HW4 Report

Problem Description

In this homework, we will be detecting the skin color pixels in hand images, using the histogram based technique. This technique starts with training images, which include samples of skin color pixels, after which we select a color space, and construct a 2D histogram based on the extracted pixels. Normalization is an important next step to be able to use a common threshold for all images in the test set. After that, we just run the a testing code on the testing images, where we lookup the normalized values of the histogram and check if they are above or below the threshold to classify as skin color pixels or not

Proposed Algorithm

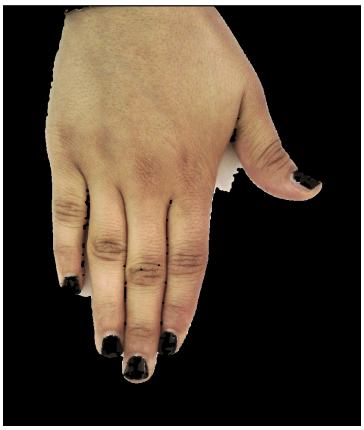
Following the steps specified above, the implementation was done as follow:

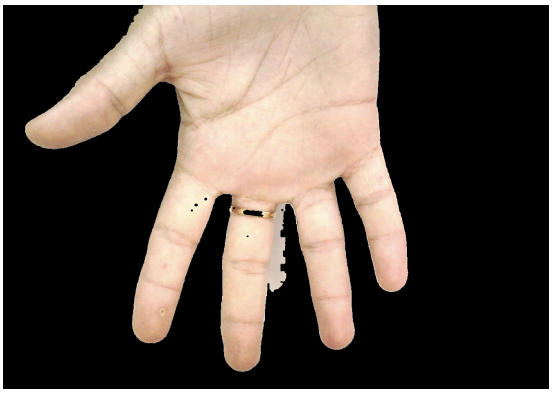
- For training, I selected some images from the hand pictures dataset on kaggle, and cropped areas that include skin only. Then, I looped over the training data and added all the pixel RGB values to the skin_pixels.csv, which will be then created to fit the histogram. This was done in get_pixels.py.
- I experimented with different color spaces, and I found that HSV does better than the rest. This is mainly because the skin tones are best captured by the hue, and saturation does a good job in restricting the valid ranges.
- The training part for the histogram was quite straightforward. I transformed the RGB values from the first step to HSV values, and took the H and S values for the extracted pixels and fitted them into a 2D histogram with K bins. I experimented with K, as seen in the histogram folder of the solution folder, and found out that the best outputs were for K=16. This is done in train.py.
- For the test images, they were first converted to HSV, normalized to get values between 0 and 1 similar to the saved histogram values. The H, S values for every pixel were queried from the histogram, and if above a certain threshold, we would classify the pixel as skin-colored. I experimented with the threshold values, and found the best value to be 0.01. This intuitively makes sense as the training sample is big and includes lots of values.
- After generating the mask for the detected pixels, I added closing to fill the holes that might have been wrongly classified, and opening after that to remove the additional noise. The last 2 points were done in test.py.

Results

Some of the results can be seen below for bins = 16 and threshold = 0.01, starting with the given images (those image were not included in the training data, so they are unseen):

We can see other results for other images like the following:





The original images can also be seen below:

Gaussian Method (Optional)

I also implemented the optional part, which given the same list of training pixels, fit a normal distribution, which means we extract the mean of H and S, as well as the standard deviation. After that, we loop over the pixels of a test image, if the pixel is within N standard deviation of the mean, we consider this pixel in our mask, if not, we omit it. This is done in gaussian_train_test.py. Some of the results can be seen below: