In [1]:
```
pip install opencv
```

```
Note: you may need to restart the kernel to use updated packages.
ERROR: Could not find a version that satisfies the requirement opencv (from versions: none)
ERROR: No matching distribution found for opencv

[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [2]:
```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\admin\appdata\local\programs\python\python312\lib\site-packages (2.0.1)
Note: you may need to restart the kernel to use updated packages.
[notice] A new release of pip is available: 24.0 -> 24.2
[notice] To update, run: python.exe -m pip install --upgrade pip
```

In [3]:
```
import cv2
import numpy as np
yolo = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")
classes = []
with open("coco.names", "r") as file:
    classes = [line.strip() for line in file.readlines()]
```

In [4]:
```
layer_names = yolo.getLayerNames()
```

In [5]:
```
try:
    # Check if getUnconnectedOutLayers returns a list of scalars or an array
    unconnected_out_layers = yolo.getUnconnectedOutLayers().flatten()  # Flatten in case it's multi-dimensional
    output_layers = [layer_names[i - 1] for i in unconnected_out_layers]
except Exception as e:
    print(f"Error retrieving output layers: {e}")

print("Output layers:", output_layers)
```

```
Output layers: ['yolo_82', 'yolo_94', 'yolo_106']
```

In [6]:
```
colorRed = (0,0,255)
colorGreen = (0,255,0)
```

In [7]:
```
mage = "image.jpg"
img = cv2.imread(mage)
height, width, channels = img.shape
new_size = (1000, 500)  # Example: resizing to 300x300 pixels
```

In [8]:
```
img = cv2.resize(img, new_size)
height, width, channels = img.shape
blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
yolo.setInput(blob)
outputs = yolo.forward(output_layers)
```

In [9]:
```
class_ids = []
confidences = []
boxes = []
for output in outputs:
    for detection in output:
        scores = detection[5:]
        class_id = np.argmax(scores)
        confidence = scores[class_id]
        if confidence > 0.5:
            center_x = int(detection[0] * width)
            center_y = int(detection[1] * height)
            w = int(detection[2] * width)
            h = int(detection[3] * height)

            x = int(center_x - w / 2)
            y = int(center_y - h / 2)

            boxes.append([x, y, w, h])
            confidences.append(float(confidence))
            class_ids.append(class_id)

indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
for i in range(len(boxes)):
    if i in indexes:
        x, y, w, h = boxes[i]
        label = str(classes[class_ids[i]])
        cv2.rectangle(img,  (x, y),  (x + w, y + h), colorGreen, 3)
        cv2.putText(img, label,  (x, y+5), cv2.FONT_HERSHEY_PLAIN, 2, colorRed, 2)


cv2.imshow("Image", img)
cv2.imwrite("output.jpg",img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

In [12]:
```
import tkinter as tk
from tkinter import filedialog, Label, Button
from PIL import Image, ImageTk
import cv2
import numpy as np

# Load YOLO
yolo = cv2.dnn.readNet("yolov3.weights", "yolov3.cfg")

# Load COCO class labels
classes = []
with open("coco.names", "r") as file:
    classes = [line.strip() for line in file.readlines()]

# Get layer names
layer_names = yolo.getLayerNames()

# Get output layers
try:
    unconnected_out_layers = yolo.getUnconnectedOutLayers().flatten()  # Flatten in case it's multi-dimensional
    output_layers = [layer_names[i - 1] for i in unconnected_out_layers]
except Exception as e:
    print(f"Error retrieving output layers: {e}")

colorRed = (0, 0, 255)
colorGreen = (0, 255, 0)

# Global variables to hold the image and its path
uploaded_image_path = None
analyzed_image = None


def analyze_image(image_path):
    global analyzed_image

    # Load and resize the image
    img = cv2.imread(image_path)
    new_size = (1000, 500)  # Resize image
    img = cv2.resize(img, new_size)
    height, width, channels = img.shape

    # Detecting objects
    blob = cv2.dnn.blobFromImage(img, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    yolo.setInput(blob)
    outputs = yolo.forward(output_layers)

    # Process the outputs
    class_ids, confidences, boxes = [], [], []
    for output in outputs:
        for detection in output:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * width)
                center_y = int(detection[1] * height)
                w = int(detection[2] * width)
                h = int(detection[3] * height)
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(classes[class_ids[i]])
            cv2.rectangle(img, (x, y), (x + w, y + h), colorGreen, 3)
            cv2.putText(img, label, (x, y + 5), cv2.FONT_HERSHEY_PLAIN, 2, colorRed, 2)

    analyzed_image = img
    show_output_image()


def upload_image():
    global uploaded_image_path
    file_path = filedialog.askopenfilename(filetypes=[("Image files", "*.jpg *.jpeg *.png")])
    if file_path:
        uploaded_image_path = file_path
        img = Image.open(file_path)
        img = img.resize((300, 200))  # Resize for display in the app
        img_tk = ImageTk.PhotoImage(img)
        image_label.config(image=img_tk)
        image_label.image = img_tk


def show_output_image():
    global analyzed_image
    if analyzed_image is not None:
        # Convert BGR to RGB for display in Tkinter
        img_rgb = cv2.cvtColor(analyzed_image, cv2.COLOR_BGR2RGB)
        img_pil = Image.fromarray(img_rgb)
        img_pil = img_pil.resize((300, 200))  # Resize for display
        img_tk = ImageTk.PhotoImage(img_pil)
        result_label.config(image=img_tk)
        result_label.image = img_tk


def open_live_camera():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
        if not ret:
            break

        # Detect objects in the live camera feed
        height, width, channels = frame.shape
        blob = cv2.dnn.blobFromImage(frame, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
        yolo.setInput(blob)
        outputs = yolo.forward(output_layers)

        class_ids, confidences, boxes = [], [], []
        for output in outputs:
            for detection in output:
                scores = detection[5:]
                class_id = np.argmax(scores)
                confidence = scores[class_id]
                if confidence > 0.5:
                    center_x = int(detection[0] * width)
                    center_y = int(detection[1] * height)
                    w = int(detection[2] * width)
                    h = int(detection[3] * height)
                    x = int(center_x - w / 2)
                    y = int(center_y - h / 2)
                    boxes.append([x, y, w, h])
                    confidences.append(float(confidence))
                    class_ids.append(class_id)

        indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
        for i in range(len(boxes)):
            if i in indexes:
                x, y, w, h = boxes[i]
                label = str(classes[class_ids[i]])
                cv2.rectangle(frame, (x, y), (x + w, y + h), colorGreen, 2)
                cv2.putText(frame, label, (x, y + 5), cv2.FONT_HERSHEY_PLAIN, 1, colorRed, 2)

        cv2.imshow('Live YOLO Detection', frame)
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()


# Tkinter window setup
root = tk.Tk()
root.title("YOLO Object Detection")

heading_label = Label(root, text="YOLO Object Detection", font=("Helvetica", 16))
heading_label.pack(pady=10)

upload_button = Button(root, text="Upload Image", command=upload_image)
upload_button.pack(pady=5)

scan_button = Button(root, text="Scan via Live Camera", command=open_live_camera)
scan_button.pack(pady=5)

analyze_button = Button(root, text="Analyze Image", command=lambda: analyze_image(uploaded_image_path))
analyze_button.pack(pady=5)

image_label = Label(root)
image_label.pack(pady=10)

result_label = Label(root)
result_label.pack(pady=10)

root.mainloop()
```

In [ ]:

In [ ]: