Segmentation by Clustering

For Homework 2: Segmentation by Clustering, I wrote my K means image processing solution in Python 3.8, with the help of the OpenCV, NumPy, and Matplotlib libraries. My inputs consist of a picture of my hand taken against a dark background, and a user-defined K value for K means. My final output is a two-dimensional array whose height and width match my input image. The output array comprises a probability map with each point showing the normalized probability for that pixel to represent a skin tone, i.e., the probability that any given pixel is part of my hand.

At the start of the program, a window displays a preview of the image used. For reference, the original image I used can be found at the end of the report. While selecting a photo with ideal conditions, I found that a smooth dark matte background worked best. If the image were textured, bright, or shiny, lower K means values (K<4) would often associate artifacts from the background with skin tones. At higher K values, the background would contain several clusters which makes the comparisons we do later more expensive. I also found that I needed to decrease the size of the image as my cellphone camera takes images that were larger than 5 MB, which took too long to process. Once the image is loaded, the program asks the user for a K value as input. I recommend 6>K>2 but allow the user to choose any value. I also added a note that darker-skinned individuals may want to select a higher K value. As with all image processing, it is important to consider the inclusion of a diversity of users.

Will all inputs ready, the program runs K means to convergence one time. The centers are initialized to random values and I added a constraint to stop if 100 cycles have been run and K means still is not finished. This constraint was simply added to save time for demonstration purposes in case anything got stuck running too long. Once one K means run is done, the program displays some sample data, as seen in the image below in the red box. During this demo run, K was set to 3. The sample data shows the 3 RGB clusters that were identified. Next, pixel (0,0) is set as the background image for consistency. This pixel must contain an example of the background color, which is a constraint on what image should be allowed for the program. Next, a three-dimensional array with size picture pixel width x picture pixel eight x total K means runs is used to store the values from the initial run. A simplified comparison is used for each pixel to evaluate whether its red value matches the red value of the background. This simplified comparison is sufficient so long as the background is not bright, which was again a constraint we were allowed based on an input image with ideal conditions. If the pixel matches the background, the array’s default value of 0 is maintained. As the instructor mentioned in class, I grouped K clusters that are not the background; thus, if the pixel is any non-background color, a 1 is assigned to the 3D array. This is because the ideal K value is 2, one for background and one for skin; however, K = 2 is not accurate enough. These values are stored in [X][Y][0] as it the first fun. After the results are stored, a few samples are given for various pixels, for demonstration purposes.

After the initial run, K means is run 99 more times to convergence. This section can be seen in the image below, in the yellow box. While I could have run all 100 times right away, I believe the demonstration from an initial run was useful; however, choosing not to separate that initial sample run would not change any output. As with the initial run, the same comparisons are stored in the 3D array. Each run uses random initialization centers and rechecks pixel (0,0) to redefine the background in case the background was assigned a different RGB value. All data is stored in [X][Y][1-99]. Finally, a probability map is generated by taking each pixel, adding up all of the 1s in [X][Y][0-99], and dividing them by the total number of runs, 100. The result is a probability map in the form of a 2D array containing the individual probabilities for each pixel being skin. Again, some samples are given for demonstration purposes.

Text

Description automatically generated

Of particular interest is pixel (325, 774) in this demo run. The probably is between 0 and 1 because, with random initialization centers on each run, 1 run labeled that pixel as background, while 99 runs labeled it as skin. Figure 1 below is displayed to the user as an example of the output of 1 complete 1 K means run. Notice that (325, 774) in the bottom right of my hand where K means had some trouble identifying hand pixels due to feint shadows on the input image.

Graphical user interface, application

Description automatically generated

Finally, the program offers a user interface to interact with the probability map output. For the demo shown below, I chose another pixel from the shadowy bottom right of my hand to once again show that the random initializations have caused some K means runs to identify this pixel as skin, while another run identified it as background.

Text

Description automatically generated

Homework 2: Segmentation by Clustering taught me many things about ideal input conditions, K means, and image processing in general, as well as the specific libraries I used. If I were to extend this project, I would add features to map other hand image poses to my trained image’s pose and pixel counts. I could then start to build a system to identify the probability that any given hand was my hand.

(Original Input Image)

