CENG466-Fundamentals of Image Processing Report III

Sena Nur Şengül
Computer Engineering
Middle East Technical University
Ankara, Turkey
e231049@metu.edu.tr

I. Introduction

In this assignment, I aimed to discuss face detection techniques and pseudo-coloring for gray images.I have used python3.10 with matplotlib, numpy, cv2,os skimage libraries.

II. FACE DETECTION

A. Introduction

Face detection is a technology that uses machine learning algorithms to identify and locate human faces in digital images or videos.
First part of assignment, I am required to implement face detection algorithm on 1source.png, 2_source.png,3_source.png based on skin color.Our pseudo-color of my implementation is:
1)Load the image into memory and convert it to HSV.
2Apply a Gaussian blur to the image to reduce noise
3)Use the threshold method to create a binary image
4)Use the findContour method to detect the contours in the image
5)Loop through each contour and check if it is a face by comparing its size and shape to known face patterns
6)If a contour matches the known face patterns, draw a bounding box around it and label it as a face
7)Save the image with the detected faces and display it to the user.

B. Implementation

I implemented face detection two ways. First way is using Haarcascade which is cascade classifier to detect objects,faces.However, it is not my implementation , also it works very well.

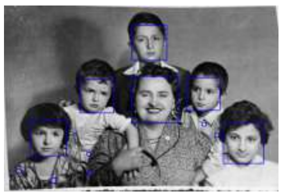


Fig. 1. Example result of cascade classifier

If we look at my implementation: My code first converts the input image from the BGR color space to the HSV color space. This is done using the cv2.cvtColor() method, which takes the image and the desired color space conversion as arguments.

Next, code defines a lower and upper threshold for the skin color in the HSV color space. These values are defined as arrays of three integers, representing the minimum and maximum acceptable values for the Hue, Saturation, and Value components of the HSV color space, respectively.

The code then uses these thresholds to create a mask that filters out all pixels in the image that are not within the specified range of skin colors. This is done using the cv2.inRange() method, which takes the image, the lower threshold, and the upper threshold as arguments and returns a binary image where pixels within the specified range are set to 1 and all other pixels are set to 0.

Next, the code applies the mask to the input image using the cv2.bitwiseand() method. This method takes the input image and the mask as arguments and returns a new image where only the pixels that were set to 1 in the mask are preserved. This effectively filters out all non-skin pixels from the input image.

The code then converts the filtered image to grayscale using the cv2.cvtColor() method, which is necessary for some of the subsequent processing steps.

Next, the code applies a Gaussian blur to the grayscale image using the cv2.GaussianBlur() method. This method takes the image and the size of the blur kernel as arguments and returns a new image where the input image has been smoothed using a Gaussian filter. This helps to reduce noise and make it easier to detect edges in the image.

The code then applies a threshold to the blurred image using the cv2.threshold() method. This method takes the image and the threshold value as arguments and returns a binary image where pixels with values above the threshold are set to 1 and all other pixels are set to 0. This helps to isolate the skin regions in the image and make it easier to detect faces.

Next, the code uses the cv2.findContours() method to detect contours, or outlines, in the thresholded image. This method takes the image and some additional parameters as arguments and returns a list of contours and some additional information.

The code then loops over the list of contours and uses the cv2.contourArea() method to compute the area of each contour. If the area of a contour is greater than a specified threshold (5000 in this case), the code uses the cv2.boundingRect() method to compute the bounding box of the contour. This method returns the coordinates of the top-left and bottom-right corners of the bounding box.

Finally, the code uses the cv2.rectangle() method to draw a rectangle around the face using the bounding box coordinates. The cv2.rectangle() method takes the image, the coordinates of the top-left and bottom-right corners of the rectangle, the color of the rectangle, and the thickness of the line as arguments.

After all of the faces in the image have been detected and marked with rectangles, the code returns the modified image.

C. Results

Haar-cascade algorithm result:
(haarcascade_frontalface_default.xml)

Fig. 2. 1source.png

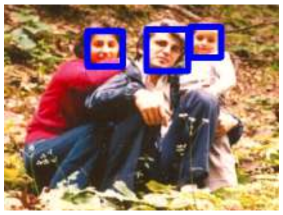


Fig. 3. 1facesh.png

Fig. 4. 2source.png

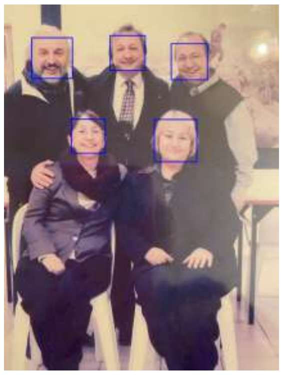
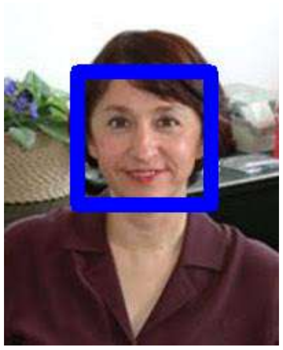
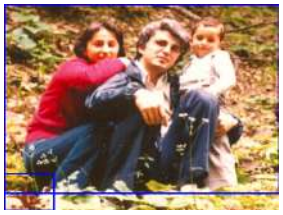


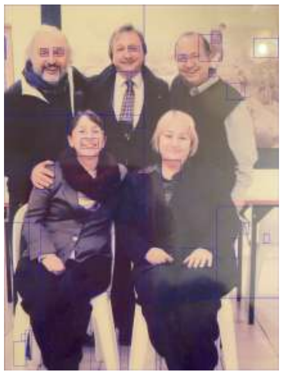
Fig. 5. 2facesh.png

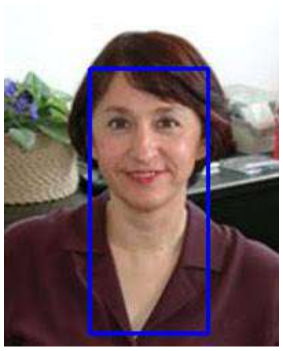
Fig. 6. 3source.png



Results My implementation:







I saw Haarcascade is true but my algorithm have some problems. I observed there are a lot of reasons like :
My code uses a hard-coded set of HSV color range thresholds to create a mask for identifying skin tones. However, these values may not work well for all images and may not be robust to different lighting conditions.

My code uses a fixed threshold for identifying contours in the image. This may not work well for all images, as the optimal threshold value can vary depending on the image content and lighting conditions.

My code uses a fixed contour area threshold for identifying faces in the image. However, this threshold may not be appropriate for all images, as the size of faces can vary significantly.

III. PSEUDO-COLORING

A. Introduction

In this part, I am required to color to gray images using colors of rgb images. Pseudo coloring is a technique used to

display a monochrome image or video as a color image. It is achieved by assigning a different color to each intensity value in the image. Pseudo coloring is often used to visualize data that is not naturally colored, such as medical images or scientific data, in order to make the information easier to interpret or to highlight specific features of the data. It can also be used as an artistic effect to create an image that appears to be in color, even though it is actually a grayscale image.
My algoritm is pseudo - coloring is: Convert the source image to grayscale.
Create an empty array to store the color mapping.
Iterate through each pixel in the source image.
For each pixel, store the grayscale value and RGB color in the color mapping array.
Create an empty array to store the resulting image. Iterate through each pixel in the input image.
For each pixel, find the corresponding RGB color in the color mapping array and store it in the resulting image.
Return the resulting image.

B. Implementation

My code is a function that takes in two images: an ”img” and a ”source” image. The function first converts the source image to grayscale and creates empty color map array with 256 rows and 3 columns.

The function then loops through every pixel in the grayscale source image and assigns the corresponding pixel value in the original source image to the color map array at the index of the grayscale pixel value.

Next, the function creates a new empty result image with the same dimensions as the input ”img” and loops through every pixel in the ”img” image. For each pixel, the function looks up the corresponding color value in the color map array using the pixel value as the index and assigns that color value to the current pixel in the result image.

Finally, the function returns the resulting colored image.

C. Results

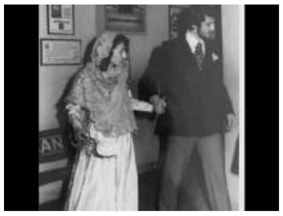


Fig. 11. 1.png

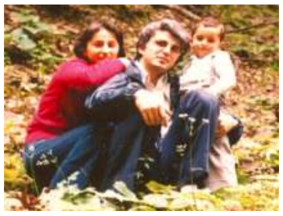


Fig. 12. 1source.png

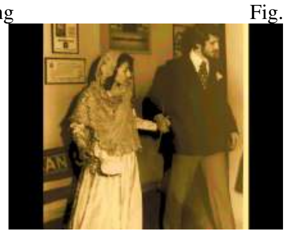


Fig. 13. 1colored.png

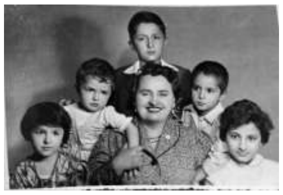


Fig. 14. 2_.png

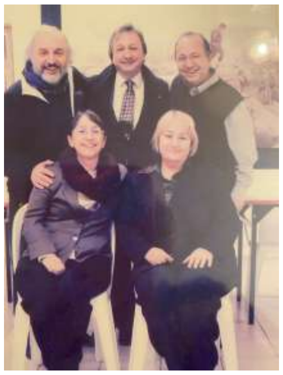


Fig. 15. 2_source.png

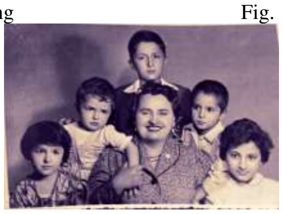


Fig. 16. 2_colored.png

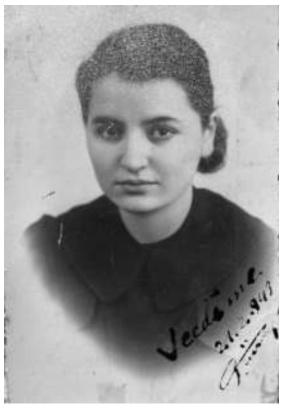


Fig. 17. 3.png

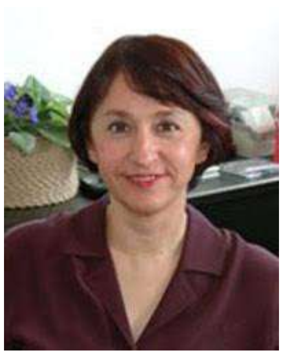


Fig. 18. 3source.png

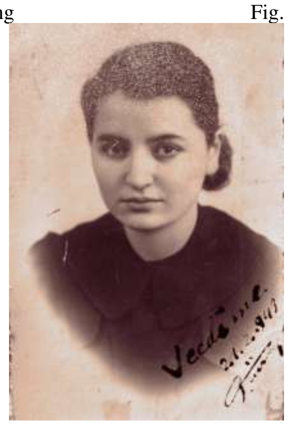


Fig. 19. 3colored.png

Images that has been colored using the color mapping created from the source images. The resulting image will have the same dimensions as the original image, but the pixel values will be replaced with the corresponding RGB values from the color mapping. This can be used to give a grayscale image a more realistic or stylized look by using the colors from another image as a reference.

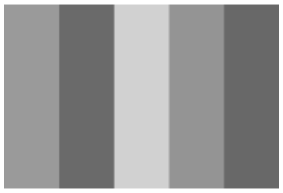


Fig. 20. 4_.png

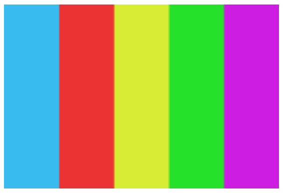


Fig. 21. 4_source.png

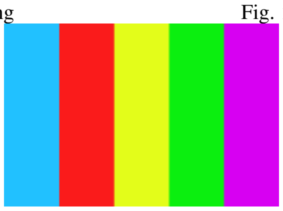


Fig. 22. 4_colored.png

IV. PLOTTING

A. Introduction

In this part, I plotted HSI and RGB channels of the pseudo-colored images which created in pseudo-coloring. The H, S, and I channels represent the hue, saturation, and value of the image, respectively, in the HSV color space. The R, G, and B channels represent the red, green, and blue channels of the image, respectively, in the RGB color space.

B. Implementation

Codes in this part is used to split an image into its color channels and display them separately. The image is first converted from BGR (Blue, Green, Red) to HSV (Hue, Saturation, Value) and then to RGB (Red, Green, Blue). The code then displays each of these channels separately in a series of subplots, using the 'gray' colormap to display the images in grayscale. The resulting plots show the individual color channels of the image, allowing the user to see how each channel contributes to the overall appearance of the image.

C. Results

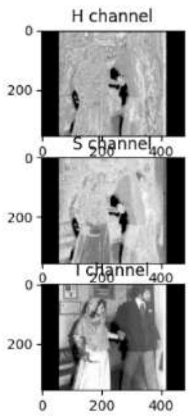


Fig. 23. 1-hsi.png

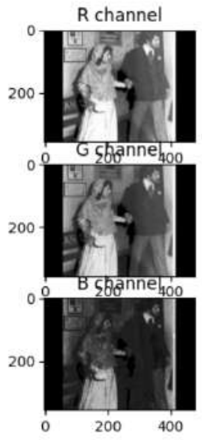


Fig. 24. 1-rgb.png

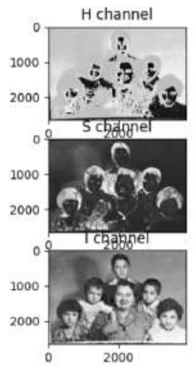


Fig. 25. 2-hsi.png

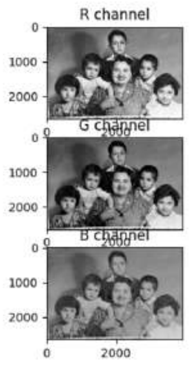


Fig. 26. 2-rgb.png

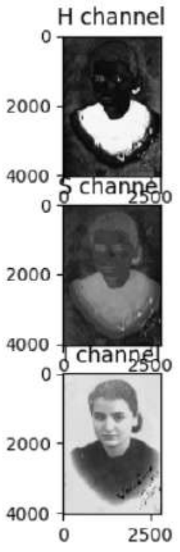


Fig. 27. 3-hsi.png

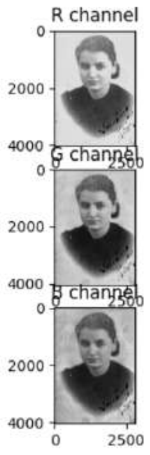


Fig. 28. 3-rgb.png

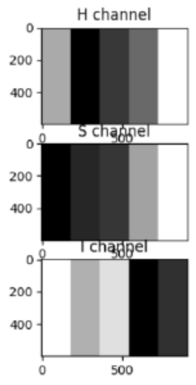


Fig. 29. 4-hsi.png

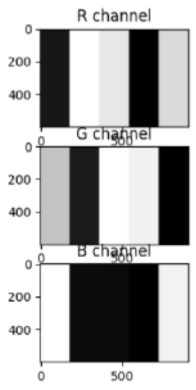


Fig. 30. 4-rgb.png

In a pseudo-colored image created using the HSI color model, the hue channel would represent the color map used to transform the grayscale image to color, the saturation channel would represent the intensity of the color map, and the intensity channel would represent the original grayscale intensity values.

In a pseudo-colored image created using the RGB color model, the red, green, and blue channels would represent the red, green, and blue components of the color map, respectively, and the intensity of each channel would represent the intensity of the corresponding color component.

Overall, the main difference between HSI and RGB pseudo-colored images is the way that the color is represented. In HSI pseudo-colored images, the color is represented by the hue and saturation channels, while in RGB pseudo-colored images, the color is represented by the red, green, and blue channels.

V. EDGE DETECTION

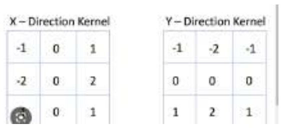
A. Introduction

Final part is in pseudocoloring section. I will discuss results of edge detection of colored images.Edge detection is a technique used in image processing to detect the boundaries or edges of objects in an image. It involves identifying the points in an image where the intensity or color changes significantly, which typically corresponds to the edges of objects in the image. I implemented this pseudo-code:

```
1)Determine the appropriate color space for the image
2)If the color space is RGB, set the image color to the input image
3)If the color space is HSI, convert the input image to HSI and set the image color to the result
4)Calculate the gradients in the x and y directions using the Sobel operator
5)Calculate the magnitude of the gradients
6)Normalize the gradients by dividing by the maximum value and multiplying by 255
7)Convert the gradients to uint8 data type
8)Return the edge-detected image
```

B. Implementation

My code is a function that takes an image and a color space as input and returns an edge-detected version of the image. The first block of code converts the image to the specified color space if necessary (either RGB or HSI). The next two lines calculate the gradients of the image in the x and y directions using the Sobel operator. The gradients are then combined to calculate the magnitude of the gradients. The gradients are then normalized to a range of 0-255 and converted to unsigned 8-bit integers. Finally, the edge-detected image is returned. Sobel operators:



X – Direction Kernel

-1	0	1
-2	0	2
[illegible]	0	1

Y – Direction Kernel

-1	-2	-1
0	0	0
1	2	1

Fig. 31.

C. Results

The resulting images have higher intensity values (closer to 255) at locations where edges are detected, and lower intensity values (closer to 0) in areas without edges.I realized images of HSI,RGB channels of 2source.png are very dark but work.

Fig. 32. 4.png

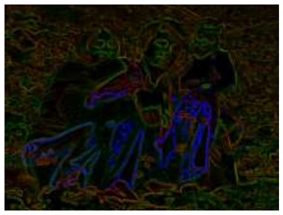


Fig. 33. ED images of 1source.png in HSI channels

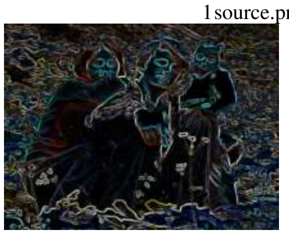


Fig. 34. ED images of 1source.png in RGB channels

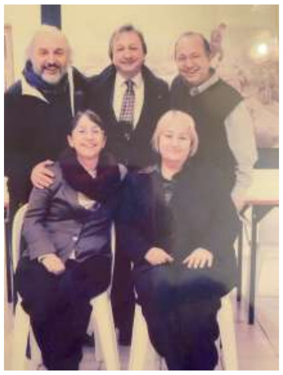


Fig. 35. 2.png

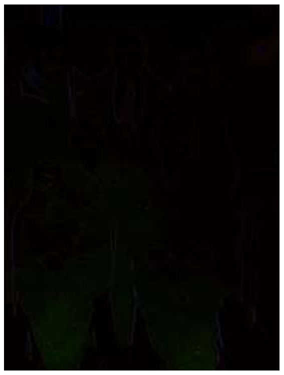


Fig. 36. ED images of 2source.png in HSI channels

Fig. 37. ED images of 2source.png in RGB channels

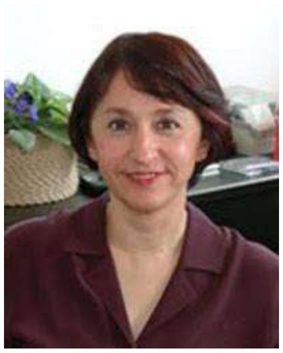


Fig. 38. 4.png

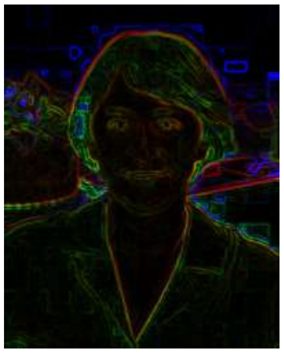


Fig. 39. ED images of 3source.png in HSI channels

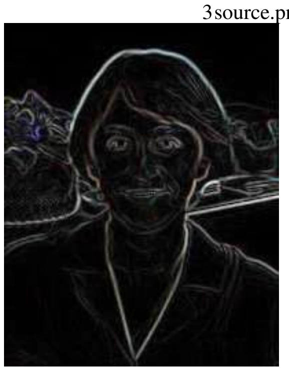


Fig. 40. ED images of 3source.png in RGB channels