**Report on**

Project 2: Image Segmentation

Name: Maliha Jannat (马里哈)

Student ID: W2110805009

**Code Explanation**

### Imports



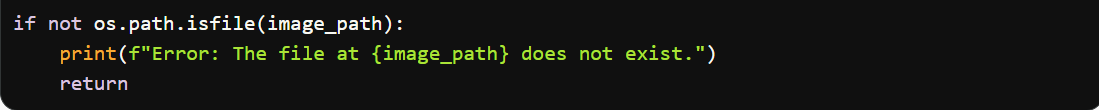
* **cv2**: The OpenCV library for image processing.
* **numpy**: A library for numerical operations, used here for array manipulations.
* **os**: A module to interact with the operating system, primarily for file path operations.
* **matplotlib.pyplot**: A plotting library used to visualize the original and segmented images.

### Function Definition



* **segment\_person**: This function takes one parameter, image\_path, which is a string representing the path to the input image.

### Step 1: File Existence Check



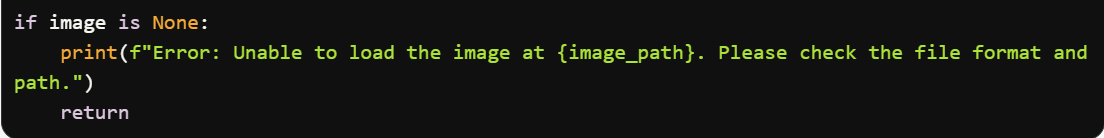
* **image\_path**: Checks if the specified image file exists. If not, an error message is printed, and the function exits.

### Step 2: Load the Image



* **image**: Loads the image from the specified path. The result is a NumPy array representing the image in BGR format.

### Step 3: Check Image Loading



* This ensures that the image was loaded correctly. If not, an error message is displayed.

### Step 4: Convert to Grayscale



* **gray**: Converts the loaded image to grayscale. This simplifies the processing by reducing color information, making it easier to detect contours.

### Step 5: Apply Gaussian Blur



* **blurred**: Applies a Gaussian blur to the grayscale image to reduce noise. The parameters (5, 5) specify the kernel size, and 0 signifies that the standard deviation is calculated from the kernel size.

### Step 6: Otsu’s Thresholding



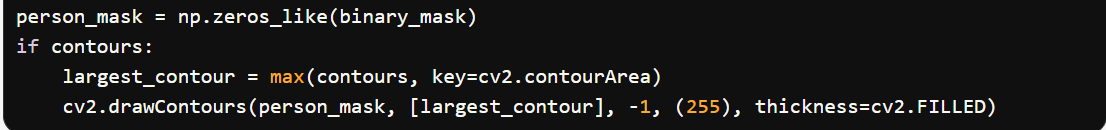
* **binary\_mask**: This binary mask represents the segmented image, where pixels are either 0 (black) or 255 (white). Otsu's method automatically determines the optimal threshold value.

### Step 7: Find Contours



* **contours**: This variable holds the detected contours from the binary mask. The cv2.RETR\_EXTERNAL flag retrieves only the outer contours, and cv2.CHAIN\_APPROX\_SIMPLE compresses horizontal, vertical, and diagonal segments, keeping only their endpoints.

### Step 8: Create a Mask for the Detected Person



* **person\_mask**: Initializes a mask with the same shape as the binary mask, filled with zeros (black).
* **largest\_contour**: Finds the largest contour based on area, assuming it corresponds to the person.
* **drawContours**: Fills the detected contour on the mask with white (255).

### Step 9: Create the Final Binary Image



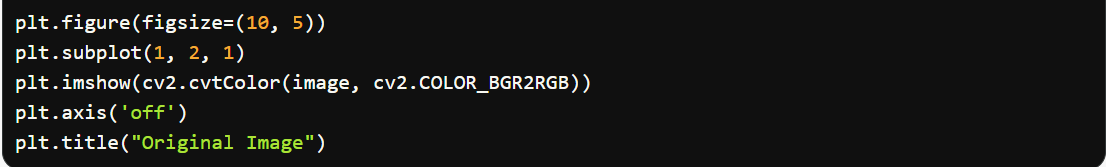
* **final\_mask**: Converts the person mask into an 8-bit binary image, where pixels corresponding to the person are set to 255 (white) and the rest to 0 (black).

### Step 10: Save the Result



* **output\_path**: Specifies the file name for saving the segmented result.
* **cv2.imwrite**: Saves the final mask as an image file.

### Visualization



* Displays the original image using Matplotlib.



* Displays the segmentation result in grayscale.

## Conclusion

This code implements a straightforward approach to segmenting a person from an image by leveraging contour detection and thresholding techniques. It efficiently processes the image to create a binary mask representing the segmented area.

## Codes with Results

