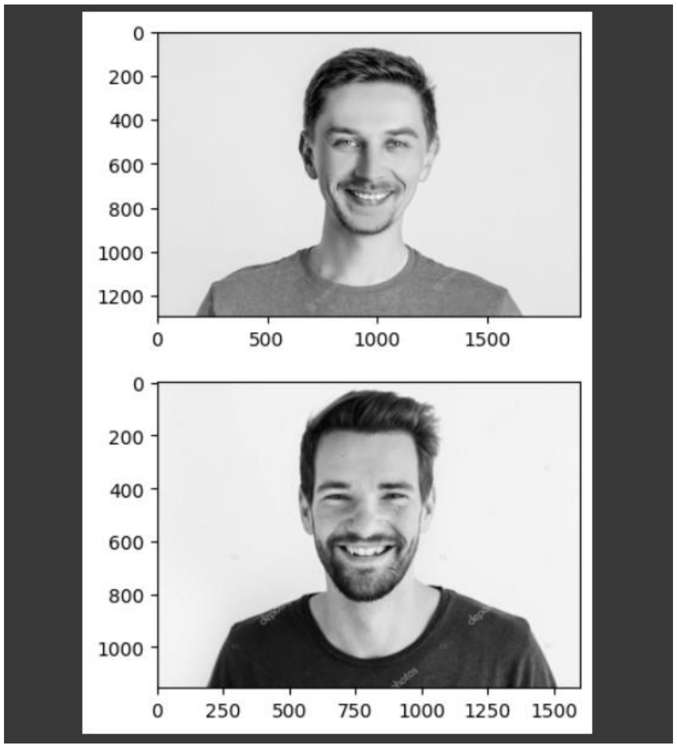
Image processing, assignment 3

Farimah Rashidi (99222040)

Link:
https://colab.research.google.com/drive/1Ay5No9Pj3xQLN5LyYGGassi_ILxtWvht#scrollTo=14QfsZISxHai

Question1:

In this question, we try to combine two human portraits.

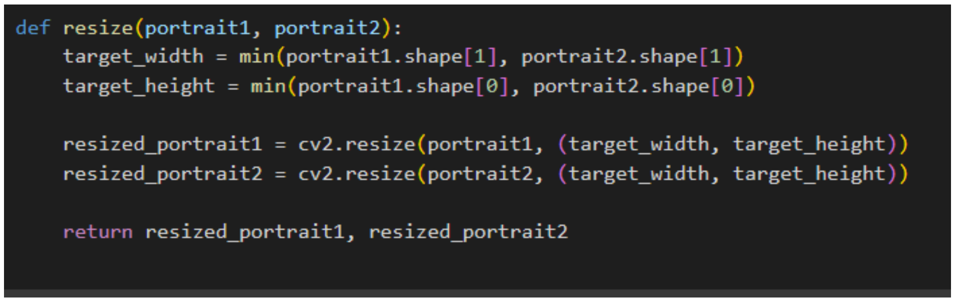


These are two portraits which we want to work on them.

Resize:

As you can see, images are different in sizes. So, we should resize them.

The resize function takes two input images, portrait1 and portrait2, and ensures that they have the same dimensions by resizing them. It determines the target width and height by selecting the minimum width and height values between the two images. Using the OpenCV library's cv2.resize function, the images are then resized to the target dimensions.

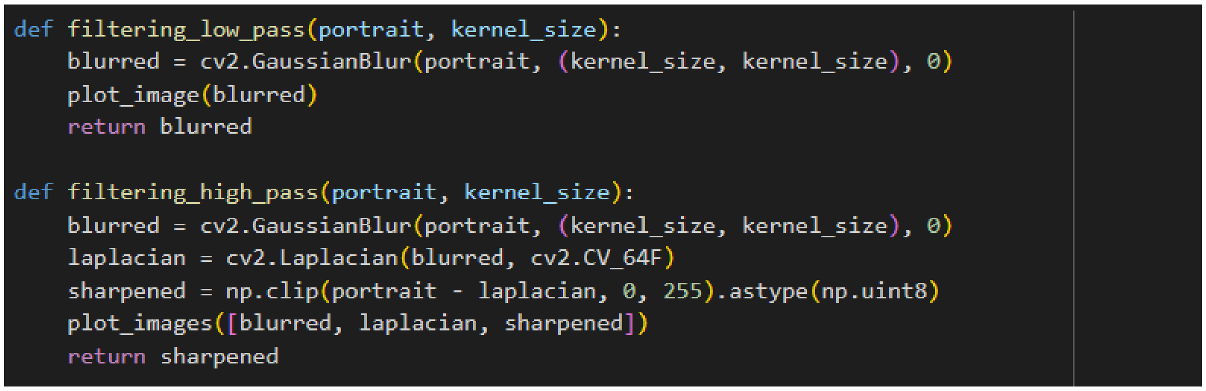


```python
def resize(portrait1, portrait2):
    target_width = min(portrait1.shape[1], portrait2.shape[1])
    target_height = min(portrait1.shape[0], portrait2.shape[0])

    resized_portrait1 = cv2.resize(portrait1, (target_width, target_height))
    resized_portrait2 = cv2.resize(portrait2, (target_width, target_height))

    return resized_portrait1, resized_portrait2
```

Low filter and high filter:

The filtering_low_pass function applies a low-pass filter to the input image using a Gaussian blur. It takes an image and a kernel size as parameters. The image is smoothed by convolving it with a Gaussian kernel of the specified size, which helps to reduce high-frequency noise and smooth out the image.

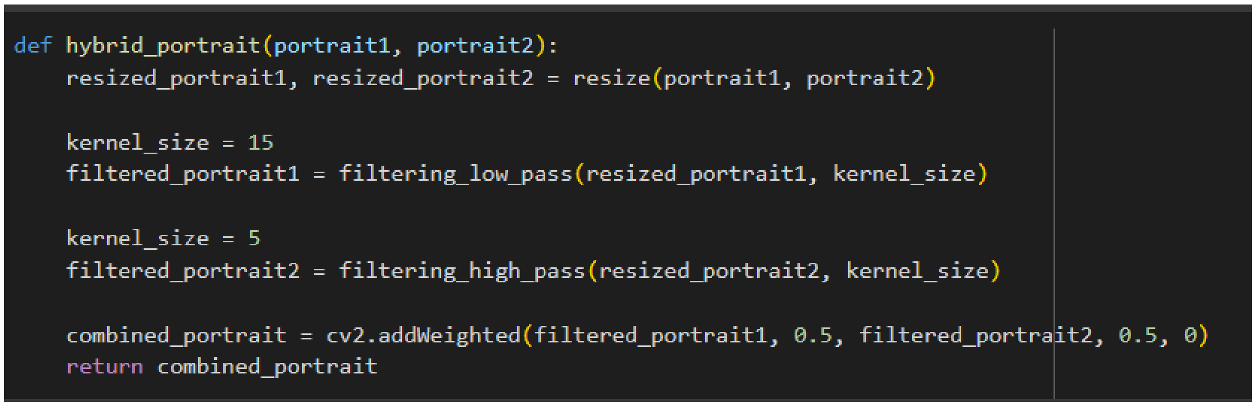
The filtering_high_pass function performs a high-pass filter operation on the input image. It first applies a Gaussian blur to the image using the specified kernel size. Then, it calculates the Laplacian of the blurred image to highlight the high-frequency components or edges. Finally, it subtracts the Laplacian from the original image and performs a clipping operation to ensure the pixel values are within the valid range of 0 to 255. The resulting sharpened image emphasizes the edges and details.



```python
def filtering_low_pass(portrait, kernel_size):
    blurred = cv2.GaussianBlur(portrait, (kernel_size, kernel_size), 0)
    plot_image(blurred)
    return blurred

def filtering_high_pass(portrait, kernel_size):
    blurred = cv2.GaussianBlur(portrait, (kernel_size, kernel_size), 0)
    laplacian = cv2.Laplacian(blurred, cv2.CV_64F)
    sharpened = np.clip(portrait - laplacian, 0, 255).astype(np.uint8)
    plot_images([blurred, laplacian, sharpened])
    return sharpened
```

Combine portraits:

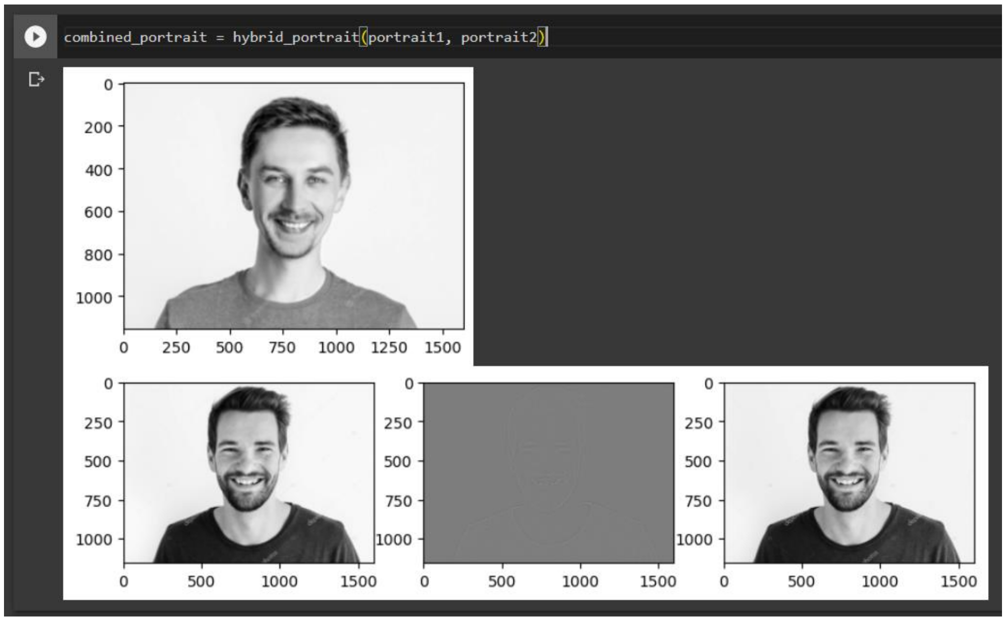
this function creates a hybrid image that combines the low-frequency content from one image and the high-frequency content from another image, resulting in an image that can exhibit different interpretations depending on the viewing distance or scale.



```python
def hybrid_portrait(portrait1, portrait2):
    resized_portrait1, resized_portrait2 = resize(portrait1, portrait2)

    kernel_size = 15
    filtered_portrait1 = filtering_low_pass(resized_portrait1, kernel_size)

    kernel_size = 5
    filtered_portrait2 = filtering_high_pass(resized_portrait2, kernel_size)

    combined_portrait = cv2.addWeighted(filtered_portrait1, 0.5, filtered_portrait2, 0.5, 0)
    return combined_portrait
```

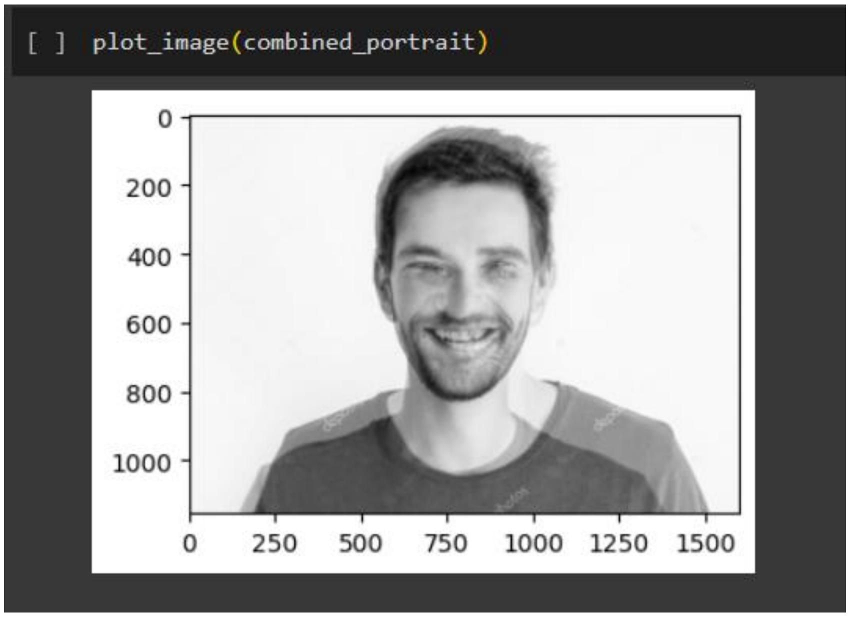
Here you can see results.

Filtered portraits:



```
combined_portrait = hybrid_portrait(portrait1, portrait2)
```

Combined portraits:

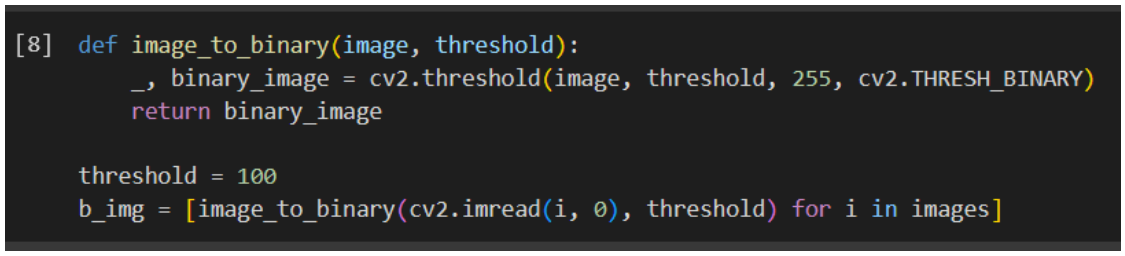


```
[ ]  plot_image(combined_portrait)
```

Question2:

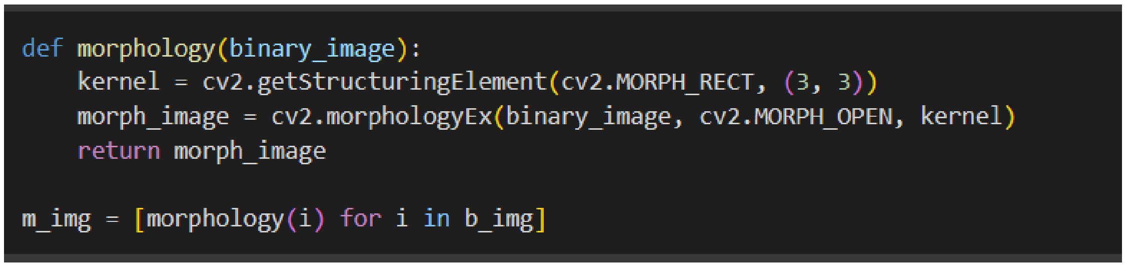
At the first, we get dataset from google drive. Then we choose 120 images as samples.

Then we change images to binary:



```
[8] def image_to_binary(image, threshold):
        _, binary_image = cv2.threshold(image, threshold, 255, cv2.THRESH_BINARY)
        return binary_image

    threshold = 100
    b_img = [image_to_binary(cv2.imread(i, 0), threshold) for i in images]
```

After that, we apply morphology function.

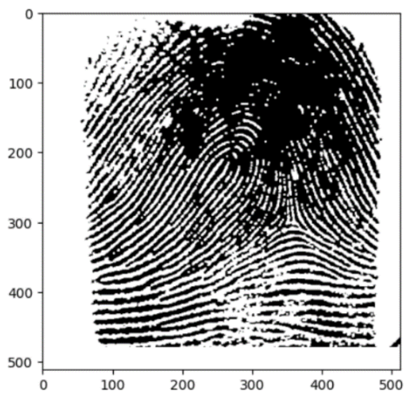


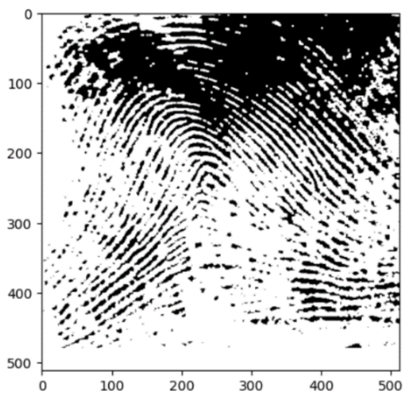
```
def morphology(binary_image):
    kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
    morph_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel)
    return morph_image

m_img = [morphology(i) for i in b_img]
```

`morphology` applies morphological opening, a combination of erosion and dilation, using a 3x3 rectangular kernel, to a binary image. It then creates a list comprehension to iterate over a list of binary images and applies the `morphology` function to each image, collecting the resulting morphologically processed images in a new list called `m_img`.

Then we use an edge detection function which applies the Canny edge detection algorithm to a list of morphologically processed images, producing a new list of edge-detected images. (You can see the code of this part in code file).

We can see the results until here:





You can see other 118 photos in code file.