Streamlit Application for Image Component Identification

Import Statements

```
import streamlit as st  # Import the Streamlit library for creating web apps
from PIL import Image  # Import the Python Imaging Library (PIL) for handling images
import torch  # Import PyTorch for working with deep learning models
from torchvision import transforms  # Import the transforms module from torchvision for image transformations
from torchvision.models.detection import fasterrcnn_resnet50_fpn  # Import the Faster R-CNN model
```

- **Streamlit**: Used for creating the web application interface.
- **PIL (Python Imaging Library)**: Handles image loading and processing.
- **PyTorch**: Provides tools for loading and running pre-trained deep learning models.
- **torchvision.transforms**: Contains image transformation utilities.
- **torchvision.models.detection**: Provides pre-trained models for object detection.

Model Loading and Preparation

```
# Load pre-trained Faster R-CNN model
model = fasterrcnn_resnet50_fpn(pretrained=True)  # Load the Faster R-CNN model pre-trained on COCO dataset
model.eval()  # Set the model to evaluation mode (disables training-specific features)
```

- **Model Loading**: `fasterrcnn_resnet50_fpn(pretrained=True)` loads the Faster R-CNN model pre-trained on the COCO dataset.
- **Evaluation Mode**: `model.eval()` sets the model to evaluation mode, disabling features used

only during training, such as dropout.

Image Transformation

```
# Define the transformation
transform = transforms.Compose([
    transforms.ToTensor(),  # Convert the PIL image to a PyTorch tensor
])
```

- **Transformations**: Converts images from PIL format to PyTorch tensors, which are the required input format for the model.

COCO Classes

```
# COCO classes
COCO_INSTANCE_CATEGORY_NAMES = [
    '__background__', 'person', 'bicycle', 'car', 'motorcycle', 'airplane', 'bus',
    'train', 'truck', 'boat', 'traffic light', 'fire hydrant', 'N/A', 'stop sign',
    'parking meter', 'bench', 'bird', 'cat', 'dog', 'horse', 'sheep', 'cow',
    'elephant', 'bear', 'zebra', 'giraffe', 'N/A', 'backpack', 'umbrella', 'N/A',
    'N/A', 'handbag', 'tie', 'suitcase', 'frisbee', 'skis', 'snowboard', 'sports ball',
    'kite', 'baseball bat', 'baseball glove', 'skateboard', 'surfboard', 'tennis racket',
    'bottle', 'N/A', 'wine glass', 'cup', 'fork', 'knife', 'spoon', 'bowl', 'banana',
    'apple', 'sandwich', 'orange', 'broccoli', 'carrot', 'hot dog', 'pizza', 'donut',
    'cake', 'chair', 'couch', 'potted plant', 'bed', 'N/A', 'dining table', 'N/A', 'N/A',
    'toilet', 'N/A', 'tv', 'laptop', 'mouse', 'remote', 'keyboard', 'cell phone', 'microwave',
```

```
    'oven', 'toaster', 'sink', 'refrigerator', 'N/A', 'book', 'clock', 'vase', 'scissors',
    'teddy bear', 'hair drier', 'toothbrush'
]  # List of COCO instance category names
```

- **COCO Class Names**: A list of human-readable labels corresponding to the class indices used in the COCO dataset. This list is used to convert the model's numeric output into meaningful names.

Main Function

```
def main():
    st.title("Analyze Image - Component Identifier")  # Update title
    st.write("Upload an image to identify its components")  # Update description

    # Image upload
        uploaded_file = st.file_uploader("Analyze Image", type=["jpg", "jpeg", "png"])  # Update file uploader label

    if uploaded_file is not None:  # Check if a file is uploaded
        image = Image.open(uploaded_file).convert("RGB")  # Open the uploaded image and convert it to RGB format
        st.image(image, caption='Uploaded Image.', use_column_width=True)  # Display the uploaded image with a caption

        if st.button('Identify Components'):  # Create a button that triggers the identification process
            # Perform object detection
            components = identify_components(image)  # Call the function to identify components in the
```

Streamlit Application for Image Component Identification

image

st.write("Identified components in the image:") # Display a heading for the results

st.write(components) # Display the list of identified components

- **Title and Description**: `st.title()` and `st.write()` set the title and description of the app.
- **File Uploader**: `st.file_uploader()` allows users to upload images. The label is set to "Analyze Image".
- **Image Display**: If an image is uploaded, it is displayed using `st.image()`.
- **Button**: `st.button()` creates a button that triggers the object detection process when clicked.

Identify Components Function

def identify_components(image):

Transform the image

image_tensor = transform(image).unsqueeze(0) # Transform the image to a tensor and add a batch dimension

Perform object detection

with torch.no_grad(): # Disable gradient calculation (useful for inference to save memory and computation)

outputs = model(image_tensor) # Get the model predictions for the image tensor

Extract the predicted class labels

pred_classes = outputs[0]['labels'].numpy() # Extract the predicted class labels and convert them to a NumPy array

pred_scores = outputs[0]['scores'].detach().numpy() # Extract the predicted scores and convert

them to a NumPy array

```
    # Get the class labels with high scores
    keep = pred_scores > 0.9  # Filter out predictions with a score lower than 0.9
      labels = [COCO_INSTANCE_CATEGORY_NAMES[i] for i in pred_classes[keep]]  # Map the filtered class labels to their names

    return labels  # Return the list of identified component labels
```

- **Image Transformation**: Converts the image to a tensor and adds a batch dimension using `.unsqueeze(0)`.
- **Object Detection**:
  - Disables gradient calculation with `torch.no_grad()` to save memory and computation.
  - The model's predictions are obtained for the image tensor.
- **Extracting Predictions**:
  - Extracts predicted class labels and scores.
    - Filters predictions with scores above 0.9 and maps the numeric class labels to their corresponding names using `COCO_INSTANCE_CATEGORY_NAMES`.
- **Return Labels**: Returns the list of identified component labels.

Main Function Execution

```
if __name__ == "__main__":
    main()  # Call the main function to run the app
```

- **Execution Check**: Ensures that the `main()` function is called to run the app when the script is

executed.