Facial Recognition using Haar Classifiers and Local Binary Pattern Histograms

Team 16

Michael Cole

Computer Vision (EE 4780)

Xin Li

Spring 2017

Contents

Table of Figures

Overview

The goal of my final project was to achieve facial recognition. Initially, my plan was to have the program analyze a still image and determine who was in the image based on a Fisherface facial recognition algorithm. However, I came up with a more interesting project for facial recognition after the presentation.

In the evolved version of my final project, the goal was to implement real-time facial recognition by detecting and identifying faces present in the video frames as they are captured by a webcam. The user would start the program and after training it to recognize their face, if their face is in the frame, their face would be framed by a blue square and their name written in the square. I created three python modules to achieve this purpose: datasetCreator.py, trainer.py, and recognizer.pi. These modules are based on a tutorial [1] on thecodacus.com which utilize some of the built in functions in openCV. The training data acquisition and tuning of the Facial recognition parameters were done by me.

Algorithms

This project required me to utilize two computer vision algorithms: Haar cascades and Local Binary Pattern Histogram facial recognition.

Haar cascade classifiers are a method of object detection, which is to say it is a method of discovering whether there are objects meeting certain criteria within an image. I needed to use Haar cascades to detect the presence of a human face. A Haar-like feature considers neighboring rectangular regions at a specific location in a detection window, sums up the pixel intensities in each region and calculates the difference between these sums. This difference is then used to categorize subsections of an image. An example of this would be the detection of human faces. Commonly, the areas around the eyes are darker than the areas on the cheeks. One example of a Haar-like feature for face detection is therefore a set of two neighboring rectangular areas above the eye and cheek regions. A Haar cascade classifier then uses a series of learning stages to scan an image in a predetermined window size to check for the presence of a Haar-like feature [1]. The classifier must first be trained to recognize these features by analyzing many “positive” images which contain the object in question and “negative” images which do not. This is a complicated machine learning algorithm, but luckily, opencv has included a cascade classifier function that takes in an xml file with the positive and negative images and produces a trained Haar classifier.

The second algorithm I need to utilize was local binary pattern history facial recognition. The difference between facial recognition and face detection is that facial recognition requires us to associate a face with a specific person, instead of merely detecting that there is a generic human face present. There are three classic face recognition algorithms, Eigenfaces, Fisherfaces, and local binary pattern histogram. The first two algorithms find a mathematical

description of the dominant features present in a set of the same face while LBPH analyzes each face in the training set independently. According to my research [2], LBPH outperformed Eigen and Fisherfaces in a variety of environments and lighting conditions and was faster because of it's simplicity, so I chose to work with it. Luckily, the latest versions of OpenCV include all three of these face recognition functions, so I did not need to implement the algorithm by hand, but I didn't need to understand the basics of how they worked in order to properly implement them.

LBPH face recognizers characterize each image in the dataset locally; and when a new unknown image is provided, we perform the same analysis on it and compare the result to each of the images in the dataset. The way which we analyze the images is by characterizing the local patterns in each location in the image. An example of such local binary patterns that are produced is below:

Figure 1 LBPH face recognition local binary patterns [3]

To achieve the facial recognition, I would need a data set of faces to recognize. The datasetCreator.py module that I created allows the creation of a dataset consisting of 301 images of each face that the user would like to have recognized by detecting faces in a webcam frame and storing an image of the face in a database.

1. Create a Haar cascade classifier object called 'detector' that recognizes front facing human faces by calling cv2.CascadeClassifier() and using the frontalface_default xml file provided on opencv's Github as the training set.
2. Have the user input an ID number of the face in the video frame to be used later.
3. Utilize cv2.VideoCapture(0).read() to open the computer's default webcam.
4. Convert the video capture to grayscale
5. Using our detector object, call detector.detectMultiScale() to detect front facing human faces in the video frame.
6. Draw a square around faces detected.
7. Store the contents of the square in a file called dataset/User.ID.sampleNumber.jpg

8. Wait 100ms and capture/store another sample; do this 301 times.

```
import numpy as np
import cv2
#create haar cascade classifier
detector = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
#start webcam
cap = cv2.VideoCapture(0)
#ask user for the id of the face it wants to store in dataset
id = raw_input('enter user id: ')
samplenum = 0
while(True):
    #capture webcam image
    ret, img = cap.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    #detect faces in the image
    faces = detector.detectMultiScale(gray, 1.3, 5)
    #store the faces in the dataset and include the user id in the filepath
    if(len(faces)!=0):
        for (x,y,w,h) in faces:
            samplenum = samplenum + 1
            cv2.imwrite("dataSet/User."+str(id)+"."+str(samplenum)+".jpg",gray[y:y+h,x:x+w])
            #draw a rectangle around the detected face
            cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
            cv2.waitKey(100)
    cv2.imshow('face',img)
    cv2.waitKey(1)
    #we take 301 sample images
    if(samplenum > 300) :
        break
cap.release()
cv2.destroyAllWindows()

```

Figure 2 datasetCreator.py

As many distinct users as desired can be put into the dataset, but the only caveat is that only one face must be in the frame at a time for the dataset to be created correctly. Also, if more than 301 samples are desired, the program can simply be run again with the same user ID.

The second module for my project is called trainer.py. This module is used to train a Local Binary Pattern Histogram face recognizer using the dataset created in datasetCreator.py. it uses the PIL library for some functions. It works in the following way:

1. Create an object called “recognizer” by using cv2.createLBPHFaceRecognizer(2,10,10,10,50). The arguments are the radius used for building the Circular Local Binary Pattern, The number of sample points (neighbors) to build a Circular Local Binary Pattern from, The number of cells in the horizontal direction, The number of cells in the vertical direction, and the threshold applied in the prediction. I had to manipulate these values to get the best results from my webcam, so experience may vary based on hardware.
2. Create another Haar cascade classifier using the same xml file from the dataset creator.
3. Get the filepath of all the images in the dataset directory.

4. Create two empty lists to store face samples and Ids.
5. Using 'Image' from the PIL library, open each image in the dataset and convert it to grayscale
6. Convert the grayscale images to numpy arrays
7. Determine what the ID number of each image is
8. Use the detector object to again scan the image and detect faces therein, just as in the dataset program.
9. Store each detected face and the associated ID in the arrays we created earlier.
10. Use 'recognizer.train()' to train the LBPHFaceRecognizer.
11. Save the trained recognizer to a yml file called recognizer/trainer.yml.

```
import cv2,os
import numpy as np
from PIL import Image

recognizer = cv2.createLBPHFaceRecognizer(2,10,10,10,50)
detector= cv2.CascadeClassifier("haarcascade_frontalface_default.xml");

def getImagesAndLabels(path):
    #get the path of all the files in the folder
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    #create empth face list
    faceSamples=[]
    #create empty ID list
    Ids=[]
    #now looping through all the image paths and loading the Ids and the images
    for imagePath in imagePaths:
        #loading the image and converting it to gray scale
        pilImage=Image.open(imagePath).convert('L')
        #Now we are converting the PIL image into numpy array
        imageNp=np.array(pilImage,'uint8')
        #getting the Id from the image
        Id=int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces=detector.detectMultiScale(imageNp)
        #If a face is there then append that in the list as well as Id of it
        for (x,y,w,h) in faces:
            faceSamples.append(imageNp[y:y+h,x:x+w])
            Ids.append(Id)
    return faceSamples,Ids

faces,Ids = getImagesAndLabels('dataSet')
recognizer.train(faces, np.array(Ids))
recognizer.save('recognizer/trainer.yml')
```

Figure 3 trainer.py

The third module of my project is the main application, recognizer.py. Recognize uses the trained LBPH face recognizer we created earlier to compute a confidence score for each face in an image. The confidence score indicates how confident the face recognizer is that it has associated a face with the correct ID. It works in the following way:

1. Create an LBPHFaceRecognizer object called “recognizer” and load the training data created in trainer.py into it.
2. Create another Haar classifer using the frontal face data we used before.
3. Open the webcam as before and detect faces in the same way as before.
4. For each face detected, call recognizer.predict() to obtain the LBPH recognizer’s best prediction for what face is being recognized. This returns a user ID associated with a face and a confidence score. The higher the confidence score, the surer the recognizer is that it’s prediction is correct.
5. If the confidence score is greater than 50, the predictor will write the name associated with the ID prediction in the same square that is being drawn around the face along with the confidence score. I hardcoded the names I wanted associated with the Id numbers, but it wouldn’t be difficult to include this data in the filepath in future versions of the code.
6. If the confidence score is less than 50, “I don’t know who this person is” will be printed in the face square.

```python
import cv2
import numpy as np

#create face recognizer object and load the training data into it
recognizer = cv2.createLBPHFaceRecognizer()
recognizer.load('recognizer/trainer.yml')
#create the face detector using opencv's training data
cascadePath = "haarcascade_frontalface_default.xml"
faceCascade = cv2.CascadeClassifier(cascadePath);
#open webcam
cam = cv2.VideoCapture(0)
#set font
font = cv2.cv.InitFont(cv2.cv.CV_FONT_HERSHEY_SIMPLEX, 1, 1, 0, 1, 1)
while True:
    #read webcam frames
    ret, im =cam.read()
    gray=cv2.cvtColor(im,cv2.COLOR_BGR2GRAY)
    #detect faces
    faces=faceCascade.detectMultiScale(gray, 1.2,5)
    for(x,y,w,h) in faces:
        cv2.rectangle(im,(x,y),(x+w,y+h),(225,0,0),2)
        #predict what ID is being recognized
        Id, conf = recognizer.predict(gray[y:y+h,x:x+w])
        # update the ID based on the prediction to reflect the name associated with the ID
        if(conf > 50):
            if(Id==1):
                Id="Mike : confidence is " + str(conf)
            elif(Id==2):
                Id="Trump : confidence is " + str(conf)
            elif(Id==3):
                Id="Morgan Freeman: confidence is " + str(conf)
            elif(Id==4):
                Id="Leo: confidence is " + str(conf)
        else:
            Id="I don't know this person : confidence is " + str(conf)
        #write the name of the person in the square surrounding their face
        cv2.cv.PutText(cv2.cv.fromarray(im),str(Id), (x,y+h),font, (255,255,255))
    cv2.imshow('im',im)
    if cv2.waitKey(10) & 0xFF==ord('q'):
        break
cam.release()
cv2.destroyAllWindows()
```

Figure 4 recognizer.py

Results

This methodology proved to be relatively accurate. I was able to get a good prediction as long as the lighting conditions were consistent and the faces were oriented towards the camera. A video demonstration can be seen at the following URL: https://www.youtube.com/watch?v=hm2RXrdl6Dk.

In my experiment, I trained the face recognizer to recognize my face, the face of an image of actor Morgan Freeman, and President Donald Trump. The dataset module worked well, however, it did include some images that were either not of a face or were bad data in other ways. Observe that User.1.13.jpg is not an image of my face:

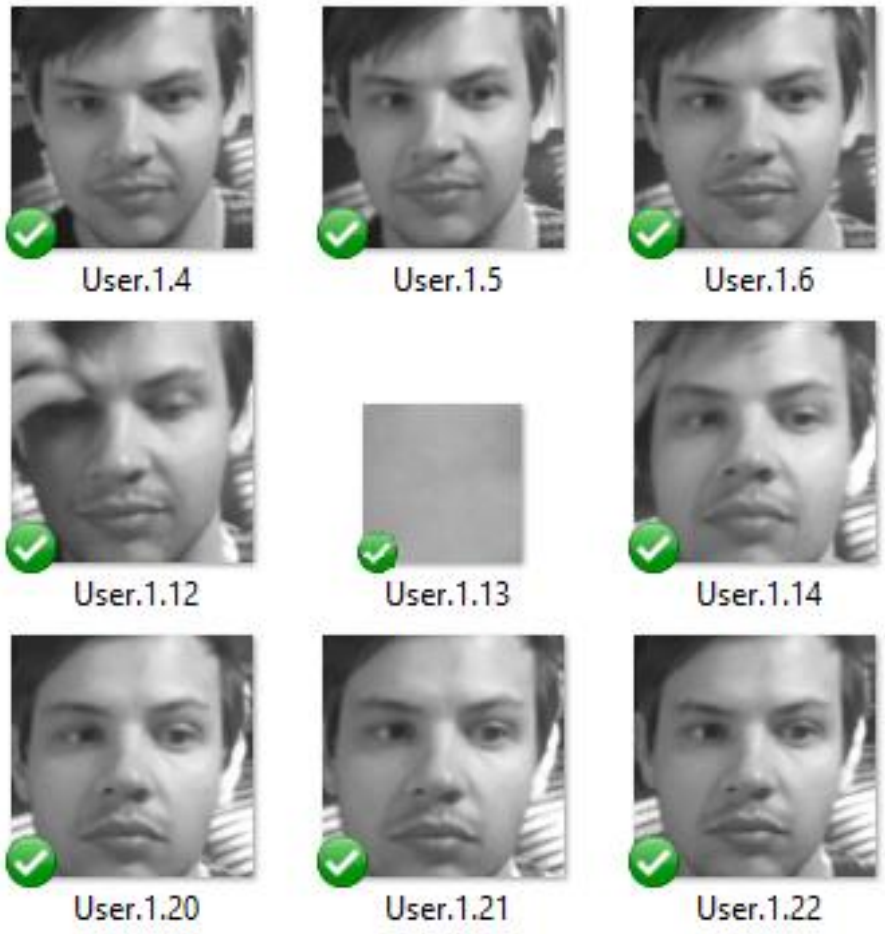

Figure 5 dataset creation with a bad sample image

This was the case for all the users in the dataset. Here are some examples of Trump and Freeman with bad data:

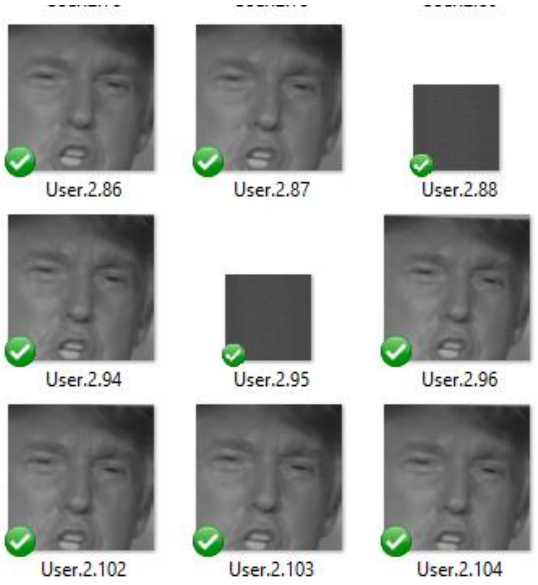

Figure 7 Donald Trump dataset with bad data

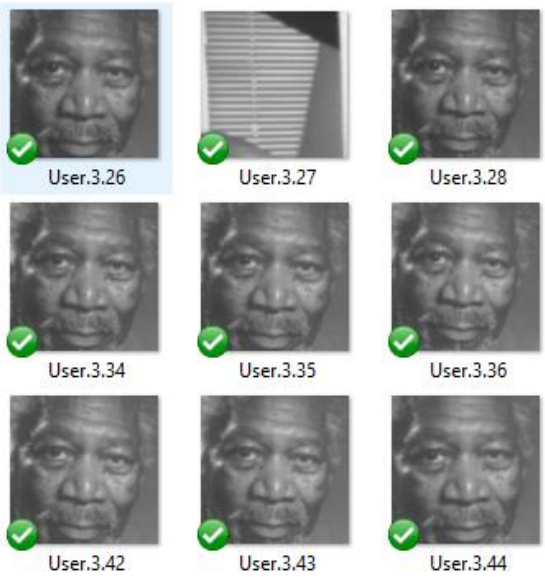

Figure 6 Morgan freeman dataset with bad data

This was mainly due to the fact that occasionally, the Haar classifier would detect objects that were not faces during the dataset creation. Although there was only one face in the frame, the classifier would detect more than one and the image of the non-face object would be stored in the dataset. This negatively affected the results.

The trainer module was difficult to tune. Frankly, it took a lot of trial and error to get the right values for the LBPHFaceRecognizer() parameters. I found that the errors included overconfidence, in which case the recognize seemed to pick one particular ID and classify all faces as that face, or underconfidence, in which the recognizer would not recognize any face at all. The parameters I finally chose worked well in the lighting I was in and with the camera I was using, but users may need to adjust these to work with their hardware and in their room of choice.

After tuning the trainer module, I found that the recognizer module worked well for the most part. It was able to correctly recognize my face and two other faces with few false readings, with fairly high confidence:

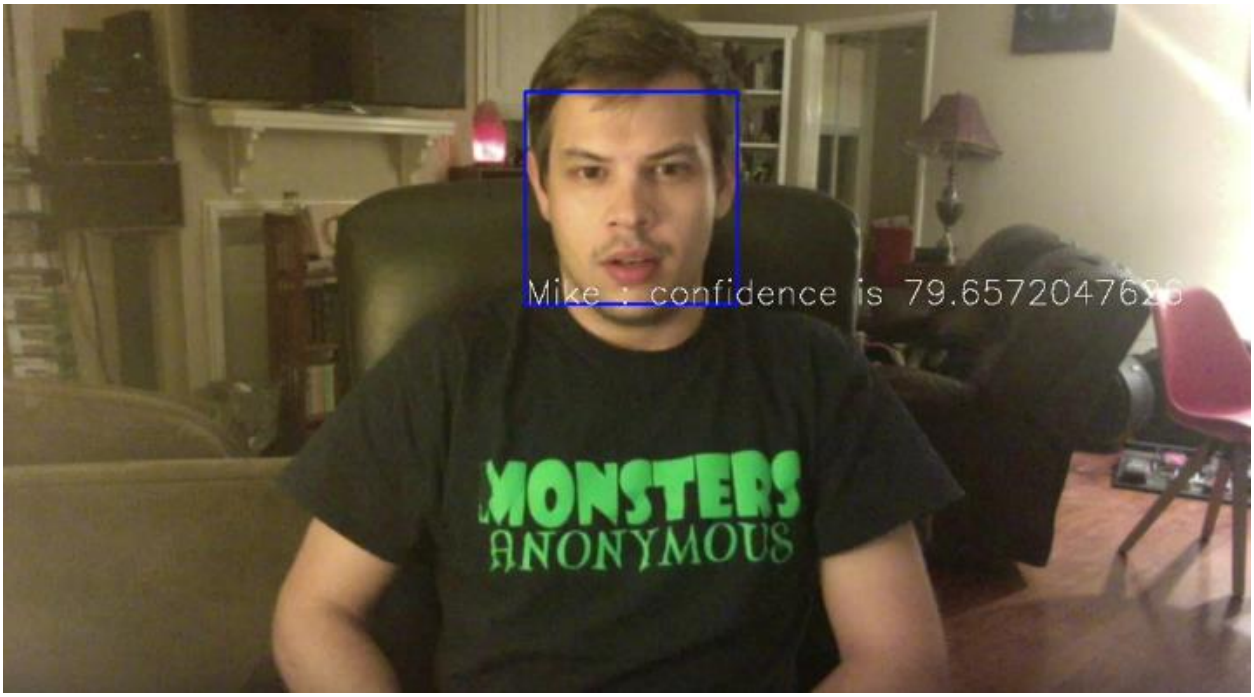

Figure 8 The recognizer recognizes me

Figure 9 the recognizer recognizes Trump

Figure 10 The Recognize recognizes Morgan Freeman

However, the program exhibited some undesirable behavior as well. For instance, the Haar classifier would occasionally judge that some inanimate object was a face and the LBHPFaceRecognizer would determine with inexplicably high confidence that the face was mine. This would usually only last a few frames.

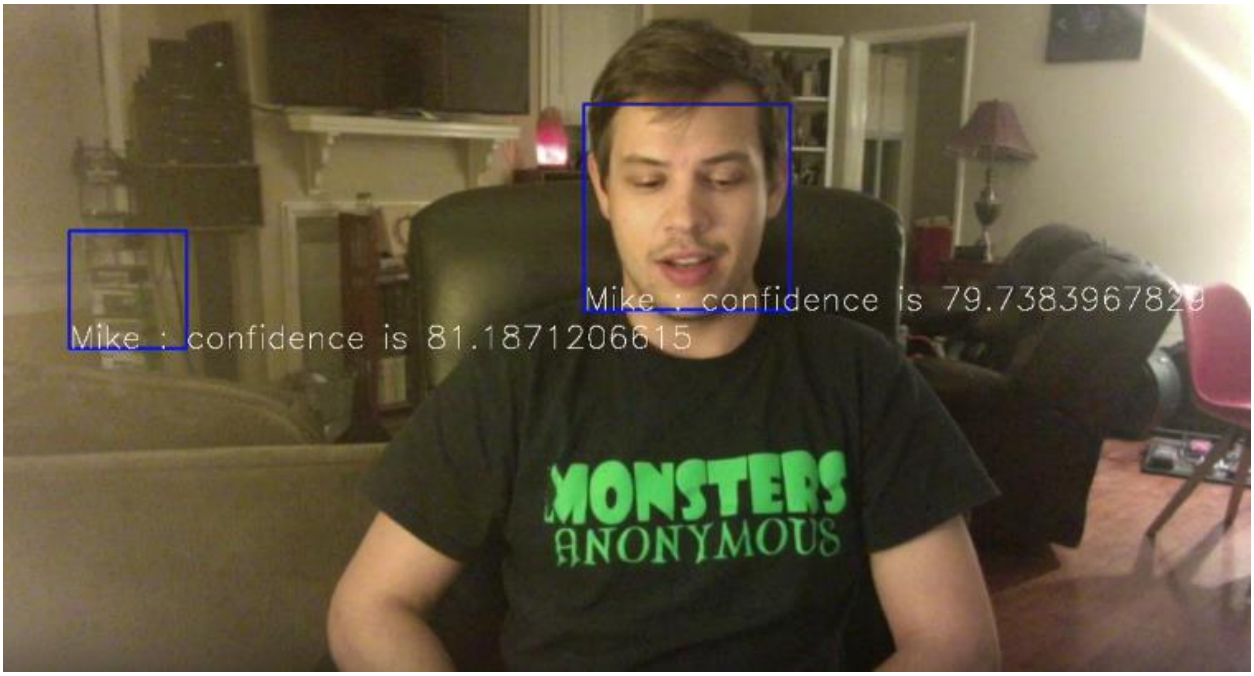

Figure 11 the recognizer thinks the shelf is my face

Additionally, the recognizer program would occasionally make a false prediction that lasted for a few frames.

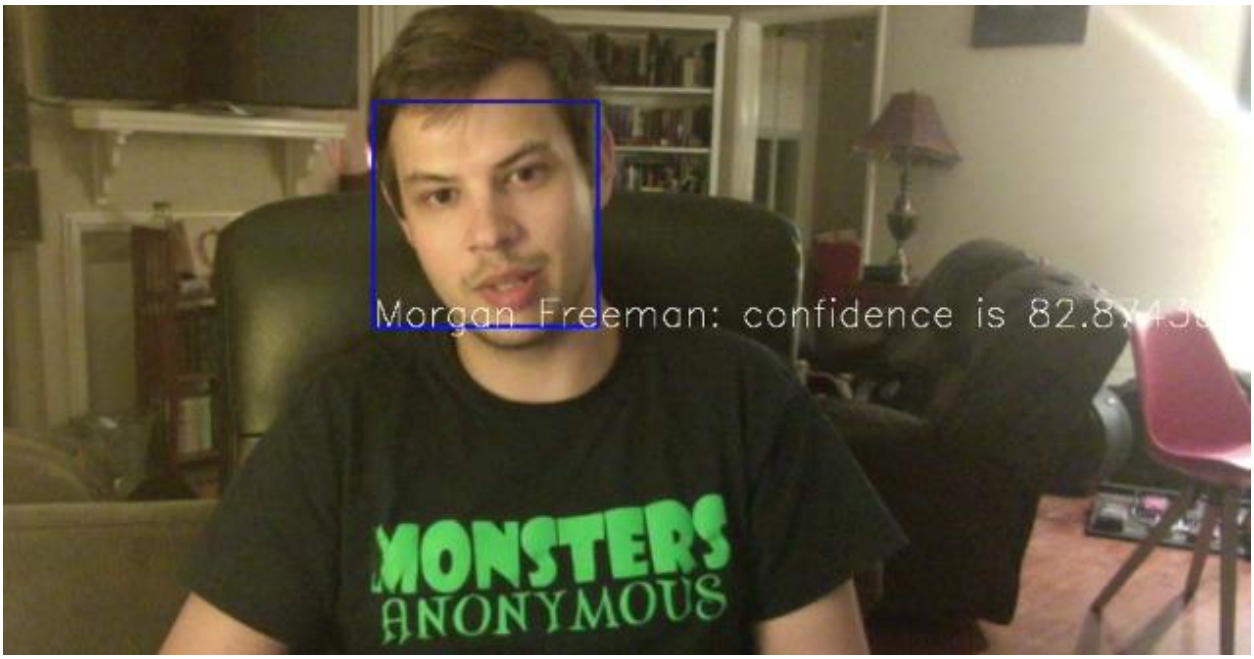

Figure 12 a momentary false prediction

Sometimes, the recognizer would fail to recognize even the most unique of faces.

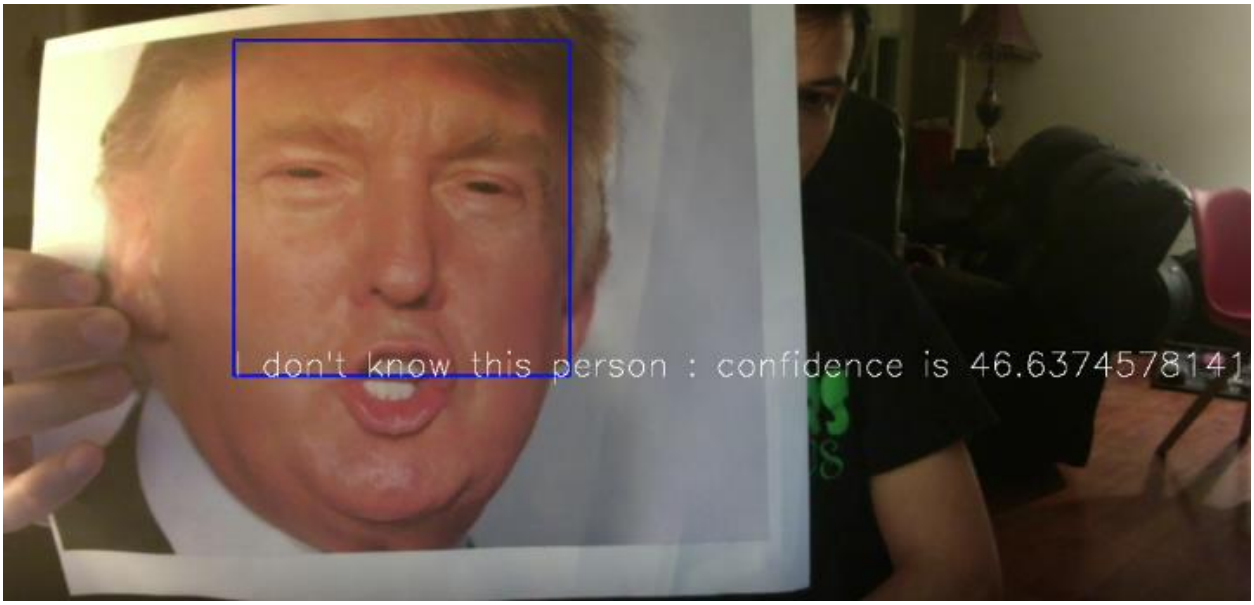

Figure 13 low confidence score

Conclusions

I found that although I was able to tune the program to respond well in my setting, it would probably not work well for most other hardware setups. Additionally, as can be seen in the demo video, the program was very slow. Even though my computer has 24GB of RAM, 4GB of VRAM, a quad core processor, and a solid-state drive, the video has a great deal of lag. This may be due in part to the wait time between frame captures I implemented. However, I think implementing this program in C++ would yield a better result in terms of speed.

This method is limited by the fact that I only used a Haar classifier that recognized faces from a front facing angle. One way to approve recognition would be to use more than one classifier. Also, it would be very helpful to have a pre-made dataset because of the false readings of the haar classifier that corrupted some of the training data. However, having a dataset creator module is convenient because many people don't have very many photographs of themselves.

References

[1] Anirban, "Object Recognition In Any Background Using OpenCV Python," 26 March 2017. [Online]. Available: http://thecodacus.com/object-recognition-using-opencv-python/#.WQqjpIjysuU. [Accessed 03 05 2017].

[2] S. Soo, "Object detection using Haar-cascade Classifier," [Online]. Available: http://ds.cs.ut.ee/Members/artjom85/2014dss-course-media/Object%20detection%20using%20Haar-final.pdf.

[3] Y.-S. L. W.-S. N. C.-W. Liu, "A comparison of different face recognition algorithms," [Online]. Available: http://cs.unc.edu/~chunwei/papers/2009pr_face_recog.pdf. [Accessed 03 05 2017].

[4] E. Arubas, "Face Detection and Recognition (Theory and Practice)," 6 April 2013. [Online]. Available: http://eyalarubas.com/face-detection-and-recognition.html. [Accessed 03 05 2017].