```
import torch
import torchvision
from torchvision.transforms import functional as F
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files
import time

# Load the pre-trained Faster R-CNN model
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()

# Define COCO object categories
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle',
    'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop
    'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraff
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite'
    'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
    'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', '
    'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'ce
    'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear
]

# Function to upload images
def upload_images():
    uploaded = files.upload()
    file_paths = []
    for fn in uploaded.keys():
        file_paths.append(fn)
    return file_paths

# Function to perform object detection
def detect_objects(image_path, confidence_threshold=0.5):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)

    # Perform inference without calculating gradients
    with torch.no_grad():
        predictions = model(img_tensor)

    filtered_predictions = []
    for i in range(len(predictions[0]['boxes'])):
        label_index = int(predictions[0]['labels'][i])
        # Ensure the label index is within the valid range
        label_index = min(label_index, len(COCO_INSTANCE_CATEGORY_NAMES) - 1)

        # Only consider detections above the confidence threshold
        if predictions[0]['scores'][i] >= confidence_threshold:
            filtered_predictions.append({
                'box': predictions[0]['boxes'][i].tolist(),
                'label': COCO_INSTANCE_CATEGORY_NAMES[label_index],
                'score': predictions[0]['scores'][i].item()
            })

    return filtered_predictions

# Function to annotate and display image
def annotate_and_display_image(image_path, detections):
    img = cv2.imread(image_path)

    # Annotate the image with bounding boxes and labels
    for detection in detections:
        print(f"Detected {detection['label']} with confidence {detection['score']:.2f}
        x1, y1, x2, y2 = map(int, detection['box'])

        # Draw bounding box
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

        # Add label with confidence score
        label_with_confidence = f"{detection['label']} ({detection['score']:.2f})"
        cv2.putText(img, label_with_confidence, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX

    # Convert image back to RGB for displaying with OpenCV
```

error

You

Please explain the error:

```
error: OpenCV(4.10.0) /io/opencv/modules/imgproc/sr
```

Gemini

Hypothesis

The error "OpenCV(4.10.0) /io/opencv/modules/imgproc/src/color.cpp:196: error: (-215:Assertion failed) !_src.empty() in function 'cvtColor'" is raised by the `cv2.cvtColor` function. It means that the input image (`img`) to `cvtColor` is empty.

This typically happens when `cv2.imread` fails to load the image specified by `image_path`. This could be due to a few reasons:

1. **Incorrect `image_path`:** The image path might be incorrect or the image file might not exist at the given location.
2. **Permissions:** The code might not have the necessary permissions to access the file at the specified path.

Suggested Changes

```
import torch
import torchvision
from torchvision import models, transforms
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files
import time
import pandas as pd
import os # Import the os module


# ... (rest of your code) ...


def detect_objects(image_path, confidence_thresho
    # Check if the image file exists
    if not os.path.exists(image_path):
        print(f"Error: Image file not found at {im
        return []  # Return an empty list to indic


    img = cv2.imread(image_path)


    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {
        return []


    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)
```

```
    # Convert image back to BGR for displaying with OpenCV
    img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    # Display the image with annotations (bounding boxes and labels)
    cv2_imshow(img_bgr)

    # Save the image with annotations for download
    output_image_path = 'annotated_' + image_path
    cv2.imwrite(output_image_path, img)
    return output_image_path

# Main function to process multiple images
def process_images(confidence_threshold=0.5):
    # Upload images
    image_paths = upload_images()

    # Process each image
    for image_path in image_paths:
        # Measure time for processing
        start_time = time.time()

        # Perform object detection
        detections = detect_objects(image_path, confidence_threshold)

        # Annotate and display the image with bounding boxes
        output_image_path = annotate_and_display_image(image_path, detections)

        # Measure and print the time taken for processing the image
        elapsed_time = time.time() - start_time
        print(f"Processed {image_path} in {elapsed_time:.2f} seconds.")

        # Provide download link for the annotated image
        files.download(output_image_path)

# Run the processing function with a confidence threshold of 0.5
process_images(confidence_threshold=0.6)
```

```
    # ... (rest of your detect_objects function) .
```

Use code with caution

Explanation of changes:

1. **Import `os`:** Import the `os` module to use for file path operations.
2. **Check for file existence:** The `os.path.exists(image_path)` function is used to check if the file exists before attempting to load it. If the file doesn't exist, a message is printed and an empty list is returned, preventing the error.
3. **Check if image loading was successful:** If `cv2.imread` fails to load the image, it will return `None`. We check for this condition and print an error message if the image could not be loaded. This helps in identifying potential problems with the image file itself, such as corruption or an unsupported format.

Rate this answer

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarr
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarr
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/fasterrcnn_resnet50_fpn_coco-258
100%|██████████| 160M/160M [00:02<00:00, 83.3MB/s]
```

Choose Files b,t,p.jpg

- **b,t,p.jpg**(image/jpeg) - 66796 bytes, last modified: 11/27/2024 - 100% done

```
Saving b,t,p.jpg to b,t,p.jpg
Detected car with confidence 0.99 at [208.40255737304688, 276.6723937988281, 596.3
Detected person with confidence 0.99 at [412.2399597167969, 220.99716186523438, 44
Detected car with confidence 0.99 at [31.752696990966797, 265.10919189453125, 246
Detected person with confidence 0.99 at [217.2163848876953, 244.16639709472656, 23
Detected person with confidence 0.99 at [315.27508544921875, 245.26199340820312, 3
Detected bus with confidence 0.98 at [205.47882080078125, 212.36073303222656, 301
Detected person with confidence 0.97 at [46.591487884521484, 234.72865295410156, 5
Detected car with confidence 0.97 at [681.0034790039062, 262.7712097167969, 758.35
Detected person with confidence 0.96 at [56.417381286621094, 240.04283142089844, 7
Detected bus with confidence 0.95 at [496.1150817871094, 34.42120361328125, 765.72
Detected person with confidence 0.94 at [300.95111083984375, 240.4636688232422, 31
Detected traffic light with confidence 0.94 at [247.85047912597656, 155.6010437011
Detected person with confidence 0.92 at [551.4993286132812, 236.51931762695312, 56
Detected bus with confidence 0.92 at [310.25238037109375, 215.3853302001953, 446.1
Detected person with confidence 0.92 at [298.28533935546875, 236.9044189453125, 36
Detected traffic light with confidence 0.90 at [251.7532196044922, 152.6574707031
Detected person with confidence 0.90 at [539.8410034179688, 238.08367919921875, 54
Detected person with confidence 0.88 at [31.08502197265625, 234.57359313964844, 49
Detected car with confidence 0.87 at [683.47021484375, 227.3993682861328, 745.4094
Detected person with confidence 0.86 at [7.221737861633301, 238.01058959960938, 18
Detected person with confidence 0.84 at [0.0, 236.98960876464844, 12.6067180633544
Detected person with confidence 0.83 at [259.0156555175781, 244.475341796875, 275
Detected bus with confidence 0.82 at [439.6507263183594, 205.1067657470703, 512.96
Detected traffic light with confidence 0.81 at [305.7436828613281, 204.3618774414
Detected person with confidence 0.79 at [15.628734588623047, 238.16407775878906, 3
Detected person with confidence 0.77 at [307.7073974609375, 246.27890014648438, 32
Detected skis with confidence 0.75 at [327.93572998046875, 260.37615966796875, 336
Detected car with confidence 0.69 at [639.5431518554688, 252.79116821289062, 732.3
Detected person with confidence 0.69 at [548.0133056640625, 240.63609313964844, 55
Detected person with confidence 0.69 at [92.49020385742188, 238.79873657226562, 11
Detected car with confidence 0.67 at [680.08251953125, 253.97763061523438, 728.947
Detected traffic light with confidence 0.66 at [454.4228820800781, 182.2203216552
Detected train with confidence 0.63 at [680.53857421875, 170.9878387451172, 760.46
Detected fire hydrant with confidence 0.61 at [142.564697265625, 145.9486083984375
```

```
Processed b,t,p.jpg in 10.69 seconds.
```

```
import torch
import torchvision
from torchvision.transforms import functional as F
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files
import time
import pandas as pd

# Load the pre-trained Faster R-CNN model
```

```
# Load the pre-trained Faster R-CNN model
model = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=True)
model.eval()

# Define COCO object categories
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle',
    'airplane', 'bus', 'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'stop
    'bird', 'cat', 'dog', 'horse', 'sheep', 'cow', 'elephant', 'bear', 'zebra', 'giraff
    'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball', 'kite'
    'skateboard', 'surfboard', 'tennis racket', 'bottle', 'wine glass', 'cup', 'fork',
    'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut', '
    'bed', 'dining table', 'toilet', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'ce
    'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 'scissors', 'teddy bear
]

# Function to upload images
def upload_images():
    uploaded = files.upload()
    file_paths = []
    for fn in uploaded.keys():
        file_paths.append(fn)
    return file_paths

# Function to perform object detection
def detect_objects(image_path, confidence_threshold=0.5):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)

    # Perform inference without calculating gradients
    with torch.no_grad():
        predictions = model(img_tensor)

    filtered_predictions = []
    for i in range(len(predictions[0]['boxes'])):
        label_index = int(predictions[0]['labels'][i])
        # Ensure the label index is within the valid range
        label_index = min(label_index, len(COCO_INSTANCE_CATEGORY_NAMES) - 1)

        # Only consider detections above the confidence threshold
        if predictions[0]['scores'][i] >= confidence_threshold:
            filtered_predictions.append({
                'box': predictions[0]['boxes'][i].tolist(),
                'label': COCO_INSTANCE_CATEGORY_NAMES[label_index],
                'score': predictions[0]['scores'][i].item()
            })

    return filtered_predictions

# Function to annotate and display image
def annotate_and_display_image(image_path, detections):
    img = cv2.imread(image_path)

    # Annotate the image with bounding boxes and labels
    for detection in detections:
        print(f"Detected {detection['label']} with confidence {detection['score']:.2f}
        x1, y1, x2, y2 = map(int, detection['box'])

        # Draw bounding box
        cv2.rectangle(img, (x1, y1), (x2, y2), (0, 255, 0), 2)

        # Add label with confidence score
        label_with_confidence = f"{detection['label']} ({detection['score']:.2f})"
        cv2.putText(img, label_with_confidence, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX

    # Convert image back to BGR for displaying with OpenCV
    img_bgr = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)

    # Display the image with annotations (bounding boxes and labels)
    cv2_imshow(img_bgr)

    # Save the image with annotations for download
    output_image_path = 'annotated_' + image_path
    cv2.imwrite(output_image_path, img)
    return output_image_path

# Function to summarize detections
```

```
# Function to summarize detections
def summarize_detections(detections):
    object_count = {}
    for detection in detections:
        label = detection['label']
        if label not in object_count:
            object_count[label] = 1
        else:
            object_count[label] += 1
    return object_count

# Function to display summary table
def display_summary_table(object_count):
    # Convert the dictionary to a pandas DataFrame
    df = pd.DataFrame(object_count.items(), columns=['Object', 'Count'])
    # Display the table
    print("\nDetection Summary:")
    print(df)

# Main function to process multiple images
def process_images(confidence_threshold=0.5):
    # Upload images
    image_paths = upload_images()

    # Initialize a global dictionary to hold overall object counts
    global_object_count = {}

    # Process each image
    for image_path in image_paths:
        # Measure time for processing
        start_time = time.time()

        # Perform object detection
        detections = detect_objects(image_path, confidence_threshold)

        # Summarize the detections for the current image
        object_count = summarize_detections(detections)

        # Add the counts to the global dictionary
        for obj, count in object_count.items():
            if obj not in global_object_count:
                global_object_count[obj] = count
            else:
                global_object_count[obj] += count

        # Annotate and display the image with bounding boxes
        output_image_path = annotate_and_display_image(image_path, detections)

        # Measure and print the time taken for processing the image
        elapsed_time = time.time() - start_time
        print(f"Processed {image_path} in {elapsed_time:.2f} seconds.")

        # Provide download link for the annotated image
        files.download(output_image_path)

    # After processing all images, display the overall summary table
    display_summary_table(global_object_count)

# Run the processing function with a confidence threshold of 0.6
process_images(confidence_threshold=0.6)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarr
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarr
  warnings.warn(msg)
```

Choose Files b,t,p.jpg

- **b,t,p.jpg**(image/jpeg) **-** 66796 bytes, last modified: 11/27/2024 **-** 100% done

```
Saving b,t,p.jpg to b,t,p (1).jpg
Detected car with confidence 0.99 at [208.40255737304688, 276.6723937988281, 596.3
Detected person with confidence 0.99 at [412.2399597167969, 220.99716186523438, 44
Detected car with confidence 0.99 at [31.752696990966797, 265.10919189453125, 246
Detected person with confidence 0.99 at [217.2163848876953, 244.16639709472656, 23
Detected person with confidence 0.99 at [315.27508544921875, 245.26199340820312, 3
Detected bus with confidence 0.98 at [205.47882080078125, 212.36073303222656, 301
Detected person with confidence 0.97 at [46.591487884521484, 234.72865295410156, 5
Detected car with confidence 0.97 at [681.0034790039062, 262.7712097167969, 758.35
Detected person with confidence 0.96 at [56.417381286621094, 240.04283142089844, 7
Detected bus with confidence 0.95 at [496.1150817871094, 34.42120361328125, 765.72
Detected person with confidence 0.94 at [300.95111083984375, 240.4636688232422, 31
Detected traffic light with confidence 0.94 at [247.85047912597656, 155.6010437011
Detected person with confidence 0.92 at [551.4993286132812, 236.51931762695312, 56
Detected bus with confidence 0.92 at [310.25238037109375, 215.3853302001953, 446.1
Detected person with confidence 0.92 at [298.28533935546875, 236.9044189453125, 36
Detected traffic light with confidence 0.90 at [251.7532196044922, 152.65747070312
Detected person with confidence 0.90 at [539.8410034179688, 238.08367919921875, 54
Detected person with confidence 0.88 at [31.08502197265625, 234.57359313964844, 49
Detected car with confidence 0.87 at [683.47021484375, 227.3993682861328, 745.4094
Detected person with confidence 0.86 at [7.221737861633301, 238.01058959960938, 18
Detected person with confidence 0.84 at [0.0, 236.98960876464844, 12.606718063354
Detected person with confidence 0.83 at [259.0156555175781, 244.475341796875, 275
Detected bus with confidence 0.82 at [439.6507263183594, 205.1067657470703, 512.96
Detected traffic light with confidence 0.81 at [305.7436828613281, 204.36187744146
Detected person with confidence 0.79 at [15.628734588623047, 238.16407775878906, 3
Detected person with confidence 0.77 at [307.7073974609375, 246.27890014648438, 32
Detected skis with confidence 0.75 at [327.93572998046875, 260.37615966796875, 336
Detected car with confidence 0.69 at [639.5431518554688, 252.79116821289062, 732.3
Detected person with confidence 0.69 at [548.0133056640625, 240.63609313964844, 55
Detected person with confidence 0.69 at [92.49020385742188, 238.79873657226562, 11
Detected car with confidence 0.67 at [680.08251953125, 253.97763061523438, 728.947
Detected traffic light with confidence 0.66 at [454.4228820800781, 182.22032165527
Detected train with confidence 0.63 at [680.53857421875, 170.9878387451172, 760.46
Detected fire hydrant with confidence 0.61 at [142.564697265625, 145.948608398437
```

```
Processed b,t,p (1).jpg in 9.88 seconds.

Detection Summary:
          Object  Count
0            car      6
1         person     17
2            bus      4
3  traffic light      4
4           skis      1
5          train      1
6   fire hydrant      1
```

```
import torch
import torchvision
from torchvision import models, transforms
```

```
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files
import time
import pandas as pd

# Load pre-trained Faster R-CNN model
model_faster_rcnn = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=Tru
model_faster_rcnn.eval()

# Load a pre-trained scene recognition model (e.g., ResNet-50 trained on Places365 data
model_scene = models.resnet50(pretrained=True)
model_scene.eval()

# Define image transformations for the scene classification model
scene_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
])

# Define COCO object categories for Faster R-CNN
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'tra
    'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse
    'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee'
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket
    'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'car
    'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 's
]

# Function to perform object detection
def detect_objects(image_path, confidence_threshold=0.5):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)

    with torch.no_grad():
        predictions = model_faster_rcnn(img_tensor)

    filtered_predictions = []
    for i in range(len(predictions[0]['boxes'])):
        label_index = int(predictions[0]['labels'][i])
        label_index = min(label_index, len(COCO_INSTANCE_CATEGORY_NAMES) - 1)

        if predictions[0]['scores'][i] >= confidence_threshold:
            filtered_predictions.append({
                'box': predictions[0]['boxes'][i].tolist(),
                'label': COCO_INSTANCE_CATEGORY_NAMES[label_index],
                'score': predictions[0]['scores'][i].item()
            })

    return filtered_predictions

# Function for scene classification
def classify_scene(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = scene_transform(img_rgb).unsqueeze(0)

    with torch.no_grad():
        outputs = model_scene(img_tensor)

    _, predicted_class = torch.max(outputs, 1)

    # Assuming you have a class-to-label mapping for Places365
    scene_class_map = {  # Just an example
        0: "Indoors", 1: "Urban", 2: "Mountain", 3: "Beach", 4: "Forest", 5: "Desert"
    }

    scene_label = scene_class_map.get(predicted_class.item(), "Unknown")
    return scene_label

# Function to summarize detections
def summarize_detections(detections):
```

```
    object_count = {}
    for detection in detections:
        label = detection['label']
        if label not in object_count:
            object_count[label] = 1
        else:
            object_count[label] += 1
    return object_count

# Main function to process multiple images
def process_images(confidence_threshold=0.5):
    image_paths = upload_images()
    global_object_count = {}

    for image_path in image_paths:
        start_time = time.time()

        # Perform object detection
        detections = detect_objects(image_path, confidence_threshold)

        # Classify the scene (e.g., urban, beach)
        scene_type = classify_scene(image_path)
        print(f"Scene classified as: {scene_type}")

        # Summarize object detections
        object_count = summarize_detections(detections)

        for obj, count in object_count.items():
            if obj not in global_object_count:
                global_object_count[obj] = count
            else:
                global_object_count[obj] += count

        # Annotate and display the image with bounding boxes
        output_image_path = annotate_and_display_image(image_path, detections)

        elapsed_time = time.time() - start_time
        print(f"Processed {image_path} in {elapsed_time:.2f} seconds.")

        files.download(output_image_path)

    # After processing all images, display the overall summary table
    display_summary_table(global_object_count)

# Run the function
process_images(confidence_threshold=0.6)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarr
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarr
  warnings.warn(msg)
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarr
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/
100%|[redacted]| 97.8M/97.8M [00:00<00:00, 149MB/s]
```

Choose Files b,t,p.jpg

- **b,t,p.jpg**(image/jpeg) - 66796 bytes, last modified: 11/27/2024 - 100% done

```
Saving b,t,p.jpg to b,t,p (2).jpg
Scene classified as: Unknown
Detected car with confidence 0.99 at [208.40255737304688, 276.6723937988281, 596.3
Detected person with confidence 0.99 at [412.2399597167969, 220.99716186523438, 44
Detected car with confidence 0.99 at [31.752696990966797, 265.10919189453125, 246
Detected person with confidence 0.99 at [217.2163848876953, 244.16639709472656, 23
Detected person with confidence 0.99 at [315.27508544921875, 245.26199340820312, 3
Detected bus with confidence 0.98 at [205.47882080078125, 212.36073303222656, 301
Detected person with confidence 0.97 at [46.591487884521484, 234.72865295410156, 5
Detected car with confidence 0.97 at [681.0034790039062, 262.7712097167969, 758.35
Detected person with confidence 0.96 at [56.417381286621094, 240.04283142089844, 7
Detected bus with confidence 0.95 at [496.1150817871094, 34.42120361328125, 765.72
Detected person with confidence 0.94 at [300.95111083984375, 240.4636688232422, 31
Detected traffic light with confidence 0.94 at [247.85047912597656, 155.6010437011
Detected person with confidence 0.92 at [551.4993286132812, 236.51931762695312, 56
Detected bus with confidence 0.92 at [310.25238037109375, 215.3853302001953, 446.1
Detected person with confidence 0.92 at [298.28533935546875, 236.9044189453125, 30
Detected traffic light with confidence 0.90 at [251.7532196044922, 152.6574707031
Detected person with confidence 0.90 at [539.8410034179688, 238.08367919921875, 54
Detected person with confidence 0.88 at [31.08502197265625, 234.57359313964844, 49
Detected car with confidence 0.87 at [683.47021484375, 227.3993682861328, 745.4094
Detected person with confidence 0.86 at [7.221737861633301, 238.01058959960938, 18
Detected person with confidence 0.84 at [0.0, 236.98960876464844, 12.6067180633544
Detected person with confidence 0.83 at [259.0156555175781, 244.475341796875, 275
Detected bus with confidence 0.82 at [439.6507263183594, 205.1067657470703, 512.96
Detected traffic light with confidence 0.81 at [305.7436828613281, 204.3618774414
Detected person with confidence 0.79 at [15.628734588623047, 238.16407775878906, 3
Detected person with confidence 0.77 at [307.7073974609375, 246.27890014648438, 32
Detected skis with confidence 0.75 at [327.93572998046875, 260.37615966796875, 336
Detected car with confidence 0.69 at [639.5431518554688, 252.79116821289062, 732.3
Detected person with confidence 0.69 at [548.0133056640625, 240.63609313964844, 55
Detected person with confidence 0.69 at [92.49020385742188, 238.79873657226562, 11
Detected car with confidence 0.67 at [680.08251953125, 253.97763061523438, 728.947
Detected traffic light with confidence 0.66 at [454.4228820800781, 182.2203216552
Detected train with confidence 0.63 at [680.53857421875, 170.9878387451172, 760.46
Detected fire hydrant with confidence 0.61 at [142.564697265625, 145.948608398437
```

```
Processed b,t,p (2).jpg in 12.18 seconds.

Detection Summary:
          Object  Count
0            car      6
1         person     17
2            bus      4
3  traffic light      4
4           skis      1
5          train      1
6   fire hydrant      1
```

```
import torch
import torchvision
from torchvision import models, transforms
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files
import time
import pandas as pd

# Load pre-trained Faster R-CNN model
model_faster_rcnn = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=Tru
model_faster_rcnn.eval()

# Load a pre-trained scene recognition model (e.g., ResNet-50 trained on Places365 data
model_scene = models.resnet50(pretrained=True)
model_scene.eval()

# Define image transformations for the scene classification model
scene_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
])

# Define COCO object categories for Faster R-CNN
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'tra
    'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse
    'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee'
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket
    'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'car
    'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 's
]

# Function to perform object detection
def detect_objects(image_path, confidence_threshold=0.5):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)

    with torch.no_grad():
        predictions = model_faster_rcnn(img_tensor)

    filtered_predictions = []
    for i in range(len(predictions[0]['boxes'])):
        label_index = int(predictions[0]['labels'][i])
        label_index = min(label_index, len(COCO_INSTANCE_CATEGORY_NAMES) - 1)

        if predictions[0]['scores'][i] >= confidence_threshold:
            filtered_predictions.append({
                'box': predictions[0]['boxes'][i].tolist(),
                'label': COCO_INSTANCE_CATEGORY_NAMES[label_index],
                'score': predictions[0]['scores'][i].item()
            })

    return filtered_predictions

# Function for scene classification
def classify_scene(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = scene_transform(img_rgb).unsqueeze(0)

    with torch.no_grad():
        outputs = model_scene(img_tensor)

    _, predicted_class = torch.max(outputs, 1)

    # Assuming you have a class-to-label mapping for Places365
    scene_class_map = {  # Just an example
        0: "Indoors", 1: "Urban", 2: "Mountain", 3: "Beach", 4: "Forest", 5: "Desert"
    }

    scene_label = scene_class_map.get(predicted_class.item(), "Unknown")
```

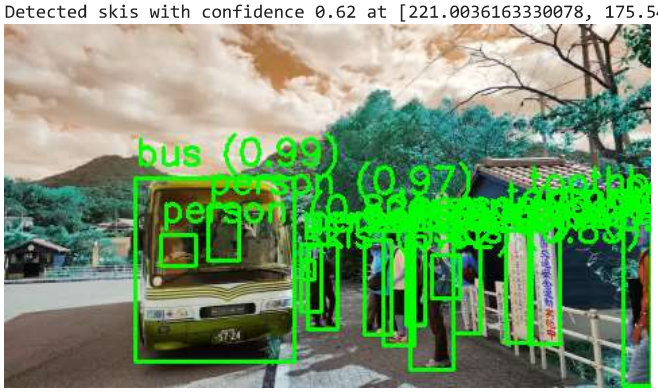
```
    scene_label = scene_class_map.get(predicted_class.item(), "unknown")
    return scene_label

# Function to summarize detections
def summarize_detections(detections):
    object_count = {}
    for detection in detections:
        label = detection['label']
        if label not in object_count:
            object_count[label] = 1
        else:
            object_count[label] += 1
    return object_count

# Main function to process multiple images
def process_images(confidence_threshold=0.5):
    image_paths = upload_images()
    global_object_count = {}

    for image_path in image_paths:
        start_time = time.time()

        # Perform object detection
        detections = detect_objects(image_path, confidence_threshold)

        # Classify the scene (e.g., urban, beach)
        scene_type = classify_scene(image_path)
        print(f"Scene classified as: {scene_type}")

        # Summarize object detections
        object_count = summarize_detections(detections)

        for obj, count in object_count.items():
            if obj not in global_object_count:
                global_object_count[obj] = count
            else:
                global_object_count[obj] += count

        # Annotate and display the image with bounding boxes
        output_image_path = annotate_and_display_image(image_path, detections)

        elapsed_time = time.time() - start_time
        print(f"Processed {image_path} in {elapsed_time:.2f} seconds.")

        files.download(output_image_path)

    # After processing all images, display the overall summary table
    display_summary_table(global_object_count)

# Run the function
process_images(confidence_threshold=0.6)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarr
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarr
  warnings.warn(msg)
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarr
  warnings.warn(msg)
```

Choose Files m.webp

- **m.webp**(image/webp) - 170682 bytes, last modified: 11/27/2024 - 100% done

```
Saving m.webp to m.webp
Scene classified as: Unknown
Detected person with confidence 1.00 at [301.1479797363281, 152.0340118408203, 333
Detected person with confidence 1.00 at [225.46035766601562, 163.9437255859375, 24
Detected person with confidence 0.99 at [459.0631103515625, 138.5209503173828, 486
Detected person with confidence 0.99 at [333.8049621582031, 162.63995361328125, 35
Detected person with confidence 0.99 at [281.84051513671875, 153.5816192626953, 36
Detected bus with confidence 0.99 at [97.21732330322266, 114.53097534179688, 215.5
Detected person with confidence 0.98 at [267.88134765625, 160.94679260253906, 287
Detected person with confidence 0.97 at [151.7953643798828, 135.34364318847656, 17
Detected person with confidence 0.86 at [220.91896057128906, 163.0912628173828, 23
Detected skis with confidence 0.85 at [317.1180114746094, 171.69717407226562, 338
Detected toothbrush with confidence 0.83 at [389.9230651855469, 135.82962036132812
Detected person with confidence 0.82 at [116.33406829833984, 156.6396484375, 142.7
Detected person with confidence 0.75 at [298.5704040527344, 157.5368194580078, 312
Detected toothbrush with confidence 0.70 at [372.4583435058594, 146.5183868408203,
Detected skis with confidence 0.62 at [221.0036163330078, 175.5452423095703, 230.4
```




```
Processed m.webp in 11.00 seconds.

Detection Summary:
       Object  Count
0      person     10
1         bus      1
2        skis      2
3  toothbrush      2
```

Start coding or generate with AI.

```
import torch
import torchvision
from torchvision import models, transforms
import cv2
import numpy as np
from google.colab.patches import cv2_imshow
from google.colab import files
import time
import pandas as pd
import os # Import the os module


# ... (rest of your code) ...

def detect_objects(image_path, confidence_threshold=0.5):
    # Check if the image file exists
    if not os.path.exists(image_path):
        print(f"Error: Image file not found at {image_path}")
        return []  # Return an empty list to indicate failure

    img = cv2.imread(image_path)

    # Check if image loading was successful
    if img is None:
        print(f"Error: Could not load image from {image_path}")
        return []
```

```
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)
# Load pre-trained Faster R-CNN model
model_faster_rcnn = torchvision.models.detection.fasterrcnn_resnet50_fpn(pretrained=Tru
model_faster_rcnn.eval()

# Load a pre-trained scene recognition model (e.g., ResNet-50 trained on Places365 data
model_scene = models.resnet50(pretrained=True)
model_scene.eval()

# Define image transformations for the scene classification model
scene_transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Resize(256),
    transforms.CenterCrop(224),
])

# Define COCO object categories for Faster R-CNN
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus', 'tra
    'fire hydrant', 'stop sign', 'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse
    'zebra', 'giraffe', 'backpack', 'umbrella', 'handbag', 'tie', 'suitcase', 'frisbee'
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket
    'knife', 'spoon', 'bowl', 'banana', 'apple', 'sandwich', 'orange', 'broccoli', 'car
    'chair', 'couch', 'potted plant', 'bed', 'dining table', 'toilet', 'tv', 'laptop',
    'microwave', 'oven', 'toaster', 'sink', 'refrigerator', 'book', 'clock', 'vase', 's
]

# Function to perform object detection
def detect_objects(image_path, confidence_threshold=0.5):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = F.to_tensor(img_rgb).unsqueeze(0)

    with torch.no_grad():
        predictions = model_faster_rcnn(img_tensor)

    filtered_predictions = []
    for i in range(len(predictions[0]['boxes'])):
        label_index = int(predictions[0]['labels'][i])
        label_index = min(label_index, len(COCO_INSTANCE_CATEGORY_NAMES) - 1)

        if predictions[0]['scores'][i] >= confidence_threshold:
            filtered_predictions.append({
                'box': predictions[0]['boxes'][i].tolist(),
                'label': COCO_INSTANCE_CATEGORY_NAMES[label_index],
                'score': predictions[0]['scores'][i].item()
            })

    return filtered_predictions

# Function for scene classification
def classify_scene(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img_tensor = scene_transform(img_rgb).unsqueeze(0)

    with torch.no_grad():
        outputs = model_scene(img_tensor)

    _, predicted_class = torch.max(outputs, 1)

    # Assuming you have a class-to-label mapping for Places365
    scene_class_map = {  # Just an example
        0: "Indoors", 1: "Urban", 2: "Mountain", 3: "Beach", 4: "Forest", 5: "Desert"
    }

    scene_label = scene_class_map.get(predicted_class.item(), "Unknown")
    return scene_label

# Function to summarize detections
def summarize_detections(detections):
    object_count = {}
    for detection in detections:
        label = detection['label']
        if label not in object_count:
```

```
                object_count[label] = 1
            else:
                object_count[label] += 1
        return object_count

    # Main function to process multiple images
    def process_images(confidence_threshold=0.5):
        image_paths = upload_images()
        global_object_count = {}
```

Enter a prompt here

0 / 1000

Responses may display inaccurate or offensive information that doesn't represent Google's views. Learn more