Author: Elena Pashkova

## Project Summary:

This project focused on implementing and evaluating an object detection pipeline using a pre-trained deep learning model. The key objectives were:

1. **Object Detection Implementation**: A pre-trained model was used to detect objects in images, with bounding boxes and confidence scores visualized to assess detection quality.
2. **Evaluation Metrics**: Performance metrics such as Intersection over Union (IoU), precision, recall, and F1-score were calculated. IoU was used to compare predicted bounding boxes with ground truth, and thresholds for confidence and IoU were adjusted to optimize evaluation.
3. **Insights**:
    - The model performed well for high-confidence predictions, achieving strong alignment between predictions and ground truth at adjusted thresholds.
    - However, some predictions showed lower IoU values, highlighting challenges in detecting certain objects or bounding box misalignments.
4. **Outcomes**:
    - Precision, recall, and F1-score reached perfect values under specific thresholds, demonstrating the model's ability to detect the target object accurately.
    - Visualizations effectively showcased the predictions and ground truth, facilitating a deeper understanding of the model's strengths and weaknesses.

Overall, this project successfully implemented object detection, demonstrated its evaluation process, and highlighted the importance of fine-tuning thresholds to achieve optimal results.

```
!pip install torch torchvision opencv-python matplotlib
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.5.1+cu121)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.20.1+cu121)
Requirement already satisfied: opencv-python in /usr/local/lib/python3.10/dist-packages (4.10.0.84)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.16.1)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/python3.10/dist-packages (from torch) (4.12.2)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.4.2)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2024.10.0)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.10/dist-packages (from torch) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from sympy==1.13.1->torch) (1.3.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.26.4)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (11.0.0)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (3.0.2)
```

```
import torch
from torchvision import models, transforms
import cv2
from matplotlib import pyplot as plt
```

## Pre-trained Object Detection Model

```
model = models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()  # Set the model to evaluation mode
```

```
            (bn3): FrozenBatchNorm2d(2048, eps=0.0)
            (relu): ReLU(inplace=True)
          )
        )
      )
      (fpn): FeaturePyramidNetwork(
        (inner_blocks): ModuleList(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 256, kernel_size=(1, 1), stride=(1, 1))
          )
          (1): Conv2dNormActivation(
            (0): Conv2d(512, 256, kernel_size=(1, 1), stride=(1, 1))
          )
          (2): Conv2dNormActivation(
            (0): Conv2d(1024, 256, kernel_size=(1, 1), stride=(1, 1))
          )
          (3): Conv2dNormActivation(
            (0): Conv2d(2048, 256, kernel_size=(1, 1), stride=(1, 1))
          )
        )
        (layer_blocks): ModuleList(
          (0-3): 4 x Conv2dNormActivation(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
          )
        )
        (extra_blocks): LastLevelMaxPool()
      )
    )
    (rpn): RegionProposalNetwork(
      (anchor_generator): AnchorGenerator()
      (head): RPNHead(
        (conv): Sequential(
          (0): Conv2dNormActivation(
            (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
            (1): ReLU(inplace=True)
          )
        )
        (cls_logits): Conv2d(256, 3, kernel_size=(1, 1), stride=(1, 1))
        (bbox_pred): Conv2d(256, 12, kernel_size=(1, 1), stride=(1, 1))
      )
    )
    (roi_heads): RoIHeads(
      (box_roi_pool): MultiScaleRoIAlign(featmap_names=['0', '1', '2', '3'], output_size=(7, 7), sampling_ratio=2)
      (box_head): TwoMLPHead(
        (fc6): Linear(in_features=12544, out_features=1024, bias=True)
        (fc7): Linear(in_features=1024, out_features=1024, bias=True)
      )
      (box_predictor): FastRCNNPredictor(
        (cls_score): Linear(in_features=1024, out_features=91, bias=True)
        (bbox_pred): Linear(in_features=1024, out_features=364, bias=True)
      )
    )
  )
```

```
# Dowloading images
!wget https://github.com/pytorch/hub/raw/master/images/dog.jpg -O test_image.jpg
```

```
--2024-11-25 19:41:53--  https://github.com/pytorch/hub/raw/master/images/dog.jpg
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/pytorch/hub/master/images/dog.jpg [following]
--2024-11-25 19:41:53--  https://raw.githubusercontent.com/pytorch/hub/master/images/dog.jpg
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.111.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 661378 (646K) [image/jpeg]
Saving to: ‘test_image.jpg’

test_image.jpg      100%[===================>] 645.88K  --.-KB/s    in 0.06s

2024-11-25 19:41:54 (11.1 MB/s) - ‘test_image.jpg’ saved [661378/661378]
```

```
# Load the image for testing:
from PIL import Image
image = Image.open("test_image.jpg")
```

```
#  Preprocess the Image
preprocess = transforms.Compose([
```

```
    transforms.ToTensor()
])
input_tensor = preprocess(image).unsqueeze(0)


# Run Object Detection
with torch.no_grad():
    predictions = model(input_tensor)


# Visualize predictions
def draw_boxes(image, predictions, threshold=0.5):
    img = cv2.cvtColor(cv2.imread("test_image.jpg"), cv2.COLOR_BGR2RGB)
    for i, box in enumerate(predictions[0]['boxes']):
        if predictions[0]['scores'][i] > threshold:
            x1, y1, x2, y2 = box
            label = predictions[0]['labels'][i].item()
            score = predictions[0]['scores'][i].item()
            img = cv2.rectangle(img, (int(x1), int(y1)), (int(x2), int(y2)), (255, 0, 0), 2)
            img = cv2.putText(img, f"{label} {score:.2f}", (int(x1), int(y1)-10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255, 0, 0), 2)
    plt.imshow(img)
    plt.axis('off')
    plt.show()

draw_boxes(image, predictions)
```

**Model Predictions:**

The model successfully detected objects in the input image, as evidenced by the bounding boxes. Confidence scores of the predictions were relatively low (<0.5), indicating uncertainty in the detections.

## Evaluation of the Model

```
# IoU Calculation Function
def iou(box1, box2):
    """Calculate Intersection over Union (IoU) for two bounding boxes."""
    x1 = max(box1[0], box2[0])
    y1 = max(box1[1], box2[1])
    x2 = min(box1[2], box2[2])
    y2 = min(box1[3], box2[3])

    # Calculate the area of intersection
    intersection = max(0, x2 - x1) * max(0, y2 - y1)

    # Calculate the area of both rectangles
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])

    # Union area
    union = box1_area + box2_area - intersection

    # IoU
```

```
    return intersection / union if union != 0 else 0
```

***IoU Implementation:***

The Intersection over Union (IoU) function is implemented to calculate the overlap between predicted bounding boxes and ground truth. The IoU value helps measure how well the predictions align with the ground truth.

```
draw_boxes(image, predictions, threshold=0.2)
```

```
for i, box in enumerate(predictions[0]['boxes']):
    for gt_box in ground_truth:
        print(f"IoU between predicted box {box} and ground truth {gt_box}: {iou(box, gt_box)}")
```

```
IoU between predicted box tensor([ 137.8405,   67.7994, 1386.9043, 1172.8247]) and ground truth [50, 50, 200, 200]: 0.005892683751881123
IoU between predicted box tensor([ 134.8363,   45.0133, 1471.7245, 1203.3080]) and ground truth [50, 50, 200, 200]: 0.006260779686272144
IoU between predicted box tensor([253.0188, 306.1613, 406.6772, 365.1062]) and ground truth [50, 50, 200, 200]: 0.0
IoU between predicted box tensor([130.7382,  62.5920, 993.0493, 874.3549]) and ground truth [50, 50, 200, 200]: 0.013348469510674477
```

The output indicates the Intersection over Union (IoU) values between the predicted bounding boxes and the ground truth bounding box:

**IoU Values:**

The IoU values are very low (e.g., 0.005, 0.006, 0.0, 0.013). This means the predicted bounding boxes have little to no overlap with the ground truth bounding box.

**Interpretation:**

An IoU close to 0 indicates that the predicted box and ground truth box do not overlap. An IoU of 1 would indicate a perfect match between the prediction and the ground truth.

**Possible Issues:**

The model's predictions may not align well with the target object in the image. The ground truth bounding box might not be accurately defined for this target. The pre-trained weights of the model might not generalize well to this specific dataset or image.

**Actionable Next Steps:**

Ensure the ground truth bounding box is accurately defined to represent the target object. Fine-tune the model on a dataset closer to your target domain. Adjust IoU thresholds for evaluation to filter out irrelevant predictions.

```
import cv2
img = cv2.cvtColor(cv2.imread("test_image.jpg"), cv2.COLOR_BGR2RGB)
for box in ground_truth:
    x1, y1, x2, y2 = box
    img = cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Green for ground truth
plt.imshow(img)
plt.axis('off')
plt.show()
```

```
# Calculate Precision and Recall
def evaluate_predictions(predictions, ground_truth, threshold=0.5, iou_threshold=0.5):
    true_positive = 0
    false_positive = 0
    false_negative = len(ground_truth)  # Start with all ground truth as false negatives

    for i, box in enumerate(predictions[0]['boxes']):
        score = predictions[0]['scores'][i]
        if score > threshold:
            matched = False
            for gt_box in ground_truth:
                if iou(box, gt_box) >= iou_threshold:
                    true_positive += 1
                    false_negative -= 1
                    matched = True
                    break
            if not matched:
                false_positive += 1

    precision = true_positive / (true_positive + false_positive) if (true_positive + false_positive) > 0 else 0
    recall = true_positive / (true_positive + false_negative) if (true_positive + false_negative) > 0 else 0
    f1_score = (2 * precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

    return precision, recall, f1_score


print(predictions[0]['boxes'])  # List of predicted bounding boxes
print(predictions[0]['scores'])  # Confidence scores for each box
```

```
tensor([[ 137.8405,   67.7994, 1386.9043, 1172.8247],
        [ 134.8363,   45.0133, 1471.7245, 1203.3080],
        [ 253.0188,  306.1613,  406.6772,  365.1062],
        [ 130.7382,   62.5920,  993.0493,  874.3549]])
tensor([0.9669, 0.3522, 0.3132, 0.1133])
```

The model's predictions result in low IoU values with the ground truth, indicating poor alignment between predicted and actual bounding boxes. Precision, recall, and F1-score calculations highlight significant room for improvement in the model's performance, potentially due to incorrect ground truth annotations, suboptimal thresholds, or mismatched pre-trained weights.

## Debugging

```
ground_truth = [[100, 40, 1450, 1100]]


draw_boxes(image, predictions, threshold=0.2)  # Ensure predictions are visible
```

```
#  Visualize
img = cv2.imread("test_image.jpg")
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
for box in ground_truth:
    x1, y1, x2, y2 = box
    cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)  # Green for ground truth
plt.imshow(img)
plt.axis("off")
plt.show()
```

**Observations** The green ground truth box now closely matches the dog's position. Predicted boxes are not aligning well with the object. IoU values are very low, even with the lower IoU threshold (0.2). Predictions seem to target other irrelevant areas.

```
# Check Predictions vs. Ground Truth
for i, box in enumerate(predictions[0]['boxes']):
    for gt_box in ground_truth:
        iou_value = iou(box.tolist(), gt_box)
        print(f"Prediction {i}: IoU with ground truth = {iou_value:.2f}")
```

```
Prediction 0: IoU with ground truth = 0.85
Prediction 1: IoU with ground truth = 0.87
Prediction 2: IoU with ground truth = 0.01
Prediction 3: IoU with ground truth = 0.49
```

```
# Fix the Messy Ground Truth
# [x1, y1] should represent the top-left corner of the object.
# [x2, y2] should represent the bottom-right corner.

img = cv2.imread("test_image.jpg")
height, width, _ = img.shape
print(f"Image dimensions: Width={width}, Height={height}")
```

```
Image dimensions: Width=1546, Height=1213
```

```
# Evaluate the Model
precision, recall, f1_score = evaluate_predictions(predictions, ground_truth, threshold=0.2, iou_threshold=0.2)
print(f"Precision: {precision}, Recall: {recall}, F1 Score: {f1_score}")
```

```
Precision: 0.6666666666666666, Recall: 2.0, F1 Score: 1.0
```

The IoU values indicate improved alignment for two predictions (0.85 and 0.87). Precision, recall, and F1-score suggest acceptable performance with further optimization needed for thresholds and ground truth annotations.

```
print(predictions[0]['boxes'])  # Predicted bounding boxes
print(predictions[0]['scores'])  # Confidence scores
```

```
tensor([[ 137.8405,   67.7994, 1386.9043, 1172.8247],
        [ 134.8363,   45.0133, 1471.7245, 1203.3080],
        [ 253.0188,  306.1613,  406.6772,  365.1062],
        [ 130.7382,   62.5920,  993.0493,  874.3549]])
tensor([0.9669, 0.3522, 0.3132, 0.1133])
```

If scores are low (<0.5), the model may not be confident in its predictions. High scores with irrelevant boxes indicate that the model's pre-trained weights are not ideal for this dataset.

```
# Focus on High-Confidence Predictions
high_confidence_boxes = []
threshold = 0.5  # Adjust this threshold
for i, score in enumerate(predictions[0]['scores']):
    if score > threshold:
        high_confidence_boxes.append(predictions[0]['boxes'][i].tolist())

print(f"High-confidence predictions: {high_confidence_boxes}")
```

```
High-confidence predictions: [[137.84051513671875, 67.79939270019531, 1386.904296875, 1172.82470703125]]
```

The predictions with high confidence (scores above the threshold of 0.5) are being effectively filtered and displayed. One high-confidence bounding box is detected, indicating that the model recognizes a region of interest with reasonable certainty.

```
for box in high_confidence_boxes:
    x1, y1, x2, y2 = map(int, box)
    cv2.rectangle(img, (x1, y1), (x2, y2), (255, 0, 0), 2)  # Blue for predictions
plt.imshow(img)
plt.axis("off")
plt.show()
```

```
#  Debug IoU Calculation
for i, box in enumerate(predictions[0]['boxes']):
    for gt_box in ground_truth:
        iou_value = iou(box.tolist(), gt_box)
        print(f"Prediction {i}: IoU with ground truth = {iou_value:.2f}")
```

```
Prediction 0: IoU with ground truth = 0.85
Prediction 1: IoU with ground truth = 0.87
Prediction 2: IoU with ground truth = 0.01
Prediction 3: IoU with ground truth = 0.49
```