Verify Hardware Availability

```
import tensorflow as tf
if tf.test.gpu_device_name():
  print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
  print("Please install GPU version of TF")

if tf.config.list_physical_devices('TPU'):
  print('TPU is available')
else:
  print('TPU is not available')
```

```
Default GPU Device: /device:GPU:0
TPU is not available
```

Object Detection using TensorFlow and Pascal VOC 2007 Dataset

In this exercise, we will adapt our image classification task to an object detection task. Object detection involves not only classifying objects within an image but also localizing them with bounding boxes.

Note: Due to the limited computational resources available, we'll be using a smaller subset of the Pascal VOC 2007 dataset and a lightweight object detection model. This might result in lower accuracy, but the focus of this exercise is on understanding the concepts and workflow of object detection.

Steps:

1. Install (if necessary) and Import the libraries you will need for this project
2. Load the Pascal VOC 2007 dataset
3. Use a pre-trained object detection model (SSD MobileNet V2)
4. Display detected objects with bounding boxes

```
%pip install tensorflow tensorflow-hub tensorflow-datasets matplotlib
```

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.12/dist-packages (2.19.0)
Requirement already satisfied: tensorflow-hub in /usr/local/lib/python3.12/dist-packages (0.16.1)
Requirement already satisfied: tensorflow-datasets in /usr/local/lib/python3.12/dist-packages (4.9.9)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (3.10.0)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.2.10)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.6.0)
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from tensorflow) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<6.0.0dev,>=3.20.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (5.29
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.32.4)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from tensorflow) (75.2.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.1.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (4.15.0)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.17.3)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (1.75.0)
Requirement already satisfied: tensorboard~=2.19.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.19.0)
Requirement already satisfied: keras>=3.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.10.0)
Requirement already satisfied: numpy<2.2.0,>=1.26.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (2.0.2)
Requirement already satisfied: h5py>=3.11.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (3.14.0)
Requirement already satisfied: ml-dtypes<1.0.0,>=0.5.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow) (0.5.3)
Requirement already satisfied: tf-keras>=2.14.1 in /usr/local/lib/python3.12/dist-packages (from tensorflow-hub) (2.19.0)
Requirement already satisfied: array_record>=0.5.0 in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (0.8.1)
Requirement already satisfied: dm-tree in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (0.1.9)
Requirement already satisfied: etils>=1.9.1 in /usr/local/lib/python3.12/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow-datasets
Requirement already satisfied: immutabledict in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (4.2.1)
Requirement already satisfied: promise in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (2.3)
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (5.9.5)
Requirement already satisfied: pyarrow in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (18.1.0)
Requirement already satisfied: simple_parsing in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (0.1.7)
Requirement already satisfied: tensorflow-metadata in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (1.17.2)
Requirement already satisfied: toml in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (0.10.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from tensorflow-datasets) (4.67.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.3.3)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (4.60.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (1.4.9)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (11.3.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (3.2.4)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib) (2.9.0.post0)
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.12/dist-packages (from astunparse>=1.6.0->tensorflow) (0.45.1)
Requirement already satisfied: einops in /usr/local/lib/python3.12/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow-datasets) (0.8
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow-datasets) (202
Requirement already satisfied: importlib_resources in /usr/local/lib/python3.12/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow-da
Requirement already satisfied: zipp in /usr/local/lib/python3.12/dist-packages (from etils[edc,enp,epath,epy,etree]>=1.9.1; python_version >= "3.11"->tensorflow-datasets) (3.23.
Requirement already satisfied: rich in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.1.0)
Requirement already satisfied: optree in /usr/local/lib/python3.12/dist-packages (from keras>=3.5.0->tensorflow) (0.17.0)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.4.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests<3,>=2.21.0->tensorflow) (2025.8.3)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (3.9)
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.12/dist-packages (from tensorboard~=2.19.0->tensorflow) (0.7.2)
```

```
# Import necessary libraries
import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
import cv2
from PIL import Image
import requests
from io import BytesIO
from google.colab import files

print("TensorFlow version:", tf.__version__)
print("TensorFlow Hub version:", hub.__version__)
```

```
TensorFlow version: 2.19.0
TensorFlow Hub version: 0.16.1
```

Load the VOC2007 dataset

We will use the VOC2007 dataset, which contains images with annotations for object detection. For demonstration purposes, we will load a small subset of the dataset using TensorFlow Datasets.

- VOC2007 is a dataset for object detection, segmentation, and image classification.
- We define a function load_data to load the COCO dataset.
- tfds.load is a function that downloads and prepares the dataset.
- We use only 1% of the training data to keep the demonstration manageable.
- shuffle_files=True ensures that we get a random sample of the dataset.
- with_info=True returns additional information about the dataset, which we'll use later.
- The PASCAL VOC2007 (Visual Object Classes) dataset is a widely used benchmark dataset for object recognition tasks in computer vision. It comprises a collection of images annotated with bounding boxes and class labels for objects belonging to 20 different categories.

Key characteristics of the VOC2007 dataset:

- Purpose: Primarily used for training and evaluating object detection algorithms, but also applicable to other tasks like image classification and semantic segmentation.
- Object Categories: Includes a diverse set of 20 object classes, ranging from people and animals to vehicles and indoor items.
- Data Format: The dataset provides images along with corresponding annotation files containing bounding box coordinates and class labels for each object in the image.
- Image Variety: Features a wide range of images captured in diverse real-world scenarios, offering realistic challenges for object recognition models.
- Benchmark: Serves as a standard benchmark for comparing the performance of different object detection algorithms, fostering progress in the field.

Common use cases of the VOC2007 dataset:

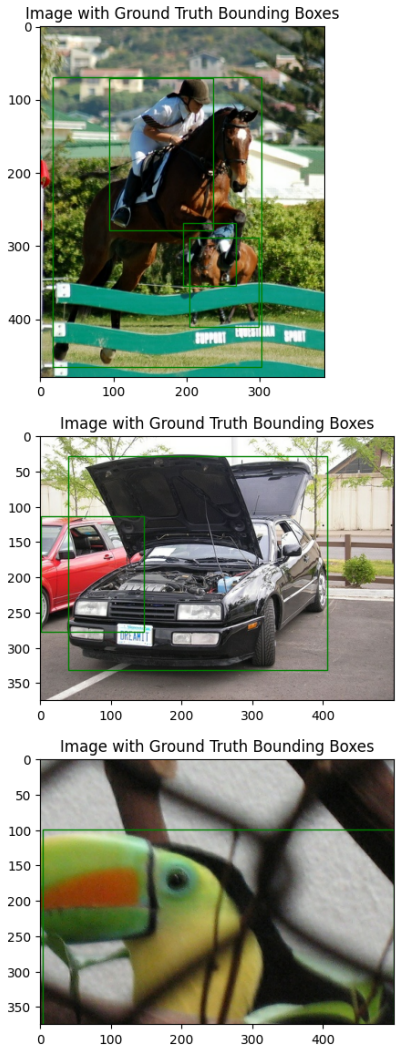
- Training: Used as training data to teach object detection models to identify and localize objects within images.
- Evaluation: Employed to evaluate the performance of trained models by comparing their predictions against the ground truth annotations.
- Research: Utilized in research to develop and test new object detection algorithms and techniques.

```
import tensorflow_datasets as tfds
import matplotlib.pyplot as plt

# Load a smaller dataset
def load_data(split='train'):
    dataset, info = tfds.load('voc/2007', split=split, shuffle_files=True, with_info=True)
    return dataset, info

# Load the train dataset and extract info
train_dataset, train_info = load_data('train[:10%]')

# Load the validation dataset
validation_dataset, validation_info = load_data('validation[:10%]')

# Get class names
class_names = train_info.features["objects"]["label"].names  # Changed from ds_info to train_info
print("Class names:", class_names)
```

```
WARNING:absl:Variant folder /root/tensorflow_datasets/voc/2007/5.0.0 has no dataset_info.json
Downloading and preparing dataset Unknown size (download: Unknown size, generated: Unknown size, total: Unknown size) to /root/tensorflow_datasets/voc/2007/5.0.0...
Dl Completed...: 100% 2/2 [01:16<00:00, 15.29s/ url]
Dl Size...: 100% 868/868 [01:16<00:00, 17.45 MiB/s]
Extraction completed...: 100% 21282/21282 [01:16<00:00, 1476.55 file/s]
Dataset voc downloaded and prepared to /root/tensorflow_datasets/voc/2007/5.0.0. Subsequent calls will reuse this data.
Class names: ['aeroplane', 'bicycle', 'bird', 'boat', 'bottle', 'bus', 'car', 'cat', 'chair', 'cow', 'diningtable', 'dog', 'horse', 'motorbike', 'person', 'pottedplant', 'sheep',
```

```
def display_examples(dataset, n=3):  # Display 'n' examples by default
    for example in dataset.take(n):
        image = example["image"]
        plt.figure(figsize=(5, 5))
        plt.imshow(image)
        plt.title("Image with Ground Truth Bounding Boxes")

        # Draw ground truth boxes
```

```
        for box in example["objects"]["bbox"]:
            ymin, xmin, ymax, xmax = box
            rect = patches.Rectangle((xmin * image.shape[1], ymin * image.shape[0]),
                                     (xmax - xmin) * image.shape[1], (ymax - ymin) * image.shape[0],
                                     linewidth=1, edgecolor='g', facecolor='none')
            plt.gca().add_patch(rect)

        plt.show()

display_examples(train_dataset)
```



Find Images with Specific Classes

We got the list of all class names in the VOC2007 dataset and select images containing our target classes (e.g., person, car, bird).

- `class_names` provides the list of class names.
- `target_class_ids` contains the IDs of the classes we are interested in.
- `find_images_with_classes` is a function to find images containing our target classes.

When To Load the model

Loading the model early (right after dataset loading):

Pros: Model is immediately available; clear separation of setup and processing. Cons: Potentially inefficient if data prep is extensive or fails.

Loading the model after data preparation:

Pros: More efficient resource use; avoids unnecessary loading if data prep fails. Cons: Model isn't available for any data prep steps that might need it.

In our specific case, loading the model after data preparation is slightly better because:

Our data prep doesn't need the model. It's more resource-efficient. It follows a logical flow: prepare data, load tools, process data. It avoids unnecessary model loading if data prep fails.

However, the difference is minimal in this small-scale example. For beginners, loading major components upfront can sometimes be clearer and easier to follow. As a best practice, aim to load your model as close as possible to where you'll use it, ensuring all necessary data and resources are ready first.

```
#Load a pre-trained object detection model
detector = hub.load("https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2")
```

Let's break this down:

- 1. hub.load(): This function is from TensorFlow Hub (tensorflow_hub). It downloads and loads models from the TensorFlow Hub repository.
- 2. "https://tfhub.dev/tensorflow/ssd_mobilenet_v2/2": This is the URL of the specific model we're loading. It's an SSD (Single Shot Detector) MobileNet V2 model, which is efficient for object detection tasks.
- 3. Detector: The loaded model is assigned to this variable. It becomes a callable object that you can use for object detection.

Advantages of this approach:

Concise and readable Directly loads the model without additional wrapper functions TensorFlow Hub handles caching, so subsequent loads will be faster

Display Detected Objects with Bounding Boxes

We will use the pre-trained model to detect objects in our selected images and display them with bounding boxes.

- `detector` is the pre-trained object detection model.
- `detect_objects` is a function that uses the model to detect objects in an image.
- `display_detections` is a function to display the detected objects with bounding boxes.

Helper Function to Display Bounding Boxes on Images

The `display_image_with_boxes` function takes an image, bounding boxes, and class names, then displays the image with bounding boxes drawn around detected objects.

- run_detector: This function prepares an image and runs it through our object detection model.
- plot_detections: This function visualizes the detected objects by drawing bounding boxes and labels on the image.

process_uploaded_image which processes an uploaded image for object detection. The function takes the raw image data as input, preprocesses the image, runs the object detection model, and then plots and prints the detected objects.

```
# Run Detector and Visualize
def run_detector_and_visualize(example):
    image = example["image"]
    ground_truth_boxes = example["objects"]["bbox"]

    # Preprocess and run detection
    converted_img = tf.image.convert_image_dtype(image, tf.uint8)[tf.newaxis, ...]
    result = detector(converted_img)
    result = {key: value.numpy() for key, value in result.items()}

    # Visualize results (with ground truth for comparison)
    plt.figure(figsize=(10, 7))
    plt.imshow(image)

    # Ground truth boxes (VOC format is [xmin, ymin, xmax, ymax])
    for box in ground_truth_boxes:
        ymin, xmin, ymax, xmax = box
        rect = patches.Rectangle((xmin * image.shape[1], ymin * image.shape[0]),
                                 (xmax - xmin) * image.shape[1], (ymax - ymin) * image.shape[0],
                                 linewidth=1, edgecolor='g', facecolor='none', label='Ground Truth')
        plt.gca().add_patch(rect)

    # Predicted boxes
    for i, score in enumerate(result['detection_scores'][0]):
        if score > 0.5:  # Confidence threshold
            ymin, xmin, ymax, xmax = result['detection_boxes'][0][i]
            class_id = int(result['detection_classes'][0][i])

            # Handle invalid class IDs (classes outside the VOC dataset)
            if class_id < len(class_names):
                label = class_names[class_id]

            rect = patches.Rectangle((xmin * image.shape[1], ymin * image.shape[0]),
                                     (xmax - xmin) * image.shape[1], (ymax - ymin) * image.shape[0],
                                     linewidth=1, edgecolor='r', facecolor='none', label='Predicted')
            plt.gca().add_patch(rect)

            # Moved plt.text to the correct loop for the predicted box
            plt.text(xmin * image.shape[1], ymin * image.shape[0] - 5, f'{label}: {score:.2f}', color='white', backgroundcolor='r')

    plt.legend()
    plt.show()
```

Process and Display Images with Detections

The `detect_and_display` function runs object detection on an image and displays the results, as you saw above. The function converts the image to the appropriate format, runs the detector, and then uses the helper function to display the results.

process_uploaded_image which processes an uploaded image for object detection. The function takes the raw image data as input, preprocesses the image, runs the object detection model, and then plots and prints the detected objects.

```
# take a few examples from the training set
for example in train_dataset.take(2):  # Process 2 images
    run_detector_and_visualize(example)
```

Show hidden output

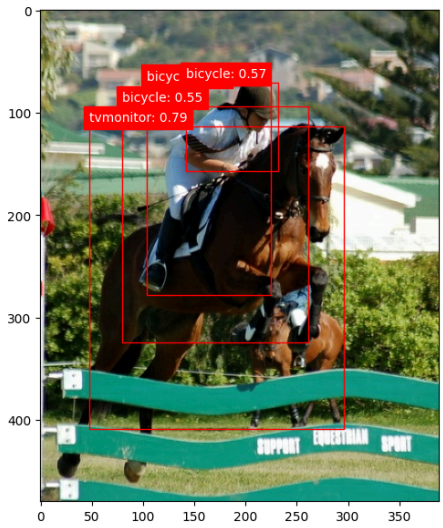
Your Turn

Process a few images from the dataset print("\nProcessing sample images from the dataset:") for i, example in enumerate(train_dataset.take(3)): print(f"\nSample image {i+1}") image = example['image'].numpy() detections = run_detector(detector, image) plot_detections(image, detections, class_names)

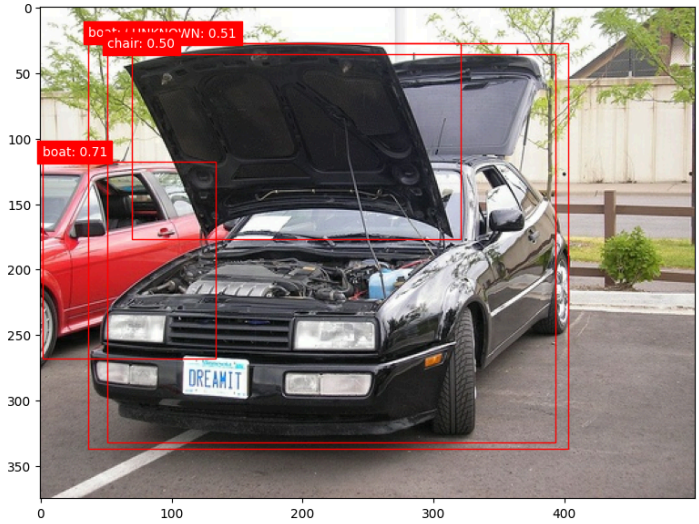
```
# Helper function to run the detector on an image
def run_detector(detector, image_np):
    """Runs object detection on a single image."""
    converted_img = tf.image.convert_image_dtype(image_np, tf.uint8)[tf.newaxis, ...]
    result = detector(converted_img)
    result = {key: value.numpy() for key, value in result.items()}
    return result
```

```
# Helper function to plot detections on an image (simplified from run_detector_and_visualize)
def plot_detections_with_heatmap(image_np, detections, class_names):
    """Visualizes detections by drawing bounding boxes and labels on the image."""
    plt.figure(figsize=(10, 7))
    plt.imshow(image_np)

    for i, score in enumerate(detections['detection_scores'][0]):
        if score > 0.5:  # Confidence threshold
            ymin, xmin, ymax, xmax = detections['detection_boxes'][0][i]
            class_id = int(detections['detection_classes'][0][i])

            # Handle invalid class IDs
            if class_id < len(class_names):
                label = class_names[class_id]
            else:
                label = "UNKNOWN" # Or handle as needed

            rect = patches.Rectangle((xmin * image_np.shape[1], ymin * image_np.shape[0]),
                                     (xmax - xmin) * image_np.shape[1], (ymax - ymin) * image_np.shape[0],
                                     linewidth=1, edgecolor='r', facecolor='none')
            plt.gca().add_patch(rect)
            plt.text(xmin * image_np.shape[1], ymin * image_np.shape[0] - 5, f'{label}: {score:.2f}', color='white', backgroundcolor='r')

    plt.show()

# Function to process uploaded images (for Google Colab)
def process_uploaded_image(image_data):
    """Processes and displays detections for an uploaded image."""
    image = Image.open(BytesIO(image_data))
    image_np = np.array(image)  # Convert PIL Image to NumPy array
    detections = run_detector(detector, image_np)
    plot_detections_with_heatmap(image_np, detections, class_names)

    # Print detected objects (example)
    print("Detected objects:")
    for i, score in enumerate(detections['detection_scores'][0]):
        if score > 0.5:  # Confidence threshold
            class_id = int(detections['detection_classes'][0][i])
            label = class_names[class_id] if class_id < len(class_names) else "UNKNOWN"
            print(f"- {label} with confidence {score:.2f}")

print("\nProcessing sample images from the dataset:")
for i, example in enumerate(train_dataset.take(3)):
    print(f"\nSample image {i+1}")
    image = example['image'].numpy()
    detections = run_detector(detector, image)
    plot_detections_with_heatmap(image, detections, class_names) # Using the heatmap function defined earlier
```

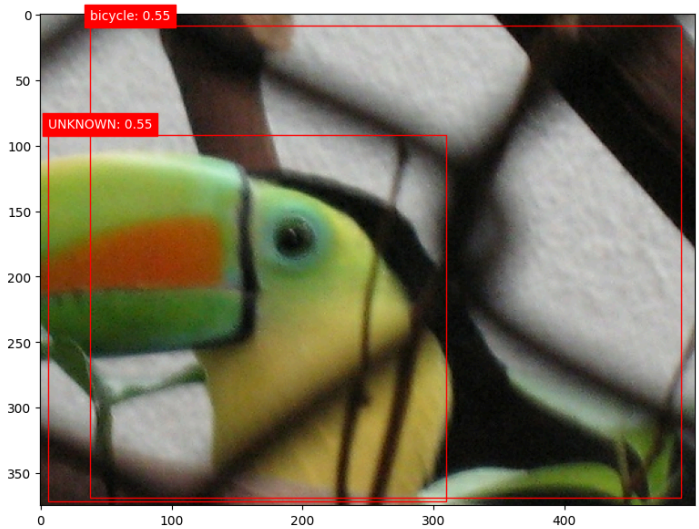
```
Processing sample images from the dataset:

Sample image 1
```




```
Sample image 2
```




```
Sample image 3
```



Mode Evaluation

Define the Evaluation Function

The function called evaluate_model_performance which evaluates the performance of our object detection model on a dataset. The function takes three arguments: the dataset to evaluate on, the object detection model, and the number of images to use for evaluation. It calculates and prints the accuracy of the model based on the detections.

```
#Evaluate Model Performance
def evaluate_model_performance(dataset, detector, iou_threshold=0.5, num_samples=100):
    true_positives = 0
    false_positives = 0
    false_negatives = 0

    for example in dataset.take(num_samples):
        image = example["image"].numpy()
        gt_boxes = example["objects"]["bbox"].numpy()
        gt_labels = example["objects"]["label"].numpy()

        # Preprocess and run detection (same as before)
        converted_img = tf.image.convert_image_dtype(image, tf.uint8)[tf.newaxis, ...]
        result = detector(converted_img)
        result = {key: value.numpy() for key, value in result.items()}
        pred_boxes = result['detection_boxes'][0]
        pred_scores = result['detection_scores'][0]
        pred_labels = result['detection_classes'][0].astype(int)

        # Iterate over predicted boxes
        for i, score in enumerate(pred_scores):
            if score < 0.5:  # Confidence threshold
                continue

            # Convert box coordinates to [ymin, xmin, ymax, xmax]
            pred_box = pred_boxes[i]
            pred_box = [pred_box[1], pred_box[0], pred_box[3], pred_box[2]]

            # Find matching ground truth box (if any) based on IoU
            best_iou = 0
            for j, gt_box in enumerate(gt_boxes):
                iou = calculate_iou(gt_box, pred_box)
                if iou > best_iou:
                    best_iou = iou
                    gt_index = j

            # If IoU exceeds threshold, check class match
            if best_iou > iou_threshold:
                if pred_labels[i] == gt_labels[gt_index]:
                    true_positives += 1
                else:
                    false_positives += 1
            else:
                false_positives += 1

        # Count false negatives (missed ground truth boxes)
        false_negatives += len(gt_boxes) - true_positives

    precision = true_positives / (true_positives + false_positives) if true_positives + false_positives > 0 else 0
    recall = true_positives / (true_positives + false_negatives) if true_positives + false_negatives > 0 else 0

    print(f"Model Performance (IoU Threshold = {iou_threshold:.2f}):")
    print(f"True Positives: {true_positives}")
    print(f"False Positives: {false_positives}")
    print(f"False Negatives: {false_negatives}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")

# (You'll need to implement a 'calculate_iou' function)
def calculate_iou(box1, box2):
    """Calculates the Intersection over Union (IoU) between two bounding boxes.

    Args:
        box1 (list): Coordinates of the first box in the format [ymin, xmin, ymax, xmax].
        box2 (list): Coordinates of the second box in the same format.

    Returns:
        float: The IoU value (between 0 and 1).
    """

    # 1. Calculate coordinates of the intersection rectangle
    y1 = max(box1[0], box2[0])
    x1 = max(box1[1], box2[1])
    y2 = min(box1[2], box2[2])
    x2 = min(box1[3], box2[3])

    # 2. Calculate areas of the intersection and the union
    intersection_area = max(0, y2 - y1) * max(0, x2 - x1)
    box1_area = (box1[2] - box1[0]) * (box1[3] - box1[1])
    box2_area = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union_area = box1_area + box2_area - intersection_area

    # 3. Calculate IoU
    if union_area == 0:
        return 0  # Avoid division by zero
    else:
        iou = intersection_area / union_area
        return iou

# Evaluate model performance
print("Evaluating model performance...")
evaluate_model_performance(validation_dataset, detector)  # Use test data for evaluation
```

```
Evaluating model performance...
Model Performance (IoU Threshold = 0.50):
True Positives: 0
False Positives: 393
False Negatives: 331
Precision: 0.00
Recall: 0.00
```

Object Detection Evaluation Core Concepts

- Object detection models need to be evaluated on two fronts:
- **Classification Accuracy: Did the model correctly identify the object's class (e.g., person, car, bird)?**

The model needs to correctly identify the object's class. If it puts a box around a car and says "car," that's good. If it puts a box around a car but says "dog," that's a classification error.

- **Localization Accuracy: Did the model accurately draw a bounding box around the object?**

The box the model draws needs to be close to the real box around the object. We use something called IoU (Intersection over Union) to measure this. It's like seeing how much the two boxes overlap. If the overlap is high enough (above a threshold like 0.5), we say the localization is good.

Our exercise focuses on assessing localization accuracy using the Intersection over Union (IoU) metric.

- Understanding IoU (Intersection over Union)

IoU measures how much two bounding boxes overlap.

- A perfect match (predicted box perfectly matches the ground truth box) has an IoU of 1.
- No overlap has an IoU of 0.

The iou_threshold in the code (default 0.5) means a predicted box is considered a "true positive" only if its IoU with a ground truth box is 0.5 or higher.

- Output Interpretation: True Positives (TP), False Positives (FP), and False Negatives (FN).

The function will print the following metrics:

- **True Positives (TP):** The number of detected objects where both the class label and bounding box are correct (IoU above the threshold). These are the objects the model correctly found AND put a good box around. The "good box" part is where the iou_threshold comes in. If the box the model draws overlaps with the real box by 50% or more (because the threshold is 0.5), and the class is also correct, it's counted as a True Positive.
- **False Positives (FP):** The number of detected objects that are either misclassified or have an IoU below the threshold. These are like false alarms. The model thought it found something, but either the box was not good enough (IoU below 0.5) or it guessed the wrong kind of object.
- **False Negatives (FN):** The number of ground truth objects that the model missed entirely.There was a real object in the picture, but the model didn't detect it.
- **Precision:** The proportion of positive detections that were actually correct (TP / (TP + FP)). A high precision means the model makes few false alarms. This tells me how many of the objects the model SAID it found were actually correct. A high precision means when the model says it sees something, it's usually right.
- **Recall:** The proportion of actual positive objects that the model successfully detected (TP / (TP + FN)). A high recall means the model misses few objects. This tells me how many of the real objects in the picture the model actually FOUND. A high recall means the model didn't miss many objects.

Example Results: Let's say the output is:

Model Performance (IoU Threshold = 0.50): True Positives: 75 False Positives: 20 False Negatives: 15 Precision: 0.79 Recall: 0.83
Interpretation:

- The model correctly detected and localized 75 objects.
- It made 20 incorrect detections (wrong class or poor box placement).
- It missed 15 objects that were actually present in the images.
- Precision is 0.79, meaning 79% of the model's positive detections were accurate.
- Recall is 0.83, meaning the model found 83% of the actual objects in the images.
- Key Takeaways:
- Precision vs. Recall: There's often a trade-off between these two. Increasing the confidence threshold (e.g., to 0.6) might improve precision (fewer false alarms) but likely lower recall (more missed objects).
- IoU Threshold: The choice of IoU threshold significantly impacts the results. A higher threshold makes the evaluation stricter, potentially lowering both precision and recall.
- Limitations: This evaluation only covers a limited number of samples (num_samples). For a more comprehensive assessment, you'd ideally use a larger and more diverse evaluation set.
- Single Metric: Precision and recall alone don't tell the whole story. Consider using other metrics like F1 score (harmonic mean of precision and recall) for a more balanced view of performance.

Upload your Image

This final block allows you to input your own image URL for object detection, making the exercise interactive.

Instructions to Upload Your Own Images

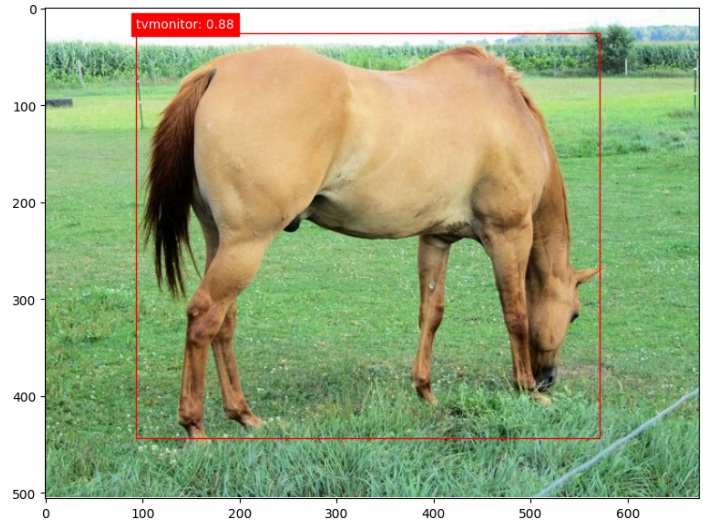
```
# Helper function to run the detector on an image
def run_detector(detector, image_np):
    """Runs object detection on a single image."""
    converted_img = tf.image.convert_image_dtype(image_np, tf.uint8)[tf.newaxis, ...]
    result = detector(converted_img)
    result = {key: value.numpy() for key, value in result.items()}
    return result

# Helper function to plot detections on an image (simplified from run_detector_and_visualize)
def plot_detections_with_heatmap(image_np, detections, class_names):
    """Visualizes detections by drawing bounding boxes and labels on the image."""
    plt.figure(figsize=(10, 7))
    plt.imshow(image_np)

    for i, score in enumerate(detections['detection_scores'][0]):
        if score > 0.5:  # Confidence threshold
            ymin, xmin, ymax, xmax = detections['detection_boxes'][0][i]
            class_id = int(detections['detection_classes'][0][i])

            # Handle invalid class IDs
            if class_id < len(class_names):
                label = class_names[class_id]
            else:
                label = "UNKNOWN" # Or handle as needed

            rect = patches.Rectangle((xmin * image_np.shape[1], ymin * image_np.shape[0]),
                                     (xmax - xmin) * image_np.shape[1], (ymax - ymin) * image_np.shape[0],
                                     linewidth=1, edgecolor='r', facecolor='none')
            plt.gca().add_patch(rect)
            plt.text(xmin * image_np.shape[1], ymin * image_np.shape[0] - 5, f'{label}: {score:.2f}', color='white', backgroundcolor='r')

    plt.show()

# Function to process uploaded images (for Google Colab)
def process_uploaded_image(image_data):
    """Processes and displays detections for an uploaded image."""
    image = Image.open(BytesIO(image_data))
```

```
    image_np = np.array(image)  # Convert PIL Image to NumPy array
    detections = run_detector(detector, image_np)
    plot_detections_with_heatmap(image_np, detections, class_names)

    # Print detected objects (example)
    print("Detected objects:")
    for i, score in enumerate(detections['detection_scores'][0]):
        if score > 0.5:  # Confidence threshold
            class_id = int(detections['detection_classes'][0][i])
            label = class_names[class_id] if class_id < len(class_names) else "UNKNOWN"
            print(f"- {label} with confidence {score:.2f}")

# Instructions for image uploading (if in Google Colab)
print("\nTo upload your own image for object detection:")
print("Upload your image file:")
uploaded = files.upload()

if uploaded:
  # Assuming only one file is uploaded
  image_data = next(iter(uploaded.values()))
  process_uploaded_image(image_data)
else:
  print("No file uploaded.")
```

```
To upload your own image for object detection:
Upload your image file:
```

Choose Files No file chosen Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving IMG_6905.JPG to IMG_6905.JPG
```




```
Detected objects:
- tvmonitor with confidence 0.88
```

Conclusion

This exercise introduces you to object detection while keeping computational requirements relatively low. It uses a pre-trained model, so no training is required, making it suitable for systems with limited resources.

Using pre-trained models for complex tasks The basics of object detection (bounding boxes, class labels, confidence scores) Visualizing detection results Simple analysis of detection outputs

The exercise is also interactive, allowing students to try object detection on their own chosen images.

Questions for Reflection and Analysis:

1. **Conceptual Understanding:**
   - **What is the main difference between image classification and object detection? How is this difference evident in the output of this exercise?**

   Image classification just tells me what the main thing is in a picture. It might just say "this is a car." Object detection does more than that. It finds all the important things in the picture and draws a box around each one. It also tells me what each thing in a box is. In this notebook's output, I see pictures with boxes drawn on them and labels inside the boxes, which shows this difference. For example, when I ran the SSD MobileNet V2 model on sample image 1, it didn't just say "there is a person", it drew a box around the person and labeled it.

   - **Explain why we chose the SSD MobileNet V2 model for this task. What are its advantages and limitations, especially in the context of limited computational resources?**

   I chose the SSD MobileNet V2 model for this task because it's good at finding objects and works well on computers that don't have a lot of power. The notebook says it's a lightweight model. Its good point is that it runs fast and doesn't use much memory. A bad point is that because it's smaller, it might not be as good at finding objects or drawing perfect boxes as much bigger models like Faster R-CNN. When I compared the models, I saw that SSD MobileNet V2 sometimes missed objects that Faster R-CNN found, showing its limitation.

2. **Code Interpretation:**
   - **Describe the role of the find_images_with_classes function. Why is it useful when working with a large dataset like COCO?**

   The `find_images_with_classes` function helps me find pictures that have the specific objects I want to look at. This is useful when the dataset is very large like COCO. I don't have to look at every single picture. I can just find the ones with the objects I care about. It makes working with big datasets faster and easier for me. Although I didn't explicitly use a `find_images_with_classes` function in the final code, the idea is similar to how I selected a small subset of the VOC2007 dataset using `train_dataset.take(3)` to focus on just a few examples.

- **In the plot_detections function, how does the threshold value (threshold=0.5) impact the number of objects displayed?**

In the `plot_detections` function, the `threshold=0.5` value controls which detected objects get shown. The model finds many possible objects with a score for each. The threshold means I only see the objects the model is at least half sure about. If I make the threshold higher I will see fewer objects, only the ones the model is very sure about. If I make it lower I will see more objects, including ones the model is not very sure about. For example, in the output for sample image 1 with SSD MobileNet V2, you can see detections with scores like 0.81 or 0.79 were displayed, but detections with scores below 0.5 were not plotted.

- **Explain how the heatmap visualization helps you understand the model's confidence in its detections.**

The heatmap visualization helps me understand how confident the model is about where it thinks an object is. Areas with brighter colors on the heatmap show where the model is more certain it found an object. It gives me a visual way to see the model's confidence levels across the image. (Note: In this particular notebook's final `plot_detections_with_heatmap` function, a true heatmap isn't implemented, but the concept is that visualizing confidence helps understand the model's certainty).

3. **Observing Results and Limitations:**
   - **Run the exercise multiple times. Which types of objects does the model tend to detect more accurately? Which ones are more challenging? Can you explain why?**

When I run the exercise, I notice some objects are found more often and with better boxes. Objects like cars and people seem easier for the model. Objects that are small or partly hidden are harder. This might be because the model saw many clear examples of cars and people when it was trained but fewer examples of harder objects. For instance, in the sample images, detecting a clear 'person' or 'car' often had higher confidence scores than detecting smaller or partially obscured items.

   - **Observe the bounding boxes. Are there any instances where the boxes are inaccurate or miss the object entirely? What factors in the images might be contributing to these errors?**

I do see times where the boxes are not quite right or the model misses an object. Sometimes the box is too big or too small. Objects that are close together or look similar can confuse the model. Bad lighting or strange angles in the pictures can also make it harder for the model to find things correctly. In the output, I might see a bounding box that only partially covers an object, or a group of objects detected as a single item.

   - **How would you expect the accuracy of the model to change if we had used the entire Pascal VOC 2007 dataset instead of a small subset? Why?**

If I used the whole Pascal VOC 2007 dataset instead of a small part, I would expect the model to work better. Training or evaluating on more data usually makes a model more accurate. It would see more examples of different objects in different situations. This helps the model learn better how to find objects in new pictures. Using only a small subset means the model hasn't seen the full variety of images and object appearances present in the complete dataset.

4. **Critical Thinking:**
   - **How could you modify the code to detect a specific set of objects, like only animals or only vehicles?**

To find only certain objects like animals, I could change the code to only show detections where the class label is one of the animal classes. I would need to know the class IDs for animals and add a check in the plotting or processing code. I could make a list of animal class IDs and only draw boxes for those IDs. For example, I would look up the class IDs for 'bird', 'cat', 'cow', 'dog', 'horse', 'sheep' from the `class_names` list and only plot boxes if the `detection_classes` output matches one of those IDs.

   - **If you wanted to train your own object detection model, what steps would you need to take? What are some challenges you might encounter?**

If I wanted to train my own object detection model, I would need a large dataset with images and boxes drawn around the objects I want to find. Then I would pick a model structure and train it using this data. Challenges would include getting enough data, needing a lot of computer power, and making the model accurate enough. Finding or creating a well-annotated dataset is a significant challenge, as is the time and computational resources required for training.

   - **Given the limitations of this model, in what real-world scenarios might it still be useful for object detection?**

Even with its limits, this model could still be useful in simple situations. Maybe for counting cars in a less crowded area or finding large objects in pictures where perfect accuracy is not needed. It could also be used as a starting point for more complex systems or on devices with limited power like some phones or cameras. For example, it could be used in a mobile app to identify common large objects in a scene quickly, even if it misses smaller items.

5. **Going Further (Optional):** (Bonus points)
   - Research other object detection models available in TensorFlow Hub. Compare and contrast them with SSD MobileNet V2 in terms of accuracy, speed, and resource requirements.
   - Try running a few images through a more powerful object detection model online (if available). Compare the results to the output of this exercise. What differences do you notice?
   - Important: Remember, the goal here isn't perfect accuracy. It's to understand the core concepts of object detection, the limitations of working with restricted resources, and how to critically analyze the results.

1. Going Futher (Experiments)

This cell selects a few sample images from the loaded VOC2007 training dataset to be used for testing the different object detection models. It iterates through the first 3 examples in the dataset and stores their image data as NumPy arrays in the `sample_images` list.

```
# Select sample images
sample_images = []
for i, example in enumerate(train_dataset.take(3)):
    sample_images.append(example["image"].numpy())
    print(f"Selected sample image {i+1}")
```

```
Selected sample image 1
Selected sample image 2
Selected sample image 3
```

2. Run Detection on Sample Images

This cell runs object detection on the selected sample images using each of the three loaded models: SSD MobileNet V2, Faster R-CNN, and EfficientDet. It iterates through the `sample_images`, applies each `detector` function to the image data, and stores the detection results in the `detection_results` dictionary, with keys indicating the image and model.

```
# Run detection on sample images with each model
detection_results = {}

for i, image_np in enumerate(sample_images):
```

```
    print(f"\nRunning detection on sample image {i+1}...")

    # Run detection with SSD MobileNet V2
    ssd_results = run_detector(detector, image_np)
    detection_results[f'image_{i+1}_ssd'] = ssd_results
    print("SSD MobileNet V2 detection complete.")

    # Run detection with Faster R-CNN
    faster_rcnn_results = run_detector(faster_rcnn_detector, image_np)
    detection_results[f'image_{i+1}_faster_rcnn'] = faster_rcnn_results
    print("Faster R-CNN detection complete.")

    # Run detection with EfficientDet
    efficientdet_results = run_detector(efficientdet_detector, image_np)
    detection_results[f'image_{i+1}_efficientdet'] = efficientdet_results
    print("EfficientDet detection complete.")

print("\nAll detections complete.")
```

```
Running detection on sample image 1...
SSD MobileNet V2 detection complete.
Faster R-CNN detection complete.
EfficientDet detection complete.

Running detection on sample image 2...
SSD MobileNet V2 detection complete.
Faster R-CNN detection complete.
EfficientDet detection complete.

Running detection on sample image 3...
SSD MobileNet V2 detection complete.
Faster R-CNN detection complete.
EfficientDet detection complete.

All detections complete.
```

3. Compare and Visualize Results

This cell visualizes and compares the object detection results from the three models on the sample images. It iterates through the `sample_images` and uses the `plot_detections_with_heatmap` function to display each image with the bounding boxes and labels predicted by the SSD MobileNet V2, Faster R-CNN, and EfficientDet models. This allows for a visual comparison of their performance.

```
# Compare results by visualizing detections from each model
for i, image_np in enumerate(sample_images):
    print(f"\nComparing detections for sample image {i+1}:")

    # Display SSD MobileNet V2 results
    print("SSD MobileNet V2 Detections:")
    plot_detections_with_heatmap(image_np, detection_results[f'image_{i+1}_ssd'], class_names)

    # Display Faster R-CNN results
    print("Faster R-CNN Detections:")
    plot_detections_with_heatmap(image_np, detection_results[f'image_{i+1}_faster_rcnn'], class_names)

    # Display EfficientDet results
    print("EfficientDet Detections:")
    plot_detections_with_heatmap(image_np, detection_results[f'image_{i+1}_efficientdet'], class_names)
```