4B BAUTISTA MP5

Machine Problem: 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.

Task:

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.

Install a pre trained YOLO Model for the activity .

!pip install ultralytics

Load the pre-trained model:

# 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 1.jpg', 'traffic 2.jpg', 'traffic 3.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)