for image processing
import time  # Time module to measure inference time
import matplotlib.pyplot as plt  # Matplotlib for displaying images
from ultralytics import YOLO  # Import the YOLO class from the ultralytics package

# Load the pre-trained YOLO model
model = YOLO('yolov8n.pt')  # Use a pre-trained YOLOv8 model

# List of image paths to test and their corresponding labels
image_paths = ['traffic.jpg', 'traffic 1.jpg', 'traffic 2.jpg']
labels = ['Traffic 1', 'Traffic 2', 'Traffic 3']
```

```
# Function to perform object detection, measure performance, and display
images
def analyze_performance(image_path, label):
   # Load an image from the specified path
   image = cv2.imread(image_path)

   # Check if the image was loaded successfully
   if image is None:
       raise FileNotFoundError(f"Image not found: {image_path}")

   # Measure the inference time for object detection
   start_time = time.time()  # Start timer
   results = model(image)  # Perform object detection
   end_time = time.time()  # End timer

   # Calculate the time taken for inference
   inference_time = end_time - start_time
   print(f"Inference Time for {image_path}: {inference_time:.4f} seconds")

   # Initialize a counter for detected objects
   detected_objects = 0
   # Process the detection results
   for result in results:
        boxes = result.boxes  # Get the bounding boxes from the detection
results
         detected_objects += len(boxes)  # Count the number of detected
boxes

       # Iterate through each detected box
       for box in boxes:
           # Get bounding box coordinates and confidence score
           x1, y1, x2, y2 = box.xyxy[0].numpy()
           conf = box.conf[0].item()  # Get the confidence score
           class_id = int(box.cls[0].item())  # Get the class ID

           # Filter out low confidence detections
           if conf > 0.5:
               # Draw bounding box and label on the image
                       cv2.rectangle(image, (int(x1), int(y1)), (int(x2),
int(y2)), (0, 255, 0), 2)
```

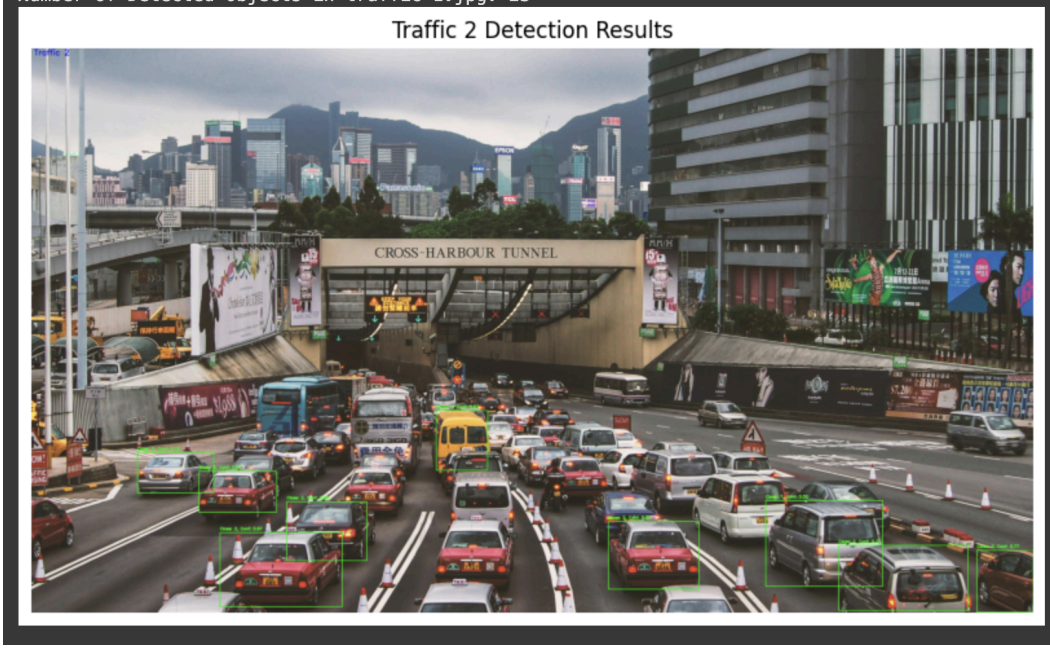
```
                cv2.putText(image, f"Class: {class_id}, Conf: {conf:.2f}",
                                                    (int(x1), int(y1) - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)

    # Output the number of detected objects
            print(f"Number of Detected Objects in {image_path}:
{detected_objects}")

    # Adding a label to the top-left corner of the image
     cv2.putText(image, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,
0, 0), 2)

    # Display the image with detections using matplotlib
    plt.figure(figsize=(10, 10))  # Set figure size for the plot
     plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))  # Convert BGR to
RGB for correct color display
    plt.axis('off')  # Hide the axis
    plt.title(f'{label} Detection Results')  # Set the title of the plot
    plt.show()  # Show the plot

# Analyze performance for each image with corresponding labels
for img_path, label in zip(image_paths, labels):
    analyze_performance(img_path, label)
```

Output:

```
0: 448x640 4 persons, 21 cars, 2 buss, 3 trucks, 212.8ms
Speed: 7.9ms preprocess, 212.8ms inference, 3.6ms postprocess per image at shape (1, 3, 448, 640)
Inference Time for traffic.jpg: 0.4426 seconds
Number of Detected Objects in traffic.jpg: 30
```




```
0: 384x640 18 cars, 2 buss, 2 trucks, 1 traffic light, 320.6ms
Speed: 8.0ms preprocess, 320.6ms inference, 2.1ms postprocess per image at shape (1, 3, 384, 640)
Inference Time for traffic 1.jpg: 0.3440 seconds
Number of Detected Objects in traffic 1.jpg: 23
```



```
0: 448x640 15 cars, 1 bus, 1 truck, 290.6ms
Speed: 12.7ms preprocess, 290.6ms inference, 2.0ms postprocess per image at shape (1, 3, 448, 640)
Inference Time for traffic 2.jpg: 0.3303 seconds
Number of Detected Objects in traffic 2.jpg: 17
```

Explanation:

This machine problem is designed to perform **object detection using a pre-trained YOLO (You Only Look Once) model**, **specifically the YOLOv8 model**. The primary objective is to analyze images of traffic scenes and identify various objects within them, such as cars, buses, and pedestrians. The process involves several key steps, which are executed systematically.

Importing Libraries:

The code begins by importing necessary libraries. OpenCV is used for image processing, while Matplotlib is employed for displaying the processed images. The time module measures how long it takes to detect objects in each image, and the ultralytics package provides access to the YOLO model.

Loading the Model:

The pre-trained YOLOv8 model is loaded into the program using YOLO('yolov8n.pt'). This model has been trained on a large dataset and can detect various objects with high accuracy.

Image Paths and Labels:

The code defines a list of image paths and their corresponding labels. This allows the program to know which images to analyze and how to title the output.

Defining the Analysis Function:

A function named analyze_performance is created to handle the object detection process. Within this function:

- An image is loaded from the specified path using OpenCV. If the image is not found, an error is raised.
- The inference time is measured by recording the time before and after the object detection process.
- The model processes the image and identifies objects, counting how many were detected.
- For each detected object, the bounding box coordinates, confidence score, and class ID are extracted. Only detections with a confidence score greater than 0.5 are considered valid. The bounding box and label are drawn on the image to visualize the detections.

Finally, the number of detected objects is printed, and the processed image is displayed using Matplotlib.

Analyzing Performance for Each Image:

The program iterates through the list of images and their labels, calling the analyze_performance function for each one. This step enables the program to apply the same object detection process across multiple images.

Summary of Comparison for Each Image

Traffic 1:

The first image, "Traffic 1," resulted in the detection of 30 objects, including 4 persons, 21 cars, 2 buses, and 3 trucks. The total inference time was recorded at 0.4426

seconds, with a breakdown showing 7.9 milliseconds for preprocessing, 212.8 milliseconds for inference, and 3.6 milliseconds for postprocessing. This performance indicates the model's ability to effectively handle diverse object types in a busy traffic scenario.

Traffic 2:

In the second image, "Traffic 2," the model detected 23 objects, which included 18 cars, 2 buses, 2 trucks, and 1 traffic light. The inference time for this image was 0.3440 seconds, with 8.0 milliseconds for preprocessing, 320.6 milliseconds for inference, and 2.1 milliseconds for postprocessing. Although fewer objects were detected compared to the first image, the model demonstrated a quick inference time, reflecting its robustness in processing traffic scenes.

Traffic 3:

The third image, "Traffic 3," yielded 17 detected objects, consisting of 15 cars, 1 bus, and 1 truck. The inference time for this image was the fastest at 0.3303 seconds, with 12.7 milliseconds for preprocessing, 290.6 milliseconds for inference, and 2.0 milliseconds for postprocessing. This scene had fewer detected objects than the previous images, yet the model maintained a high level of speed and accuracy.

Overall, the YOLO model showcased consistent performance across all three traffic images, demonstrating its effectiveness in object detection while maintaining efficient processing times. This efficiency makes it suitable for applications such as real-time traffic monitoring and management.

- *End of Documentation* -