Ryan Reynolds
Brett Dawson

Real Time Facial Recognition

Executive Summary:

In this project we trained and created a real time facial recognition program in python using LBPH in the OpenCV library.

Abstract:

Computer graphics is heavily intertwined with facial recognition. At a glance, it may seem the only time graphics come into play is to draw the box around the face being tracked. However, much more goes on behind the scenes. The heart of facial recognition is teaching a machine how to recognize object within images as a human would. To do this the computer has to trained to recognize features that make the objects distinct within each frame sent to the recognizer. Furthermore, what humans consider facial features must be converted in a way that the computer could understand. By finding a mathematical algorithm, usually geometric, way to represent the features of a human face we are letting the computer implement a guided version of its own computer graphic.

Algorithm:

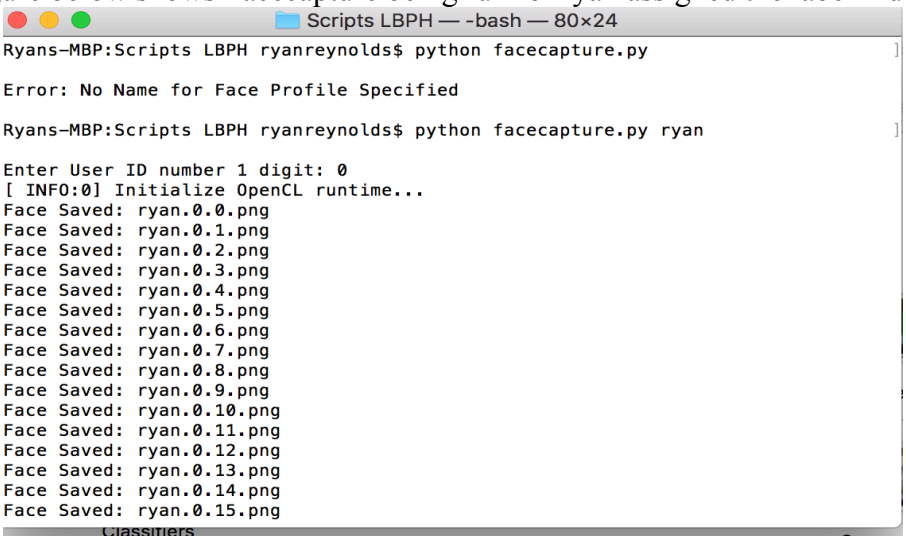
The proposed algorithm to use was the HOG or Haar Cascade algorithms for detection. We chose to use openCV's defined Haar Cascade classifier for frontal face detection. Haar Cascades detects facial features by looking at the sum of the pixel values, lumosity in the case of facial features, in a defined rectangle of space and an outter rectangle of space. This helps identify facial features such as the eye region being darker than cheek and nasal regions. The two algorithms we proposed to use was eigenfaces or local binary point histograms. After doing some research we settled on local binary patern histograms because it seemed to be more dynamic because it would work in instances of varying light conditions. LBPH divides the image into 3x3 matices and compares the central pixel's lumosity value to the central pixel if it is greter than or equal the value is replaced with a 1 less than it is a 0. The matrix value is converted from binary to decimal to represent the regions lumosity value. This is applied to the whole image. Each regional lumosity value is stored in a histogram. Each bin represents a range of regional lumosity for the image. The collection of all of the bins represents the image's feautures. When you insert only facial images into the algorithm it will be able to recognize different faces by comparing the histogram of the detected face to the histograms within recognizer classifier.

Code High level Functionality/Results:

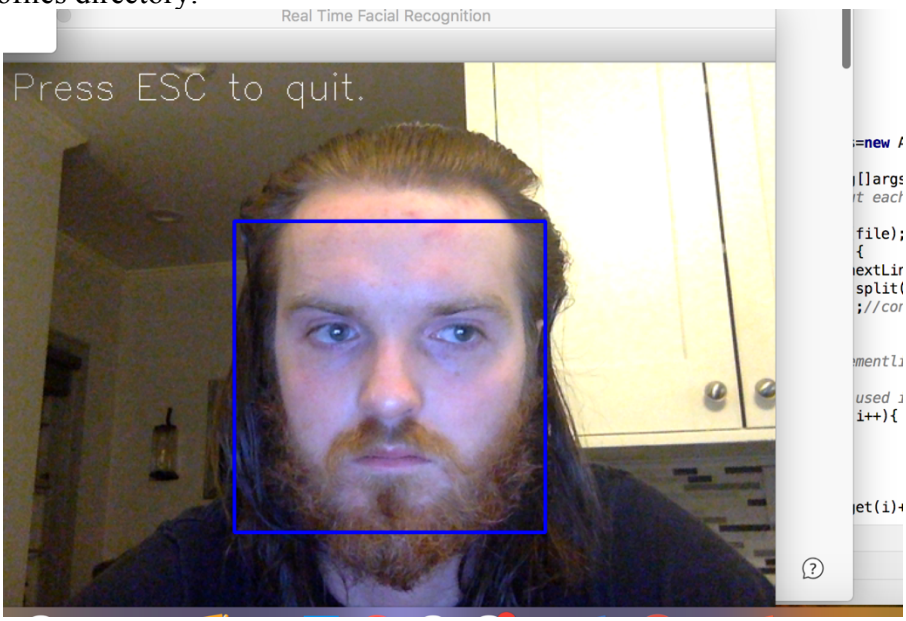
Three python programs were created facecapture.py, recognizer.py, and trainer.py. Screenshots of an example run of each program is run after each description.

The Facecapture script takes in the name of the person's face you would like to capture from the command line. The user is prompted for an integer identifier to be stored as the profile's label in the classifier. Facecapture then opens the computer's default video capture device and displays the image to the screen. If a face is detected within the frame a rectangle is drawn from the top of the forehead to the bottom of the chin. The program crops the image to this rectangle converts it to gray scale and writes it within the profiles directory in an identifiable format. Facecapture will continually capture the faces until the number of faces reaches the defined face count defined in the variable num_faces_to_collect.

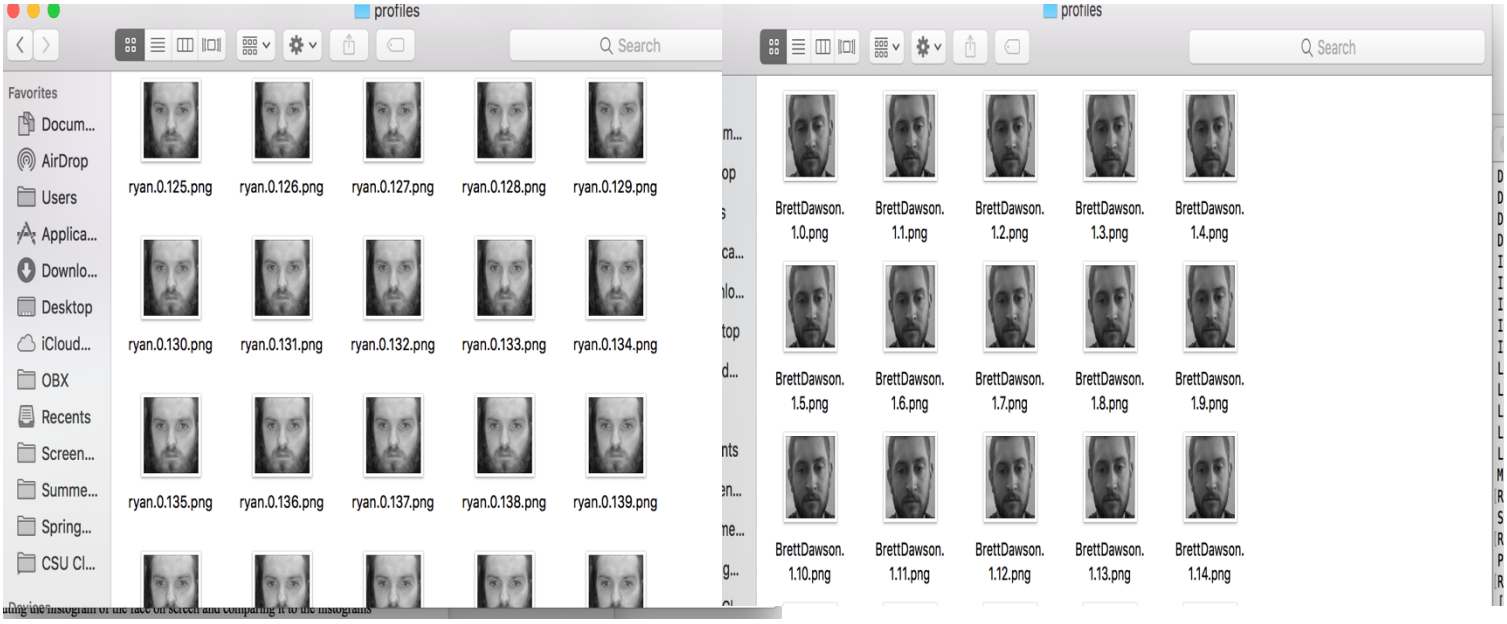
The figure below shows Facecapture being run for ryan assigned the label index 0.



The following picture shows the capturing and cropping of the images to be stored in our face profiles directory.

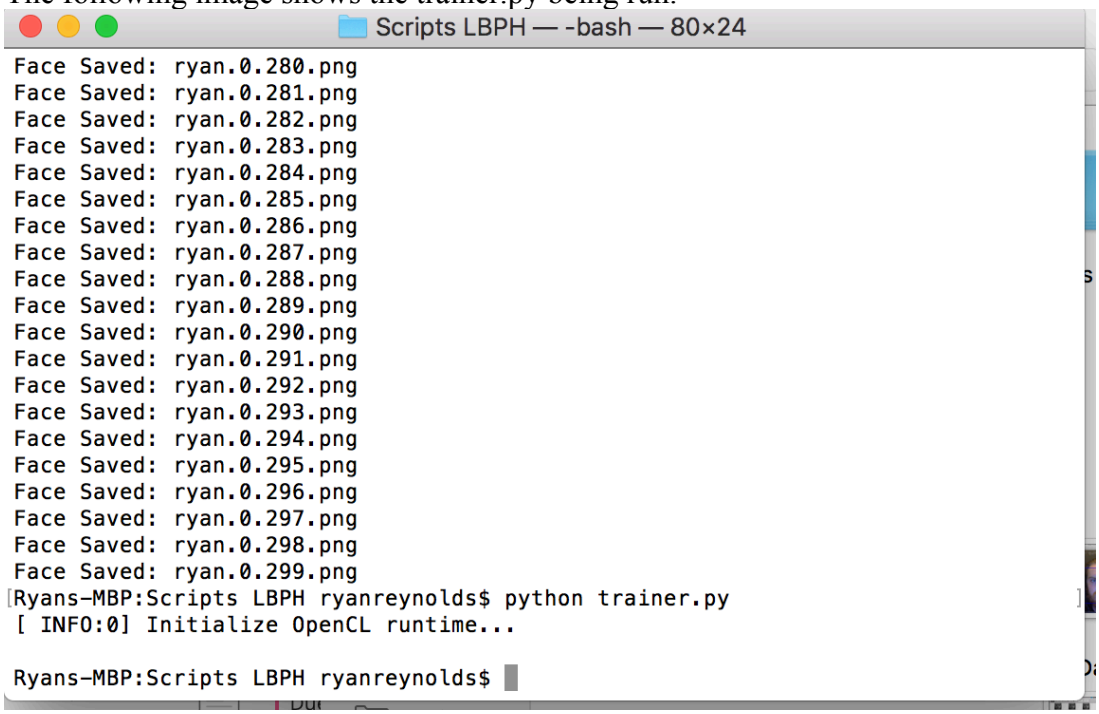


The following images show the format of each profile stored in the profiles directory.

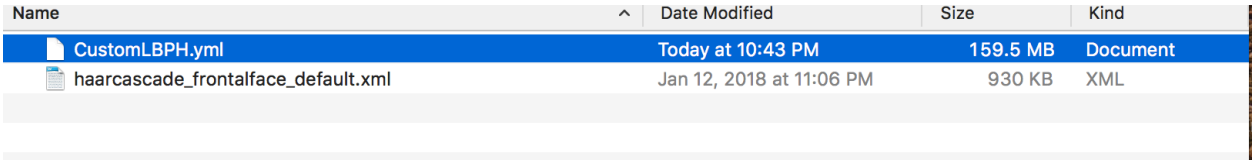


The trainer script creates an openCV face recognizer of type LBPH (local binary pattern histograms). The program opens the profiles folder and retrieves each cropped image from the profiles directory. Each image is read in as a standard data format using PIL (python imaging library). Then the images are converted to 8 bit unisigned integers and stored into a multidimensional array using the numpy library. Finally the array of images and array of labels are passed to the train method within the facerecognizer object. This creates the classifier that is used to detect all of the faces using LBPH within the profiles directory. Our custom face recognizer classifier is then written to the CustomLBPH.yml file so that it may be loaded by the recognizer class.

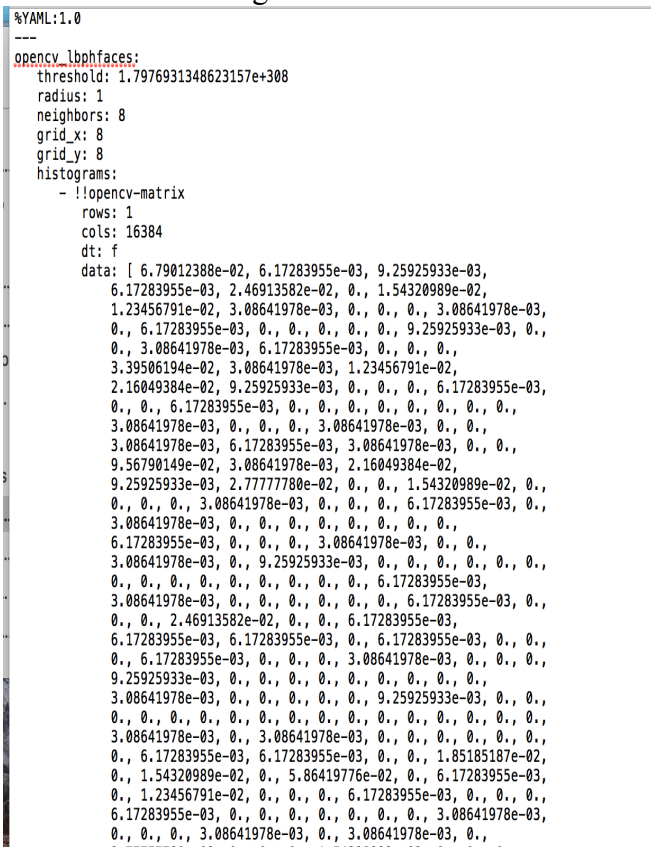
The following image shows the trainer.py being run.



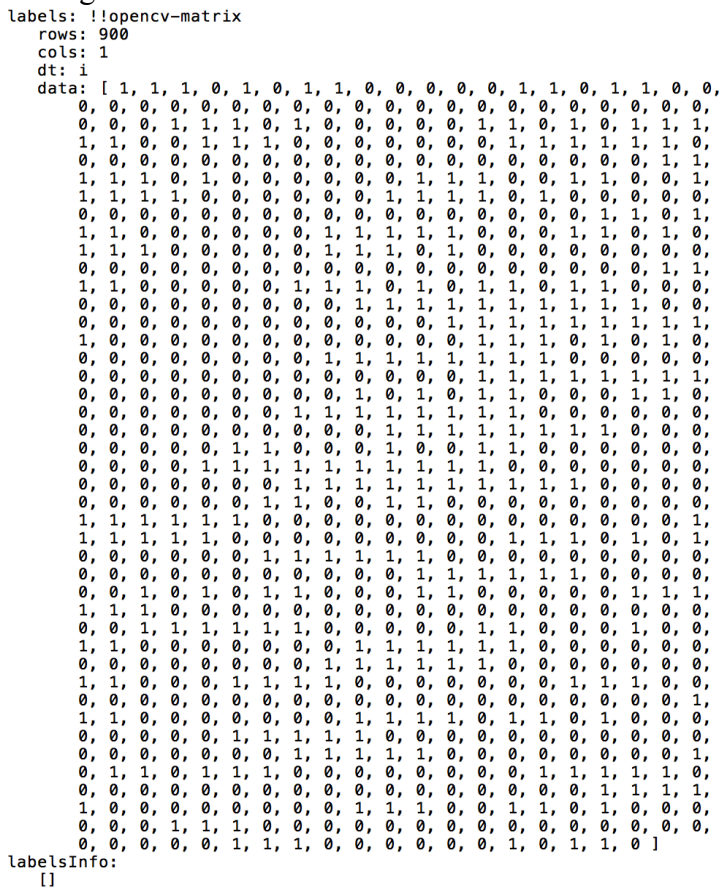
The following image shows the classifier created by the trainer.py program.



The following images show the contents of the custom classifier that contains the labels matrix and the histogram identifications matrix of the images loaded to the classifier.



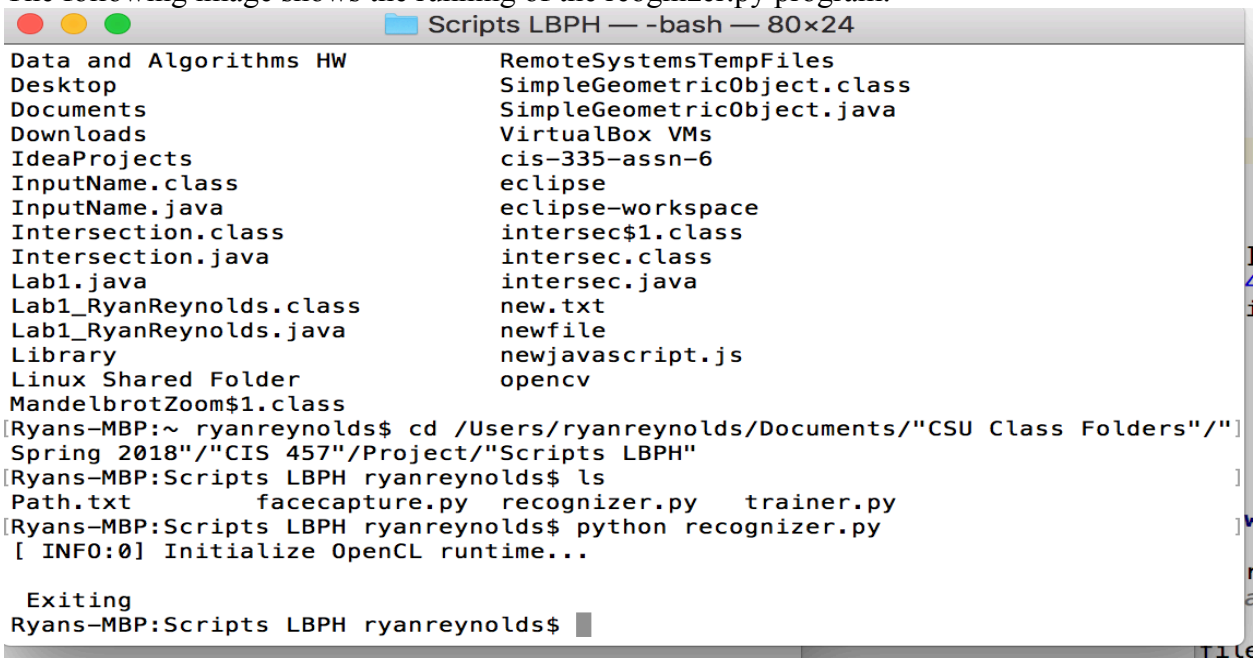
```
%YAML:1.0
---
opencv_lbphfaces:
   threshold: 1.7976931348623157e+308
   radius: 1
   neighbors: 8
   grid_x: 8
   grid_y: 8
   histograms:
      - !!opencv-matrix
         rows: 1
         cols: 16384
         dt: f
         data: [ 6.79012388e-02, 6.17283955e-03, 9.25925933e-03,
             6.17283955e-02, 2.46913582e-02, 0., 1.54320989e-02,
             1.23456791e-02, 3.08641978e-03, 0., 0., 0., 3.08641978e-03,
             0., 6.17283955e-03, 0., 0., 0., 0., 0., 9.25925933e-03, 0.,
             0., 3.08641978e-03, 6.17283955e-03, 0., 0., 0.,
             3.39506194e-02, 3.08641978e-03, 1.23456791e-02,
             2.16049384e-02, 9.25925933e-03, 0., 0., 0., 6.17283955e-03,
             0., 0., 6.17283955e-03, 0., 0., 0., 0., 0., 0., 0., 0.,
             3.08641978e-03, 0., 0., 0., 3.08641978e-03, 0., 0.,
             3.08641978e-03, 6.17283955e-03, 3.08641978e-03, 0., 0.,
             9.56790149e-02, 3.08641978e-03, 2.16049384e-02,
             9.25925933e-03, 2.77777780e-02, 0., 0., 1.54320989e-02, 0.,
             0., 0., 0., 3.08641978e-03, 0., 0., 0., 6.17283955e-03, 0.,
             3.08641978e-03, 0., 0., 0., 0., 0., 0., 0., 0.,
             6.17283955e-03, 0., 0., 0., 3.08641978e-03, 0., 0.,
             3.08641978e-03, 0., 9.25925933e-03, 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 6.17283955e-03,
             3.08641978e-03, 0., 0., 0., 0., 0., 0., 6.17283955e-03, 0.,
             0., 0., 2.46913582e-02, 0., 0., 6.17283955e-03,
             6.17283955e-03, 6.17283955e-03, 0., 6.17283955e-03, 0., 0.,
             0., 6.17283955e-03, 0., 0., 0., 3.08641978e-03, 0., 0., 0.,
             9.25925933e-03, 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             3.08641978e-03, 0., 0., 0., 0., 0., 9.25925933e-03, 0., 0.,
             0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             3.08641978e-03, 0., 3.08641978e-03, 0., 0., 0., 0., 0., 0.,
             0., 6.17283955e-03, 6.17283955e-03, 0., 0., 0., 1.85185187e-02,
             0., 1.54320989e-02, 0., 5.86419776e-02, 0., 6.17283955e-03,
             0., 1.23456791e-02, 0., 0., 0., 6.17283955e-03, 0., 0., 0.,
             6.17283955e-03, 0., 0., 0., 0., 0., 0., 0., 3.08641978e-03,
             0., 0., 0., 3.08641978e-03, 0., 3.08641978e-03, 0.,
```



```
labels: !!opencv-matrix
   rows: 900
   cols: 1
   dt: i
   data: [ 1, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0 ]
labelsInfo:
   []
```

The recognizer script opens the machine's default camera. It then calls the facecascade method and sets the images from the live feed to greyscale to better detect the changes in contrast. Then for all pixels in the face (x,y,w,h), the program calls the lbph classifier to determine in real time how confident it is that the face matches an ID. If the confidence is greater than a 45% match, the ID of the suspected person that was put into the trainer.py is displayed. If it is lower than 45%, the ID is displayed as "unknown". The confidence is determined by computing the histogram of the face on screen and comparing it to the histograms of the faces that were trained. The shortest path between the two histograms is represented by a confidence factor, and when we take the inverse of this it shows the confidence percentage on the screen for which user the program thinks it is. The user's name will be displayed above the box being drawn around the user's face. Pressing escape will exit the video at any time. When the recognizer.py is finished, it closes the camera and destroys all windows. The confidence of the image seemed to vary in our testing of the algorithms based on the difference in brightness in a room where the image is trained compared to where it is attempted to be recognized.

The following image shows the running of the reognizer.py program.



The following image shows the recognizer.py using our classifier to recognize both of us in realtime with a confidence rating of 52% and 63%.

The face is detected using openCV's cascade classifier definition loaded with the default haaar cascade classifier haarcascade_frontalface_default.xml. This classifier is called in all three programs to identify the faces on the image.

Improvements:

The confidence rating averages 60% and maxes out at 75% due to the quality of the pictures that we are passing to the classifier. A higher resolution image would increase the feauture detection thereby increasing the confidence at the cost of efficiency. Additionally the outside SVM library could be implemented with a better LBPH algorithm to create a better classifier. Another pitfall of the program is that it fails to update the profile names in the recognizer class. When you add a new face you have to manually add them to the correct index in the name array in the recognier program. System calls should be implemented to create a directory for the name entered upon executing facecapture.py. Then the trainer should read the label in the directory name then load each image contain in each profile's directory. Then the recognizer should populate the names label each time it runs by reading the directory names within the program. Additional improvements for the classifier would be to add rotations. This icludes both sideface rotation recognition and recognition as a head tilts toward each shoulder.

Conclusion:

Overall, we accomplished what we set out to do- create a real time facial recognition program using LBPH. However, we were disappointed with the accuracy of the program, only posting confidence levels in the 60% range. As stated previously the structure of the profiles directory is not where we would like it to be. The LBPH did perform better in different lighting scenarios. The training set data was taken in the CIS 457 classroom with good lighting. The result shown on the previous page shows the recognizer working on a dimly lit face and a brighter face on the phone screen.