**Facial Recognition using Template Matching and Skin Color (Paper Implementation)**

[Allison Quinlan](mailto:anq2@illinois.edu)(anq2)

**Motivation and Impact**

Face detection has increasingly become a useful feature in everything from law enforcement to social media filters. Whether or not this is a good thing is up for debate, but the fact is that there is more demand for it. As it is used more and more, there is more and more room for abusing it. When these systems aren’t implemented correctly they may have immense negative impact, for example being poor detectors of various skin tones or more likely to mark a racial group as suspicious. Understanding how these systems are implemented is essential to making sure they are implemented and tested without malice or bias, as well as improving them as demand grows.

I also have a personal interest. There are many ways of implementing facial recognition. For example, I have quite a bit of experience with machine learning and have familiarity with using neural nets to find faces. I want to take an in-depth look at it from another perspective to see how they differ in accuracy and complexity.

**Approach**

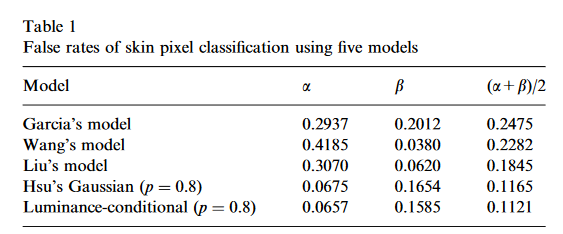
The paper uses two features for facial detection: skin detection and eye detection. I approached this problem by separately implementing both of these in the same way as the paper and then combining when they were both fairly accurate. Since both of these features-finding methods are intricate but self-contained, keeping them independent made debugging much easier. However, it did have the drawback that I needed to use multiple test sets. The eye detector works on the assumption that the input image is a cropped face, while the goal of the skin detector is to take any image and find the general face area, so I needed different inputs to debug each.

The skin detector’s main purpose is to decide where skin is and which skin is the face. The former is the trickier problem, since there are a variety of skin colors in the world and a variety of luminance conditions affecting the perceived color. The paper approaches this problem in the YCrCb color space. They look at each pixel’s luminance and using that to determine likely bounds for the other two channels. If a pixel’s Cr and Cb values are within the bounds for its luminance, it is marked as skin. Once the skin is marked, we eliminate the skin pixels that are either noise or not from the face. The paper is vague on how they implemented this, but I was able to accomplish the same goal by finding the contours of the skin mask and building a bounding box around the one that met certain criteria.

The eye detector does three basic tasks: find likely eyes using blob detection, normalize each candidate, and template match to choose the most likely location of the eyes. The blob detection works because a properly thresholded image of a face will have a dark spot at each eye surrounded by a light face area. With the paper’s guidance, I immediately eliminated some blobs pairs as eye candidates due to relative location. Normalizing each candidate is simply applying a rotation so that the eyes (and thus the face) are aligned with the camera. Once normalized, basic template matching calculates the difference between each rotated image and an average face template. As per the paper, I used SSD to get this difference and chose the highest score to be the correct location of the eyes.

At the end of this process, the location of the eyes are known for an image that has gone through much cropping, resizing, and rotating. Working backwards, the corresponding points are located on the original image and labeled.

**Results**

The paper measured their accuracy by looking at the false negative and false positive rates. As you can see in Table 1, the false negative rate was from about 6% - 40% and the false positive rate was about 3% - 20%. My goal was to achieve similar error rates. I ran my code over an image set which had only faces, annotated with the pixel locations bounding the eyes, and a different set of animals and objects but not people. After running the detector random subsets of the data set, I had an average false negative rate of 35.4% and average false positive rate of 2.76%. This is within the range of results from the paper, so I consider my code a successful implementation of their methods. I also ran a good number of tests by hand both in debugging and afterwards to make sure the results look good throughout the entire detecting process. An example of this can be seen below in the Result Example section.

**Implementation details**

The code is all written in Python3 Jupyter Notebooks. It doesn’t use much beyond standard libraries and cv2. I wrote most of the code from scratch, though some of the more boilerplate code is adapted from articles or StackOverflow posts [[1]](https://learnopencv.com/blob-detection-using-opencv-python-c/)[[2]](https://stackoverflow.com/questions/64345584/how-to-properly-use-cv2-findcontours-on-opencv-version-4-4-0). The relevant code is also labeled in the Jupyter Notebook with the source. I also used a total of three datasets for creating and testing my code. One dataset from Kaggle had the cropped and annotated face images I needed for initially testing the eye detector as well as creating an average face template [[3]](https://www.kaggle.com/drgilermo/face-images-with-marked-landmark-points/version/1?select=facial_keypoints.csv). Another data set from UTKFace had images of people, both faces and full bodies, that I used to debug skin detection and do end-to-end testing [[4]](https://susanqq.github.io/UTKFace). Lastly, I used the CIFAR-10 image set to detect false positives on images that did not contain human faces [[5]](https://www.cs.toronto.edu/~kriz/cifar.htm).

**Challenges**

Though my implementation is not particularly innovative, I do believe that it is sufficiently challenging for at least 10 points for this component. The largest challenges I faced were due vagueness from the paper, leading me to do outside research or brainstorming on how to implement them myself.

As mentioned above, skin detection was based on bounds which were decided independently for 5 different luminance ranges. The paper provides a very nice and in depth discussion on how they found gaussian distributions for each channel at each luminance range. However, they failed to provide any numbers to implement these distributions. Since they were found by analyzing only the skin pixels of many images and I couldn’t find any available image sets where the skin was annotated, it was infeasible for me to exactly replicate their distributions. Instead, I used the cropped image set in the hopes that most of each picture was taken up by skin. I made histograms of the Cr and Cb values within each luminance range and found that each had a number of peaks. I assumed that one or a few of these correspond to skin color, and experimentally tweaked the Cr and Cb bounds until the resulting skin masks generally looked good. This method isn’t as rigorous as the one detailed in the paper, but it does fairly well and follows the same conceptually ideas as the original.

The other major challenge was removing the excess skin to find the face region. The paper is vague on how to accomplish this, saying only to, “utilize the binary morphological operations to remove the false skin pixels, to extract skin regions, and to obtain the corresponding skin-region rectangles.” After experimenting, I realized that the face tended to be one of the biggest skin sections. Other body parts are often cut off by clothes or objects, but we like to get a picture that has the whole face. I also noticed that the face is typically the only skin region with holes in it, since the eyes are not labeled as skin. Using these two facts and the opencv contouring library, I was able to create a bounding box around the most likely candidate for the face, and achieve what the paper described.

**Result Example**

| **1)** Original Image |  | **5)** Template Matching Inputs - For each blob, the image is rotated and cropped (left) to align eye candidate with the eyes on the template (right) |
| --- | --- | --- |
| **2)** Skin Mask - Green indicates non skin region |  |
| **3)** Cropped Face Image - This is the region of image where we will search for eyes to confirm it contains |  | **6)** Result- The best candidate from template matching is chosen and annotated on the cropped image. Working backwards we can place the eye points back on the original image |
| **4)** Blob Detections - Blue circles indicate dark spots that may be eyes |  |