Image Processing

Assignment 3

Pouria Malek Khayat

Exercise 1

In this part we are required to combine two images. These four images are gathered from the internet.

Resizing the images

One simple but crucial part is resizing the images to be the same size. This is necessary since we can not combine them if they are not the same size. Here is how:

```
img1_resize = cv2.resize(img1, (w, h))
img2_resize = cv2.resize(img2, (w, h))
```

W and **h** are the minimum width and height among the two images.

Applying high filter

This is how we apply the high filter:

apply a Gaussian blur to the input image using a specified kernel size. Gaussian blur helps to reduce noise and smooth out the image.

Then, we apply the Laplacian operator to the blurred image. The Laplacian operator enhances the edges and details in the image.

Now, we calculate the sharpened image by subtracting the Laplacian image from the original image.

By performing these steps, the function applies a high-pass filtering technique using Gaussian blur and Laplacian operator to enhance the edges and details in the input image, resulting in a sharpened image.

Applying low filter

First we apply a Gaussian blur to the input image using a specified kernel size. The Gaussian blur operation smooths out the image and reduces high-frequency components, effectively acting as a low-pass filter.

This will effectively reduce noise and smoothing out the image. This results in a low-pass filtered image.

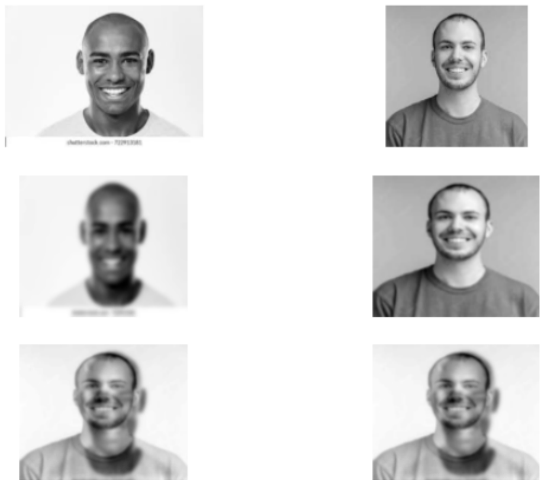
Combining the images

After applying the filters, we combine the images using the following code:

```python
combined_image = cv2.addWeighted(filtered_img1, 0.5, filtered_img2, 0.5, 0)
```

This will use 50% of each image as weight to make the combined image. Full results can be seen in the notebook. Here is an example.

```
Original Images - First row
Filtered Images - Second row
Combined Image - Third row
```



Exercise 2

First, we take the images and convert them to binary form using the following code;

```
def img_to_binary(image, threshold):
  _, binary_image = cv2.threshold(image, threshold, 255, cv2.THRESH_BINARY)
  return binary_image
```

This function extracts the binary image with the specified threshold.

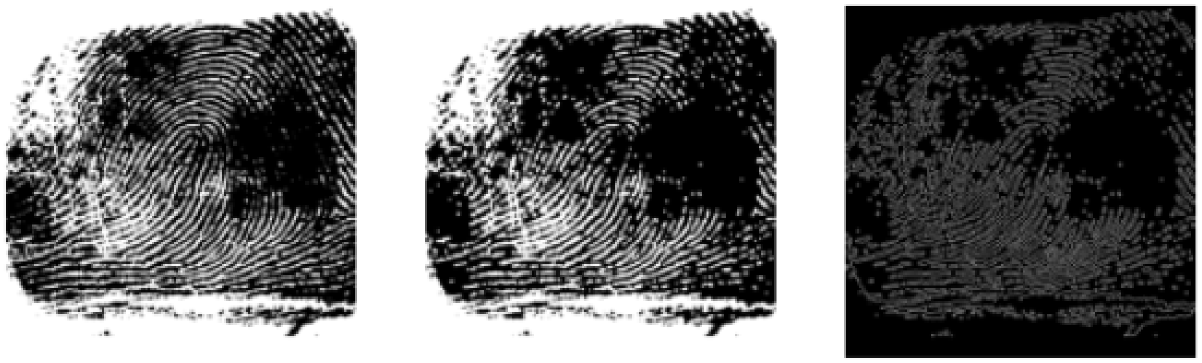
Next, we apply the **morphology_on_img** function.

First, it creates a rectangular structuring element of size 3x3. The structuring element defines the shape and size of the neighborhood used for morphological operations.

Next, it performs erosion on the binary image using the specified kernel. Erosion is a morphological operation that erodes away the boundaries of foreground objects. It shrinks the foreground regions and removes small isolated regions or thin connections.

Now, it performs dilation on the eroded image using the same kernel. Dilation is a morphological operation that expands the boundaries of the foreground objects. It helps to fill in gaps, connect broken parts, and make the foreground regions more prominent.

After that, we extract the edges from the filtered images using Canny filter.



The above photo shows the images during transition which we discussed in detail above.

Classifying the images

This is how the algorithm works:

Applies Gaussian blur to reduce noise in the image. thresholds the blurred image to create a binary image. Finds contours in the binary image. Selects the contour with the maximum area. Computes the bounding rectangle around the contour. Calculates the Hu moments of the contour to estimate the orientation angle. Computes the rotation matrix based on the estimated angle and the center of the bounding rectangle.

We preprocess the image then use the **train** function to train the classifier using a list of fingerprints and their corresponding class labels. It iterates over the fingerprints.

Now, we use the classify function to calculate the similarity between the rotation matrix of the new fingerprint and the training rotation matrices.

Link to google colab

https://colab.research.google.com/drive/11zrI3kbrTPND8OZeOUk9wzyaPgCRvA7T#scrollTo=NYstW4Kd4WS0