CENG466, Fall 2022, THE 3

1st Fırat Ağış
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
e2236867@ceng.metu.edu.tr

2nd Robin Koç
Department of Computer Engineering
Middle East Technical University
Ankara, Turkey
e2468718@ceng.metu.edu.tr

***Abstract*—**
***Index Terms*—**

I. Face Detection

For face detection, we propose an algorithm that use k-means to separate color groups, chooses the color group that is most likely to include skin colors and then uses morphological operations and convolution operation to detect areas that are most likely to include faces in the image. Our algorithm posses 3 distinct stages:

1) *Preprocessing:* Process the image to be more suitable for our algorithm and performs some processes to make the algorithm run a lot faster.
2) *K-Means:* Using k-means algorithm with randomized starting means to detect the pixels that are most likely to be in the color of skin tones.
3) *Context Based Postprocessing:* Using morphological operations and convolution operation to detect which clusters of the pixels that are found in the previous step to be the part of a face.

A. Preprocessing

1) We first read the relevant image. Shown in Row 1 of Figure 1.
2) (Optional) We perform histogram equalization to differentiate skin tones from similar colors in the image, most notably, the leaves in the 1_source.png and shirt and hair in 3_sources.png. Shown in Row 2 of Figure 1.
3) (Optional) We down-sampled the 2_source.png by a factor of 9 in order to complete the algorithm in a reasonable time.
4) We bit-sliced all images, eliminating the least significant bit. Colors that are differentiated by a single bit are most likely to be in the same color group, meaning this operation affected the quality of the k-means algorithm minimally while reducing the size of the color space by a factor of $2^3 = 8$. Shown in Row 3 of Figure 1.

B. K-Means

1) We performed k-means algorithm with randomized initial means, acquiring means and colors closest to that means.

Fig. 1. Face detection progress for preprocessing and k-means stages

2) We only took the colors whose means have the greatest red component using the knowledge that all faces are reddish in the context of color space.
3) We eliminated any colors whose red component is less then their blue or green component using the same logic as the previous step. Shown in Row 4 of Figure 1.

C. Context Based Postprocessing

1) (Optional) While processing 1_source.png and 3_sources.png, we used the assumption of faces being the central focus of images to eliminate pixels on the outer parts of the image. Shown in Row 1 of Figure 2.
2) We used opening operation with a 5x5 element to eliminate noise from the image. Shown in Row 2 of Figure 2.

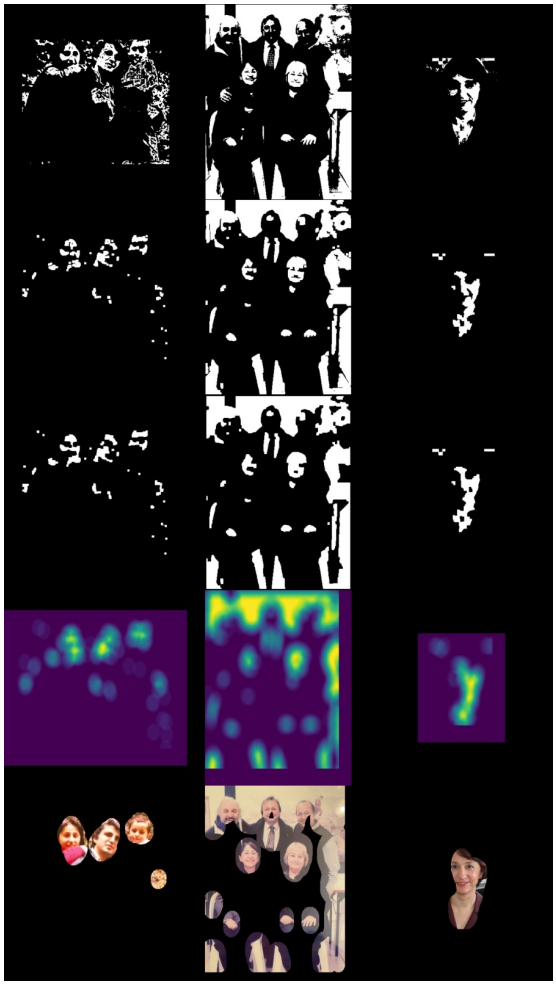

Fig. 2. Face detection progress for postprocessing stage

3) We used closing operation with a 5x5 element to fix regions that are broken up during the previous step. Shown in Row 3 of Figure 2.
4) We used convolution with a elliptical mask, mimicking the shape of he face to determine locations where a face may lie. Shown in Row 4 of Figure 2.
5) We used thresholding to the result of the convolution, including the parts of the image that passed the tresholding, concluding the algorithm. Shown in Row 5 of Figure 2.

D. Design Process

After the implementation of basic k-means algorithm, execution time of the algorithm limited the speed of our further progress by trying different processes. For that reason, we implemented the preprocessing steps 3 and 4.

Then we looked at commonalities between regions that make up the face, which brought us the reasoning we used in the k-means steps 2 and 3.

After that, we experimented with histogram equalization to differentiate colors easier.

When we arrived at the final result of k-means, given in Row 4 of Figure 1, we determined the noise created by similarly colored regions to be a problem. To solve it, we implemented opening and closing mentioned in the postprocessing steps 2 and 3. For the noise not eliminated by these steps, around the edge of the image for 1_source.png, we implemented the postprocessing step 1.

Morphological operations didn't eliminated all non-face regions. To distinguish regions that are face shaped from others, we finally implemented the postprocessing steps 4 and 5.

E. Results

Fig. 3. Original images and the end products

When we looked at the output of the algorithm, given in Figure 3, we can clearly see that it is quite accurate at finding locations that contains skin colors, while as seen in 2_faces.png, when the color space is limited, like the yellow saturation of the image, while it eliminates many non-face areas, color and even the shape information is not sufficient to perfectly detect only faces. But when dealing with more non-extreme example, such as 1_source.png and 3_source.png, false positives are very limited and all faces are detected. We can thank the postprocessing stage for eliminating false positives while keeping the faces, as a less violent method would create even more false positives and a more violent method would not guarantee the detection of all faces.

II. Pseudo-coloring

A. Coloring Grayscale Images

In a grayscale image we have 256 discrete values for every pixel that represent how much black or how much white the pixel is. 0 being black pixel and 256 being white pixel. In a multichannel image we have values ranging from 0 to 256 for every channel which are red, green and blue for rgb images. Now what we need is a function or set that maps gray values in given grayscale images to rgb values in the given source images. We know that our map should be one to one and need to take values between 0 and 256 and should give a 3 number ranging between 0 to 256 that represents the red green and blue values. For this we had couple of steps:

1) Creating a palette
2) Processing the palette
3) Mapping the grayscale image

To create a map for grayscale values to rgb values we need to first find corresponding gray values for each rgb value trio. We can use different equations for this, we can take the average of the channels, we can take minimum or maximum of the channels, we can take weighted averages etc. The goal is to create a single value between 0 and 256 from three values that are rgb. We thought logical way for this would be using the luminance. Normally when we look at an image we will probably say that the darker colors must be represented by darker gray values and lighter one by the lighter gray values but since the human visual system does not correspond to the rgb color system we can not just say bigger rgb values would be lighter. Closest to human visual system would be using the luminance, to compute luminance we can take the weighted average of the rgb values with the weights (0.2989, 0.5870, 0.1140). This values are used in many systems for computing the luminance.

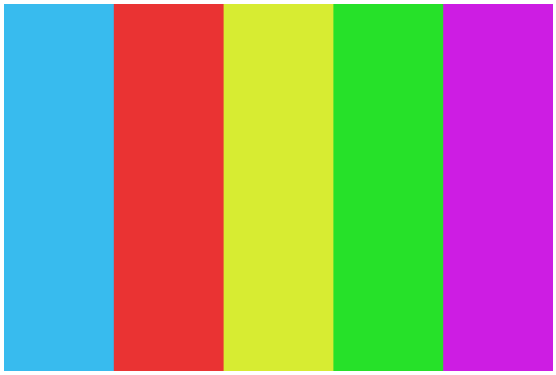

Fig. 4. Source image to use rgb values from

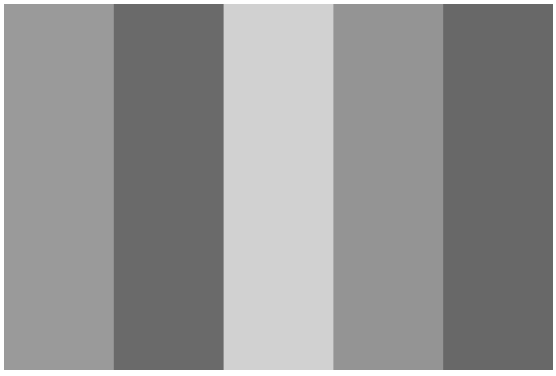

Fig. 5. We can guess that the lightest value would represent the color yellow in the picture

B. Extracting Palette from RGB Image

To extract a palette we created an empty palette with 0 value then we computed the luminance values from red green and blue values of source image and used them as indexes. Luminance values would represent the index of the palette and the palette would hold the rgb value the index created from like, palette[luminance] = (red, green, blue). This gave us a rather empty looking palette.

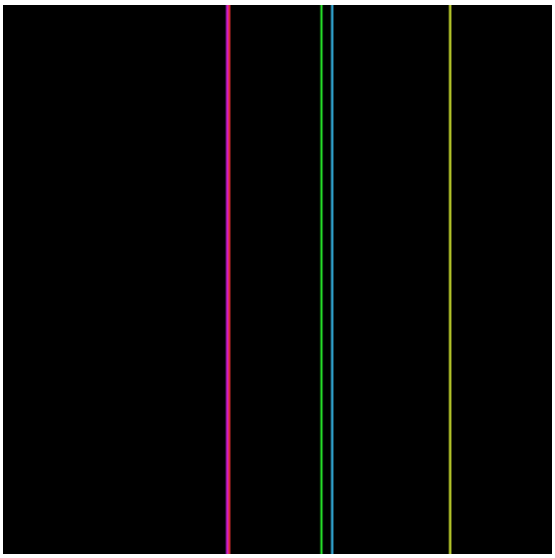

Fig. 6. Unprocessed palette

C. Processing the Palette

Now the gray values from the image may not correspond to the non-zero luminance values from the palette for this we thought of filling zeros of the palette with the weighted average of the two closest non-zero values of it using spatial distances. The result was continuous looking palette with every grayscale value corresponding to a rgb value where black values were still black and white values were still white.

Fig. 7. Filled palette

D. Using Palette to Color Grayscale Images

Now the easy part was using the grayscale values as indexes of palette and mapping those rgb values to the image for color-

ing like, colored_image = palette[gray_scale_images_values]. Results was really good for some images and good enough for some others.

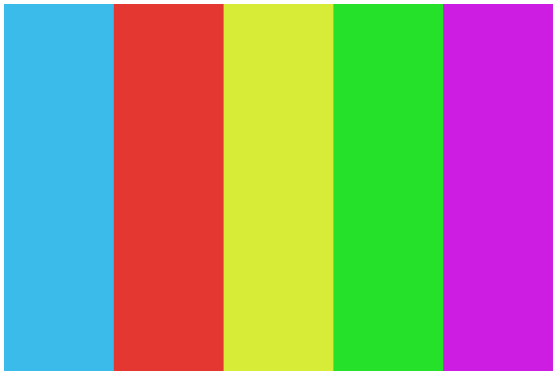

Fig. 8. Colored version of image 4

Fig. 10. Source image for image 3

Fig. 9. Grayscale image 3

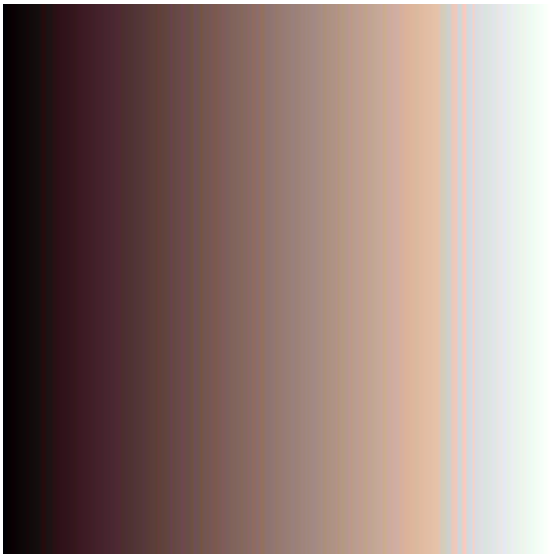

Fig. 11. Processed palette from source 3

Fig. 12. Colored version of image 3

We can see that our coloring algorithm does a bit of averaging but since these are a bit washed out old photos we are colorizing this was an expected result for us.

E. Rgb, Hsi and Histograms

We have different color systems for different purposes and usage areas, two of them we used in this homework are hsi and rgb. Hsi color space has three components named hue, saturation and intensity. These can be summarized as hue is which color are we using, saturation is degree of how much gray the color contains (sharper or washed out colors) and intensity is about brightness. We can convert values from these color systems to each other with 3 particular equations. Since these are just representations and we can convert from one of them to the other we do not lose information but they tend to give us advantages over the others in some application areas. For example the hsi color space is modeled after human visual system and easier to associate with how we see the colors.

We can extract the histograms from a hsi image just as same as rgb one. We have different curves plotted in same histogram for red, green and blue channel in one and same for hue, saturation and intensity in the others.

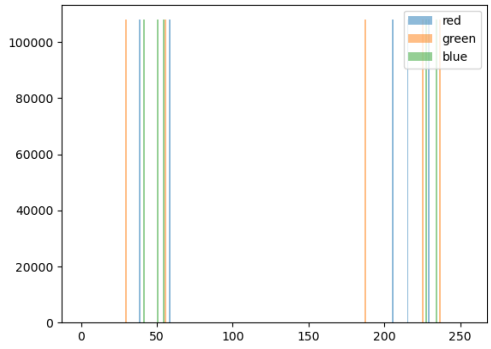

Fig. 13. Rgb histogram of colored image 4

We can see that there are a lot of 0 values in many of the buckets in histogram, this is due to the low number of color variation in the image 4 (5 different colors i particular). Same thing goes for the hsi histogram.

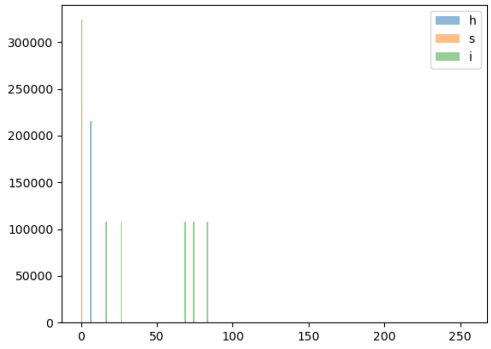

Fig. 14. Hsi histogram of colored image 4

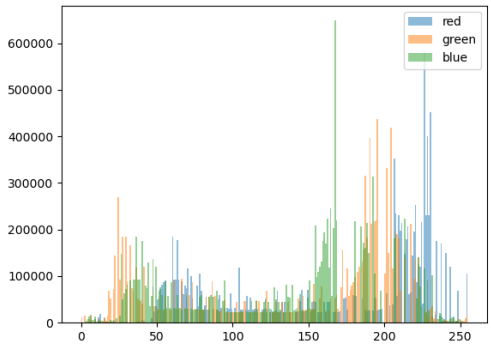

Fig. 15. Rgb histogram of colored image 3

We can see that the intensity curve is more ranged than the hue and saturation. This is because intensity channel acts as almost a mapping to a grayscale image and contains most of the information in the image.

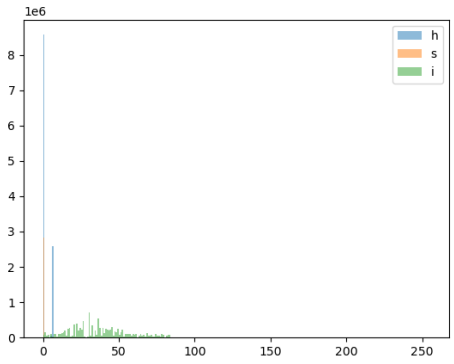

Fig. 16. Hsi histogram of colored image 3

F. Edge Detection in RGB and HSI

In this homework we were tasked with detecting edges from the images using gradient filters. For this purpose we used Sobel filters. We have two Sobel filters for horizontal edges and vertical edges named $Sobel_h$ and $Sobel_v$. Convoluting these filters gives us two different edges maps for horizontal and vertical edges. And combining these two will give us all the edges since every edge is made of vertical and horizontal components. We can do this in both rgb and hsi color space.

$$Sobel_h: \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \quad Sobel_v: \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

To do this we can find edges for red, blue and green channels and add them all to get the complete edge map.

Fig. 17. Edge map of image 1 in rgb color space

Fig. 18. Edge map of image 3 in rgb color space

To do this edge detecting method in hsi color space we almost only need the intensity channel. As we said the intensity channel of the hsi color space contains most of the information in the image. We tried using only the intensity channel for the edge detection and got results identical to the human eye.

Fig. 19. Edge map of image 1 in hsi color space

Fig. 20. Edge map of image 3 in hsi color space

III. Dependencies

We used following libraries for the described reasons.

- **os:** Handling non-existent input or output paths.
- **math:** Performing square root operation.
- **numpy:** Executing array and matrix operations.
- **PIL:** Reading images and converting them to arrays.
- **mathplotlib:** Creating histograms as graphics and writing arrays as image files.