Jeremy Midvidy, jam658
EECS 332, Fall 2018
MP4 Report

About My Code:

My code for MP3 can be found in this folder, in the python file, `MP4.py.` The file is callable from the command line with the command `$ MP4.py PATH_TO_IMAGE`.

After running, it will write a result image named "`PATH_TO_IMAGE_result` " of the classified images to the folder named "results" as well as print the histogram used for classification. Additionally, if called from the command line without an input argument, it will run tests on all of the images in the "testimages" directory and write each result image and histogram to the "results" folder.

I wrote my code in Python 3.7, and it assumes python packages `numpy`, `sys`, `matplotlib`, `os`, `math`, and `sys` are installed.

Results:

I ran my code on the given image files, and in short, my code performed pretty well. The code executes this logic:

(1) First, my code trains a histogram that is later used for classification. To train, my code iterates through all of the training images of skin found in the "skin images" directory in this submission folder. For each image, my code reads the image, converts it from RGB to HSV, and then scans the 2-D matrix and, for each occurence of a [h,s], increments the count in that bin in the histogram by 1. Likewise, if it encounters a [h,s] not yet in the histogram, the code will initialize [h,s] = 1 in the histogram dictionary.
   (a) I use a dictionary of dictionaries to represent the histogram. I.e. hist = { { h1 : {s1 : val1, s2: val2, s3:val3}, h2 : {s1 : val1, s2: val2, s3:val3} etc...}}. This allows for O(1) lookup speeds because Python implements dictionaries as hash tables. This greatly increases the speed of my code.

(2) Then, I quantize and normalize the histogram much like I did in MP3.

(3) Then, I read the input image, converting it from RGB to HSV. Then, I scan the input image and for each pixel:
   (a) See if the pixel is in the trained histogram:
      (i) If it is and that pixel's value is above the threshold (which I estimated to be 20), then I leave that pixel alone as it is presumed to be a skin pixel.
      (ii) If it is in the histogram and below the threshold, I set that pixel to black ([0,0,0]).
   (b) Likewise, if that pixel is not in the trained histogram, I set it to black.

(4) I then compare the processed input matrix to the original input matrix. For each cell, if the cell has been changed to black, I change it to black in the original input matrix. If it has not been changed to black, then it should be classified as skin, so I leave it alone in the original input matrix.
   (a) I do this because the original matrix before hsv conversion had more accurate pixel information, so using the original matrix rather than the processed matrix will lead to greater resolution output because it has not lost information where needed.

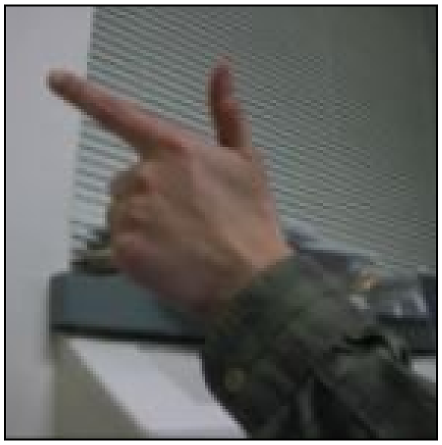
(5) Then I write this labeled matrix to an image and store it in the results folder.

Ultimately, I think my program will work well with a large set to train on. It's clear that, just from my own collecting of samples, that the training set is lacking some skin tones for specific shades, shadows, as well as parts of a face like lips or nostrils. But I think the logic of my code is correct, and that to improve, it will just need a bigger and more robust training set.

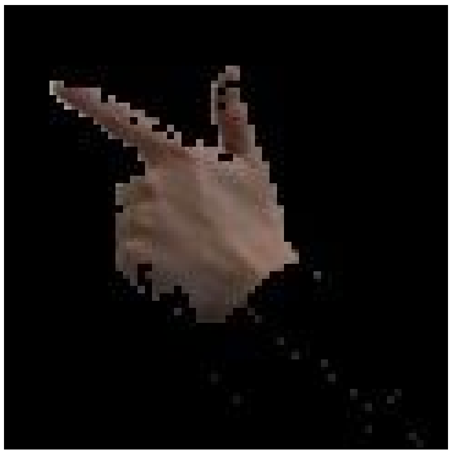
Histogram learned from inputs (with smoothing and normalization):

```
0.0 : {0.0: 0.41804116187417895}
1.0 : {0.0: 0.4550552230817886}
2.0 : {0.0: 1.4087116236072592}
3.0 : {0.0: 2.7325086362088262}
4.0 : {0.0: 3.4183574174086506}
5.0 : {0.0: 5.687101639663309}
6.0 : {0.0: 7.254755996691481}
7.0 : {0.0: 11.239210820804749, 1.0: 11.247920011677126}
8.0 : {0.0: 25.472206003989683, 1.0: 25.489624385734437}
9.0 : {0.0: 32.89461392497445, 1.0: 32.912032306719205}
10.0 : {0.0: 57.759353865615715, 1.0: 57.79419062910522}
11.0 : {0.0: 67.53542061986083, 1.0: 67.5920303605313}
12.0 : {0.0: 93.23624288425046, 1.0: 93.5846105191456}
13.0 : {0.0: 131.8157811511701, 1.0: 132.12060283170337}
14.0 : {0.0: 143.4142460954605, 1.0: 143.7343088600204}
15.0 : {0.0: 175.49019607843135, 1.0: 176.71166009828247}
16.0 : {0.0: 197.78572471172092, 1.0: 198.31263075949985}
17.0 : {0.0: 231.27038631829905, 1.0: 232.29153894808547}
18.0 : {0.0: 254.42159295479982, 1.0: 256.4290614508831}
19.0 : {0.0: 278.6244343891403, 1.0: 279.29068749087725}
20.0 : {0.0: 302.21763246241426, 1.0: 303.08419695421594}
21.0 : {0.0: 312.16135114095266, 1.0: 312.5489101347735}
22.0 : {0.0: 322.90413808203186, 1.0: 323.23291003746414}
23.0 : {0.0: 336.65377317179974, 1.0: 336.8301342869654}
24.0 : {0.0: 342.34522940689925, 1.0: 342.42143482703256}
25.0 : {0.0: 351.51600739551407, 1.0: 351.5334257772588}
26.0 : {0.0: 355.00403833990174}
27.0 : {0.0: 356.80466355276604}
28.0 : {0.0: 357.3968885320878}
29.0 : {0.0: 357.71259670121157}
30.0 : {0.0: 357.91726268671243}
31.0 : {0.0: 357.9346810684572}
33.0 : {0.0: 357.9433902593296}
358.0 : {0.0: 358.00000000000006}
```

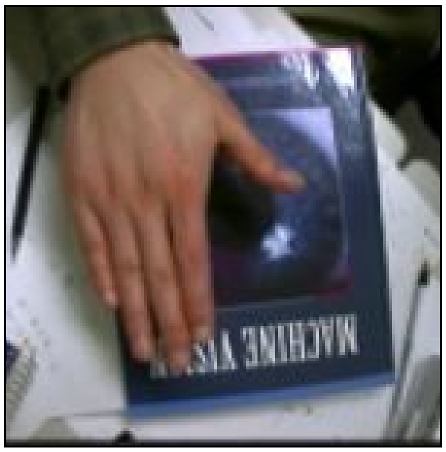
Results Analysis for gun1.bmp



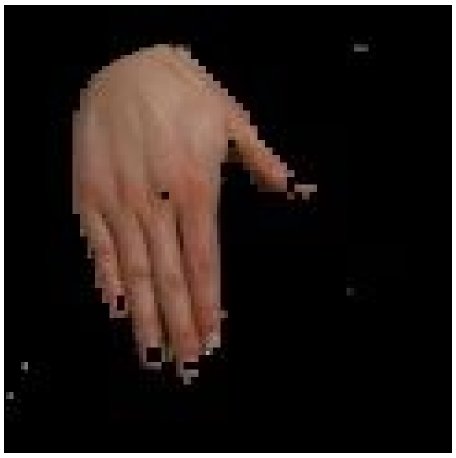
→



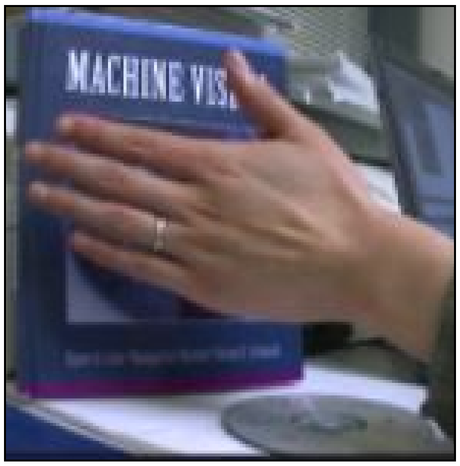
Results Analysis for joy1.bmp

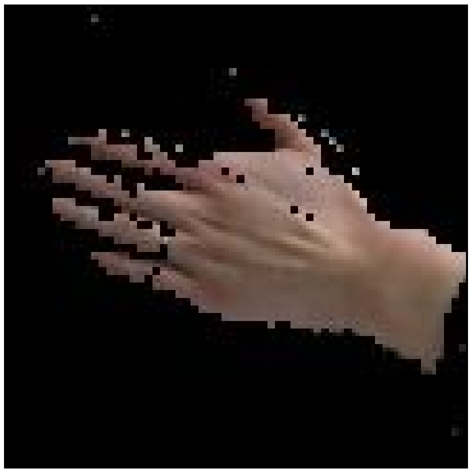


→



Results Analysis for pointer1.bmp





Results Analysis for face.bmp

→