DIY Deep Learning Project: Image Manipulation with YOLOv8

Introduction

This document walks through using **YOLOv8**, a deep learning model, for object detection and segmentation on images. We use **computer vision** (CV) techniques for manipulating images by identifying and highlighting objects with bounding boxes and overlay masks.

Table of Contents

---

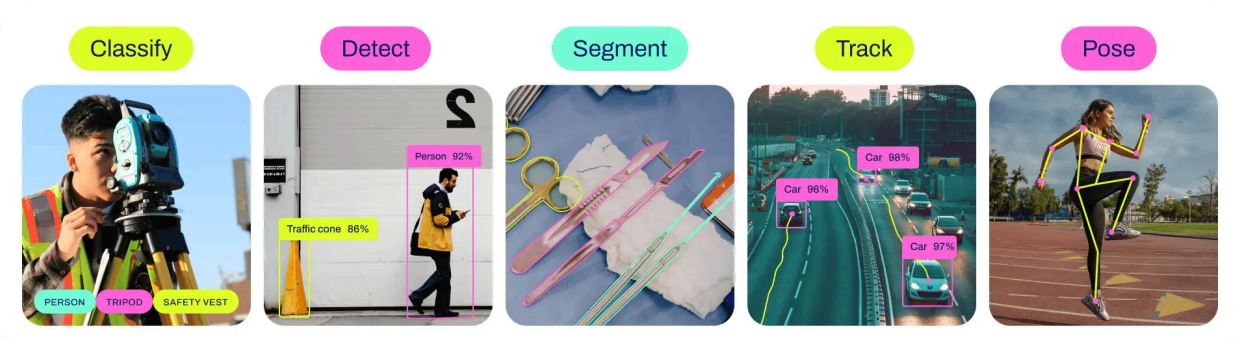
1. Objective of the Project

Our objective is to **detect and segment objects** within an image. With segmentation, we can color-code detected objects and overlay these on the original image. This is a commonly used technique in applications like security systems, autonomous vehicles, and augmented reality.

2. Overview of YOLOv8 and Deep Learning Models

- **YOLOv8** is a recent version of the **YOLO** (You Only Look Once) series of deep learning models for real-time object detection.
- **Object Detection** identifies and labels objects in an image using bounding boxes.
- **Segmentation** divides an image into meaningful parts, allowing us to overlay specific areas (e.g., only cars or people) with color masks.

> **Note:** YOLOv8 can simultaneously perform detection and segmentation, which enhances image manipulation capabilities.



3. Required Tools and Libraries

Here is a list of essential libraries and tools for this project:

- **Python**: The main programming language used.
- **OpenCV**: A computer vision library for image processing.
- **YOLOv8**: The model framework used here, downloaded via the **`ultralytics`** package.
- **Tkinter**: Python's GUI library, used to create an interface for uploading images.

You can install these packages by running the following command:

Copy code(try going for virtual environment before putting the below install using terminal)

```
pip install torch torchvision torchaudio ultralytics opencv-python pillow tkinter
```

4. Concepts: Computer Vision and Image Manipulation

Computer Vision (CV) is a field of AI that trains computers to interpret and understand visual data from the world. Using CV for **image manipulation** allows us to add or modify image content for real-time applications.

Key Terminology

- **Detection**: Identifying specific objects within an image.
- **Segmentation**: Dividing the image into distinct parts based on pixel attributes, allowing for color overlays on specific objects.
- **Bounding Box**: A rectangle that highlights the location of an object.
- **Color Masking**: Applying color overlays to objects for emphasis.

Refer to the images section below to see the detection and segmentation difference

5. Code Implementation and Explanation

Code :-

```
import cv2
import numpy as np
from PIL import Image
from ultralytics import YOLO
import tkinter as tk
from tkinter import filedialog, messagebox

# Load YOLOv8 model (for both detection and segmentation)
model_v8 = YOLO('yolov8x-seg.pt')  # YOLOv8 segmentation model

# Function to process the image using YOLOv8 for both detection and
segmentation
def process_image(image_path):
    image = Image.open(image_path)
    resized_image = image.resize((640, 640))
```

```
    resized_image_np = np.array(resized_image)

    # YOLOv8 prediction
    results_v8 = model_v8.predict(resized_image_np)

    # Extract bounding boxes and apply segmentation masks
    for r in results_v8[0].boxes.data.tolist():
        xmin, ymin, xmax, ymax, confidence, class_id = r
        label = model_v8.names[int(class_id)]
        cv2.rectangle(resized_image_np, (xmin, ymin), (xmax, ymax), (0,
255, 0), 2)
        cv2.putText(resized_image_np, label, (xmin, ymin - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

    # Apply color masks
    if results_v8[0].masks is not None:
        masks = results_v8[0].masks.data.cpu().numpy()
        for mask in masks:
            color_mask = np.random.randint(0, 255, (1, 3), dtype=np.uint8)
            mask_resized = cv2.resize(mask, (resized_image_np.shape[1],
resized_image_np.shape[0]))
            mask_applied = (mask_resized[:, :, None] *
color_mask).astype(np.uint8)
            resized_image_np = cv2.addWeighted(resized_image_np, 1,
mask_applied, 0.5, 0)

    cv2.imwrite('image_with_bboxes_and_masks_v8.jpg', resized_image_np)
    cv2.imshow('YOLOv8 Detections and Segmentation', resized_image_np)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    messagebox.showinfo("Success", "Image processed and saved as
'image_with_bboxes_and_masks_v8.jpg'.")

# Function to handle image upload and trigger detection
def upload_image():
    image_path = filedialog.askopenfilename(filetypes=[("Image files",
"*.jpg *.png *.jpeg")])
    if image_path:
        process_image(image_path)
```

```
# Setup Tkinter GUI
root = tk.Tk()
root.title("YOLOv8 Object Detection and Segmentation")
upload_btn = tk.Button(root, text="Upload Image", command=upload_image,
width=20, height=2, bg="lightblue")
upload_btn.pack(pady=20)
root.mainloop()
```

Code Explanation:-

1. Importing Required Libraries

```
import cv2
import numpy as np
from PIL import Image
from ultralytics import YOLO
import tkinter as tk
from tkinter import filedialog, messagebox
```

- **cv2 (OpenCV)**: Used for image processing tasks like drawing bounding boxes, resizing images, and applying overlays.
- **numpy**: A fundamental library for handling numerical operations and array manipulations, which are crucial for image processing.
- **PIL (Python Imaging Library)**: Specifically, the `Image` class helps with opening, manipulating, and saving images in a format compatible with YOLO and OpenCV.
- **ultralytics**: Provides access to the YOLOv8 model for loading pretrained object detection and segmentation models.
- **tkinter**: Python's GUI toolkit. Here, it creates an interface to let users upload an image and view detection results.

2. Loading the YOLOv8 Segmentation Model

```
# Load YOLOv8 model (for both detection and segmentation)
model_v8 = YOLO('yolov8x-seg.pt')  # YOLOv8 segmentation model
```

- **YOLO Model**: Loads the YOLOv8 model for segmentation (`yolov8x-seg.pt`). This model performs both object detection and segmentation, meaning it can recognize objects and create masks around them.
- **Model File (`yolov8x-seg.pt`)**: This file contains the weights and configuration needed for the YOLOv8 model to detect and segment objects accurately.

3. Image Processing Function:

The main processing happens in this function. This is where YOLOv8 detects objects and applies segmentation masks.

Step 1: Load and Resize the Image

```
image = Image.open(image_path)
resized_image = image.resize((640, 640))
resized_image_np = np.array(resized_image)
```

- **Image Loading**: Opens the image using PIL.
- **Resizing**: Resizes the image to 640x640 pixels, which is optimal for the YOLOv8 model to achieve consistent detection results.
- **Convert to Numpy Array**: YOLO and OpenCV require images as numpy arrays, so this converts the resized PIL image into a format compatible with YOLOv8.

Step 2: YOLOv8 Object Detection and Segmentation

```
# YOLOv8 prediction
results_v8 = model_v8.predict(resized_image_np)
```

- **Prediction**: YOLOv8 processes the image and returns results with detected objects' bounding boxes and segmentation masks (if present).
- **`results_v8`**: Contains two main parts:
    - **Bounding boxes** : Coordinates for each detected object's bounding box.
    - **Masks**: Pixel-based masks for each detected object, enabling segmentation.

Step 3: Extract Bounding Boxes and Labels

#Bounding box:-

```
for r in results_v8[0].boxes.data.tolist():
        xmin, ymin, xmax, ymax, confidence, class_id = r
        label = model_v8.names[int(class_id)]
```

```
        cv2.rectangle(resized_image_np, (xmin, ymin), (xmax, ymax), (0,
255, 0), 2)
        cv2.putText(resized_image_np, label, (xmin, ymin - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
```

Bounding Box Extraction: Each bounding box is defined by the coordinates `(xmin, ymin)` and `(xmax, ymax)`, marking the top-left and bottom-right corners.

- **Confidence and Class ID**: `confidence` represents the detection accuracy, and `class_id` identifies the object's type (e.g., car, person).
- **Label Retrieval**: `model_v8.names` maps each `class_id` to its name, like "car" or "person."
- **Draw Bounding Box and Label**:
    - `cv2.rectangle` draws the bounding box around the detected object.
    - `cv2.putText` places the label (object name) above the bounding box.

Step 4: Extract and Apply Segmentation Masks

```
if results_v8[0].masks is not None:
        masks = results_v8[0].masks.data.cpu().numpy()
        for mask in masks:
            color_mask = np.random.randint(0, 255, (1, 3), dtype=np.uint8)
            mask_resized = cv2.resize(mask, (resized_image_np.shape[1],
resized_image_np.shape[0]))
            mask_applied = (mask_resized[:, :, None] *
color_mask).astype(np.uint8)
            resized_image_np = cv2.addWeighted(resized_image_np, 1,
mask_applied, 0.5, 0)
```

- **Mask Extraction**: `results_v8[0].masks.data` contains the segmentation masks for each object. These masks indicate the object area by pixel.
- **Random Color for Mask**: Each mask gets a random color overlay for easy distinction.
- **Resize Mask**: Each mask is resized to match the processed image's size.
- **Apply Color Mask**: Each mask is overlayed on the original image with `cv2.addWeighted`, which combines the image and mask to create a transparent overlay effect.

4. Save and Display Processed Image

```
cv2.imwrite('image_with_bboxes_and_masks_v8.jpg', resized_image_np)
```

```
 cv2.imshow('YOLOv8 Detections and Segmentation', resized_image_np)
 cv2.waitKey(0)
 cv2.destroyAllWindows()
 messagebox.showinfo("Success", "Image processed and saved as
'image_with_bboxes_and_masks_v8.jpg'.")
```

- **Save Image**: The modified image, including bounding boxes and segmentation masks.
- **Display Image**: Opens a window displaying the final result with `cv2.imshow`.
- **Message Box**: Alerts the user that the image processing is complete.

5. Image Upload Function:

```
def upload_image():
    image_path = filedialog.askopenfilename(filetypes=[("Image files",
"*.jpg *.png *.jpeg")])
    if image_path:
        process_image(image_path)
```

- **File Dialog**: lets the user select an image from their computer.
- **Trigger Process**: Once an image is selected, `process_image` is called to perform YOLOv8-based detection and segmentation.

6. GUI Setup with Tkinter

```
root = tk.Tk()
root.title("YOLOv8 Object Detection and Segmentation")
upload_btn = tk.Button(root, text="Upload Image", command=upload_image,
width=20, height=2, bg="lightblue")
upload_btn.pack(pady=20)
root.mainloop()
```

- **Main Window**: Initializes a Tkinter window with the title "YOLOv8 Object Detection and Segmentation."
- **Upload Button**: Adds a button labeled "Upload Image" to trigger the `upload_image` function.

- **Event Loop**: `root.mainloop()` runs the Tkinter event loop, keeping the window active.
- **Model Loading**: The YOLOv8 segmentation model is loaded, which allows for simultaneous detection and segmentation.
- **Image Processing**: The image is resized to fit YOLOv8's requirements, making processing more efficient.
- **Bounding Box and Labeling**: Detected objects are surrounded by a bounding box, with a label displaying the object type.
- **Segmentation Masks**: Masks are created and applied using OpenCV to overlay colors, enhancing visual segmentation.

Tip: Add screenshots of the bounding boxes, segmentation mask application, and GUI to visualize this section better.

6. Output Analysis and Customization

The processed image, which includes bounding boxes and colored masks, is saved locally and displayed. Customization options include:

- **Adjusting Mask Colors**: Set a specific color for certain object types.
- **Bounding Box Thickness**: Adjust the bounding box border thickness to suit different images.

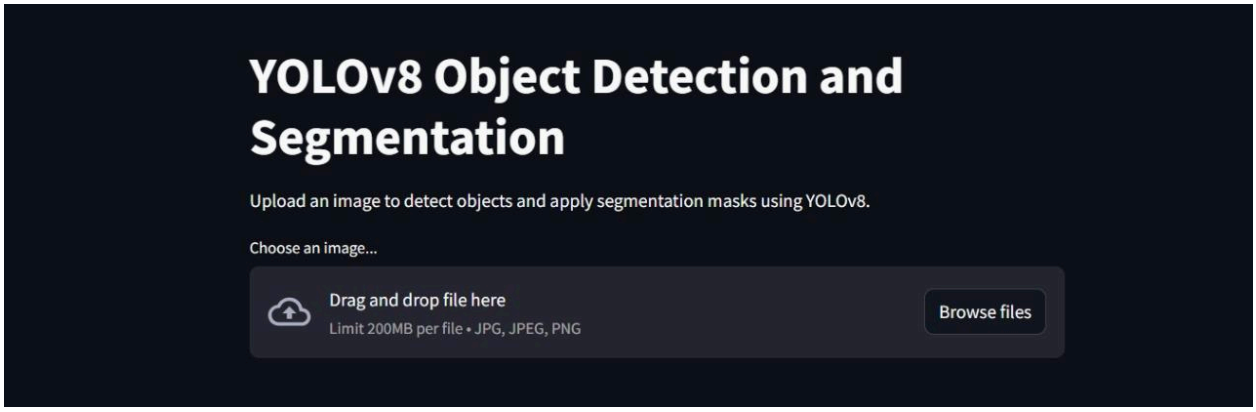
Adding visual examples of different mask colors and bounding box adjustments can help illustrate the effect.

7. Adding Images for Visual Reference

Feel free to add screenshots or online images that:

- Show the GUI interface.
- Highlight bounding box and segmentation mask overlays.
- Display different color overlays for multiple objects.

8.We have also created a streamlit for better UI:-



REFERENCES:-

https://docs.ultralytics.com/

https://www.v7labs.com/blog/yolo-object-detection

https://opencv.org/blog/deep-learning-with-computer-vision/