FACE RECOGNITION (SYSTEM)
RESEARCH

1. FACE RECOGNITION DESIGN ARCHITECTURE
   The objective is the system to be able to distinguish between people that are authorized, and the non authorized based on their faces just like humans do.
   Let's jump to how the objective can be achieved.

   Methodology:

I. Accept image input from webcam / portable camera.
II. Convert the image into grayscale to make it easier to work with. Grayscale images gives a 2D array that is easier to manipulate than colored images that gives a 3D array which is more complicated in terms of computations.
III. Detect the location of the faces in the image by using the Histogram of Oriented Gradients (HOG) algorithm. The HOG algorithm takes a grayscale image, the it goes through every pixel in the image, For every single pixel, we want to look at the pixels that directly surrounding it:

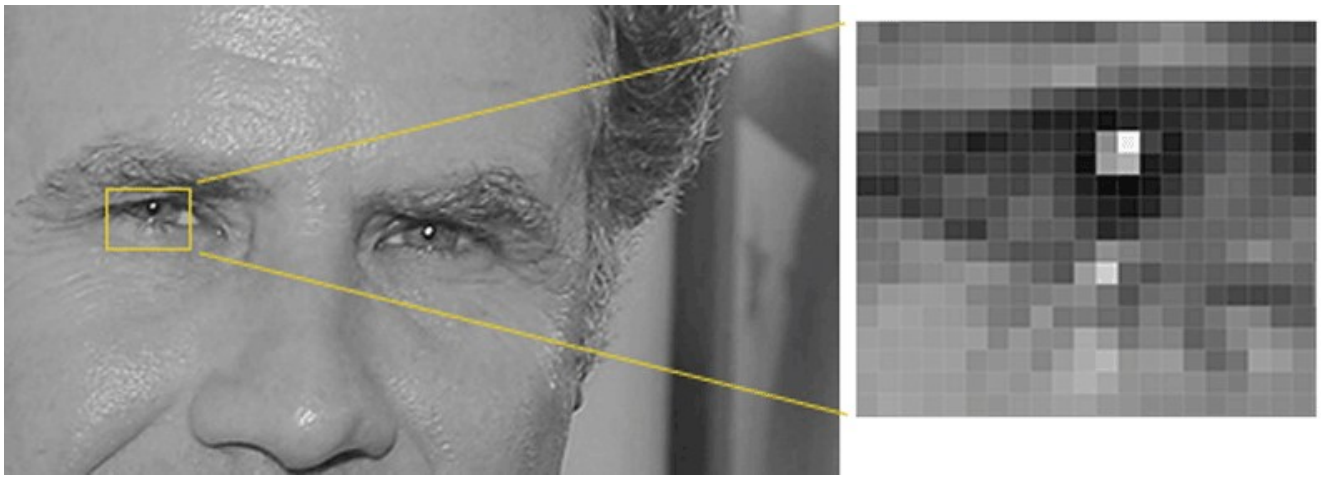

Illustration 1: Detecting pixels and their surrounding pixels. *~Geitgey – Machine Learning is fun*

The next step is to figure out how dark the current pixel is compared to the pixels that surrounds it. Then draw an arrow showing in which direction the image is getting darker.

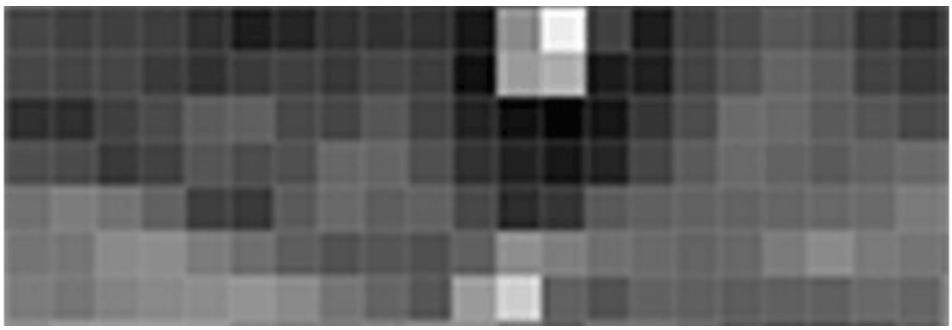

Illustration 2: Arrow showing the direction of darkness

Doing so on the entire image, we will end up having an image made of arrows. These arrows are called Gradients, and they show the flow from light to dark across the entire image.

Illustration 3: Image pixels replaced by arrows

The reason for using arrows instead of pixels directly is that, if we use pixels directly, really dark images and light images of the same person will give totally different values. But using arrows, the values will remain the same despite the darkness of lightness of the image used.

Although, all these arrows gives us a lot of information to process, we need to see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

To do this, we'll break up the image into small squares of 16x16 pixels each. In each square, we'll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we'll replace that square in the image with the arrow directions that were the strongest.

The end result is we turn the original image into a very simple representation that captures the basic structure of a face in a simple way:

Illustration 4: Image turned into HOG form

To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces.

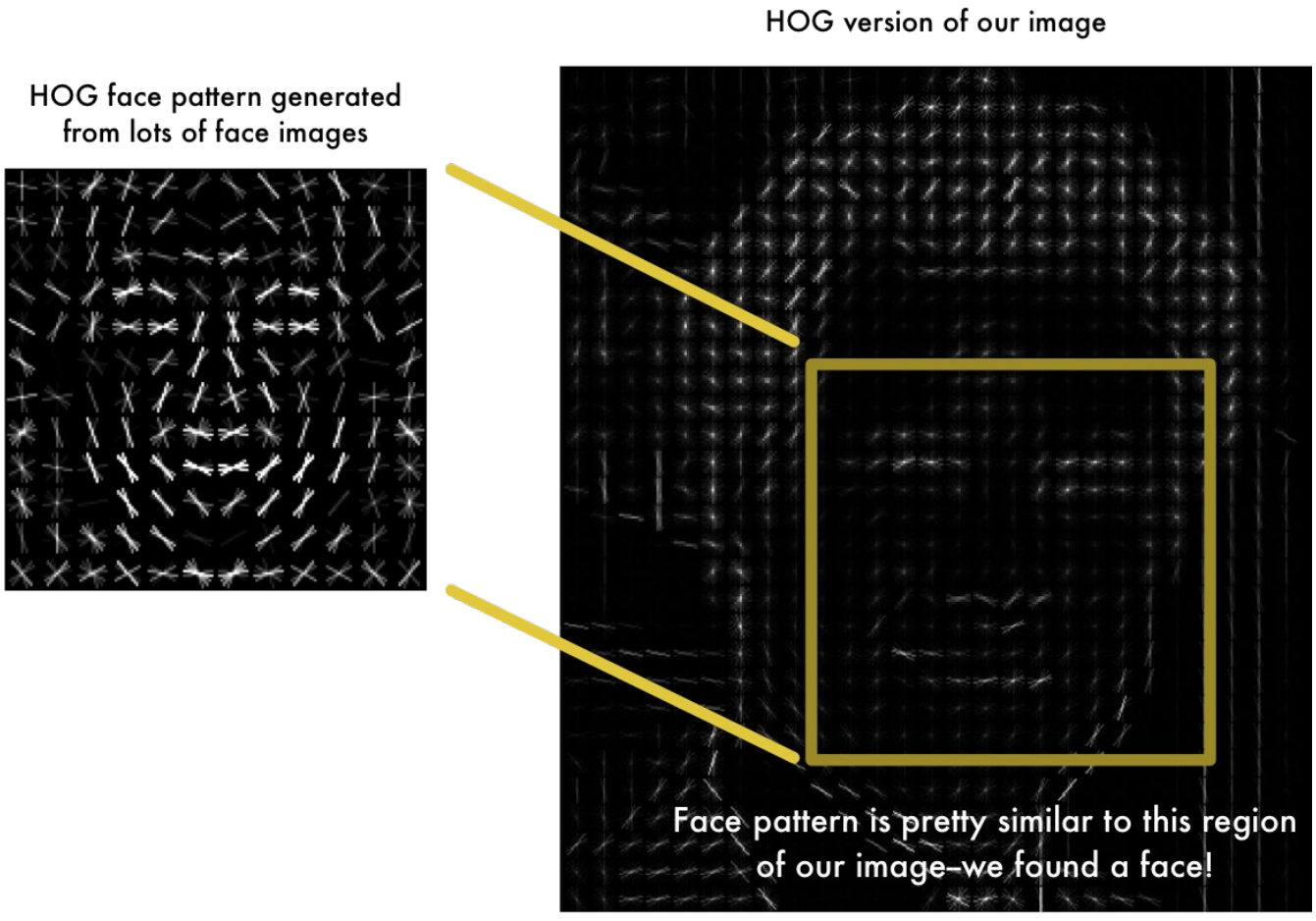

Illustration 5: Detecting face location using HOG

Using this technique, we can easily find faces in any image

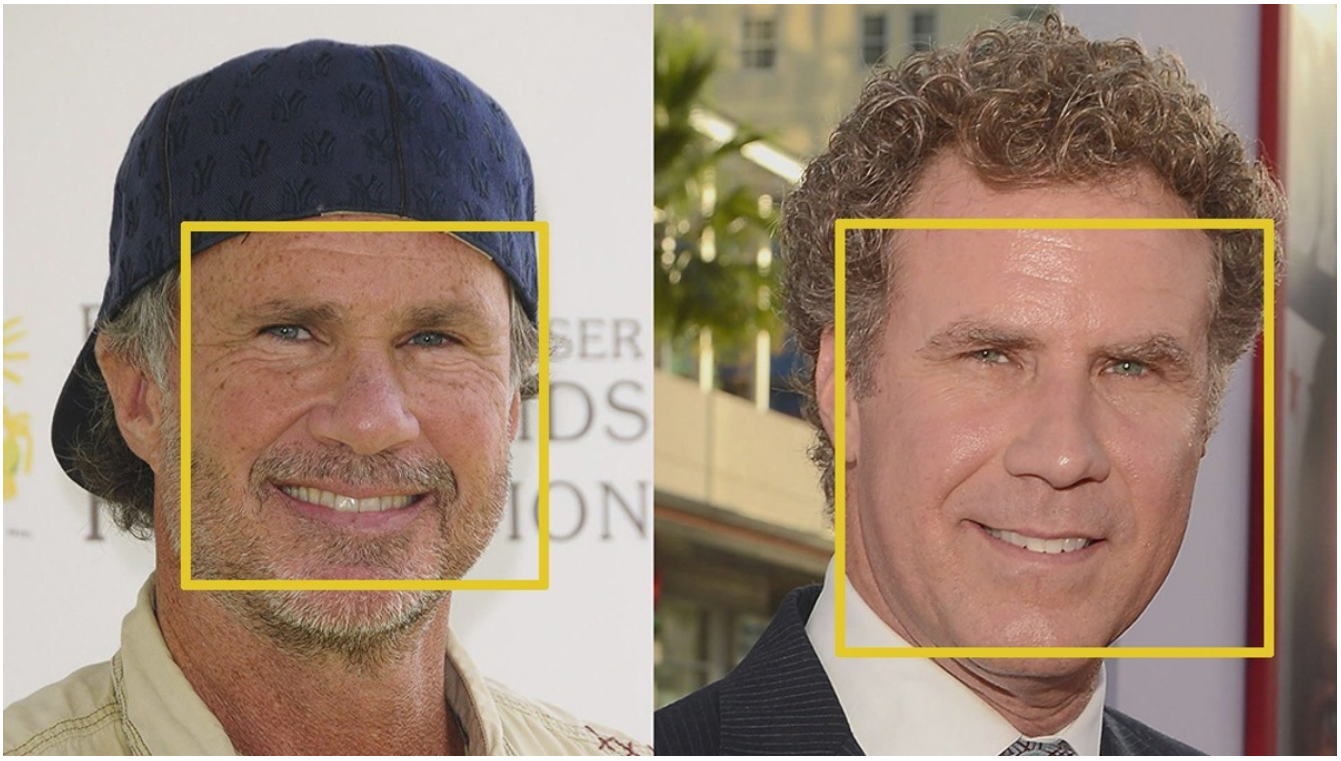

Illustration 6: Detecting face locations

IV. Posing and Projecting faces.

As we have already detected faces, the real problem is dealing with images that points in different directions. Faces of same person turned into different directions looks totally different to a computer.

To solve this, we will try to warp each picture so that the eyes and lips are always in the sample place in the image. This will make it a lot easier for us to compare faces in the next steps.

To do this, we are going to use an algorithm called **face landmark estimation**. There are lots of ways to do this, but we are going to use the approach invented in 2014 by Vahid Kazemi and Josephine Sullivan.

The basic idea is we will come up with 68 specific points (called landmarks) that exist on every face → the top of the chin, the outside edge of each eye, the inner edge of each eyebrow, etc. Then we will train a machine learning algorithm to be able to find these 68 specific points on any face.

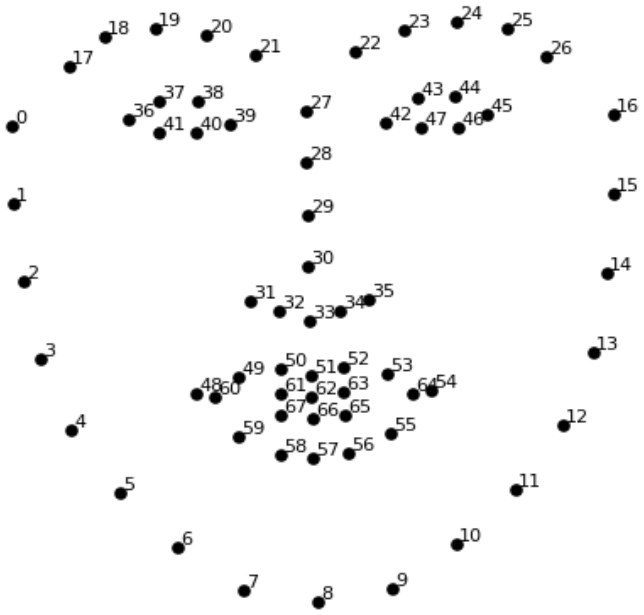

Illustration 7: T*he 68 landmarks we will locate on every face. This image was created by Brandon Amos of CMU who works on OpenFace.*

Here are the results of locating face locations and drawing 68 landmarks on the face.

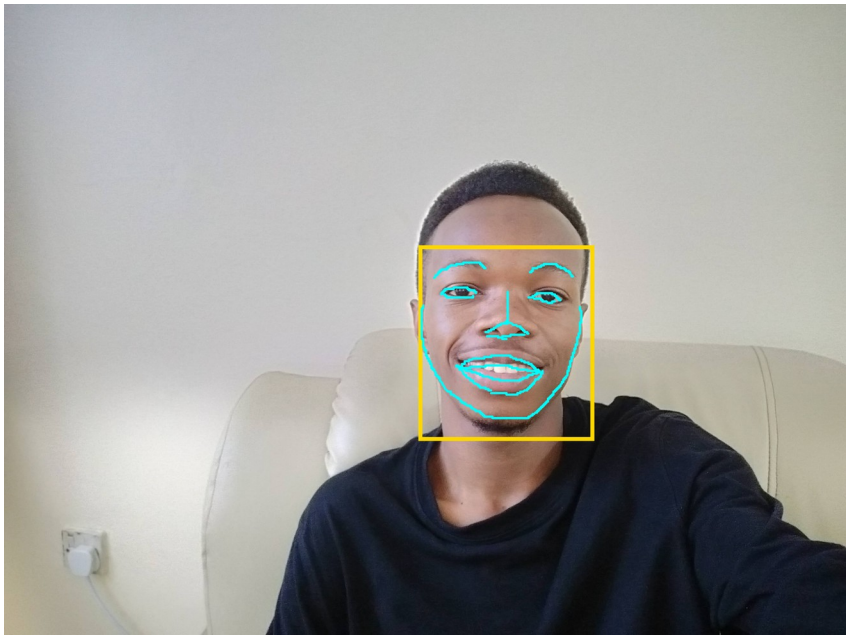

Illustration 8: drawing 68 landmarks on face.

Drawing these landmarks makes it easier for us to rotate, scale, and shear the image to make sure the eyes and mouth are centered as best as possible.

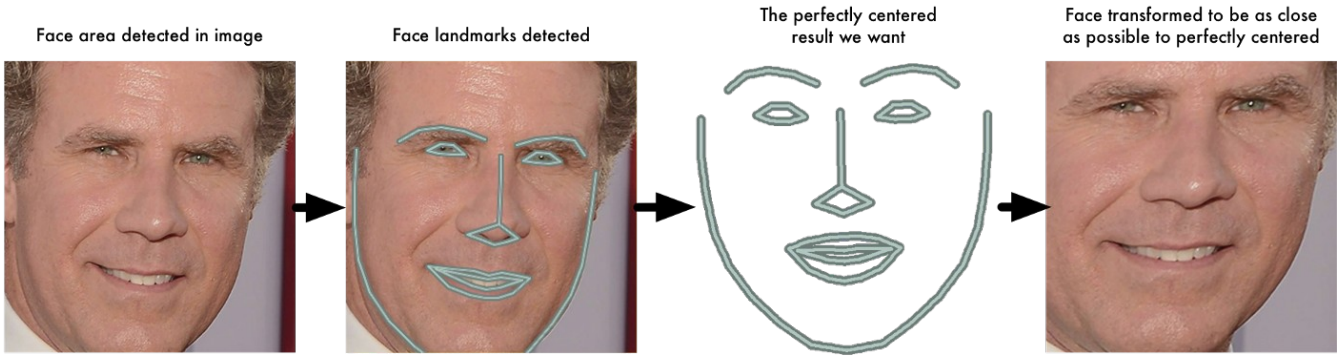

Illustration 9: projecting faces to always face forward for recognition

V. Encoding Faces

Here now we will tell that the face is known or not known, we can go further and give information of the person in the image.

The simplest approach to face recognition is to directly compare the unknown face we received from the webcam or camera with all the pictures we have of people that have already been tagged. When we find a previously tagged face that looks very similar to our unknown face, it must be the same person.

We do not need to loop through all features on the face for recognition as it will take too long, what we need is to extract a few basic measurements from each face. Then we could measure our unknown face the same way and find the known face with the closest measurements.

What features should we take for measurements?

It turns out that the measurements that seem obvious to humans (like eye color) don't really make sense to a computer looking at individual pixels in an image. The most accurate approach is to let the computer figure out the measurements to collect itself. Deep learning does a better job than humans at figuring out which parts of a face are important to measure.

The solution is to train a Deep Convolutional Neural Network. We have to train a neural network to generate 128 measurements for each face. The 128 measurements are called the **embedding.** The training requires a lot of data and computing power, lucky for us, the fine folks at OpenFace already did this and they published several trained networks which we can directly use. Thanks Brandon Amos and team!

So all we have to do is run our face images through their pre-trained network to get the 128 measurements for each face. Here's the measurements for the test image.

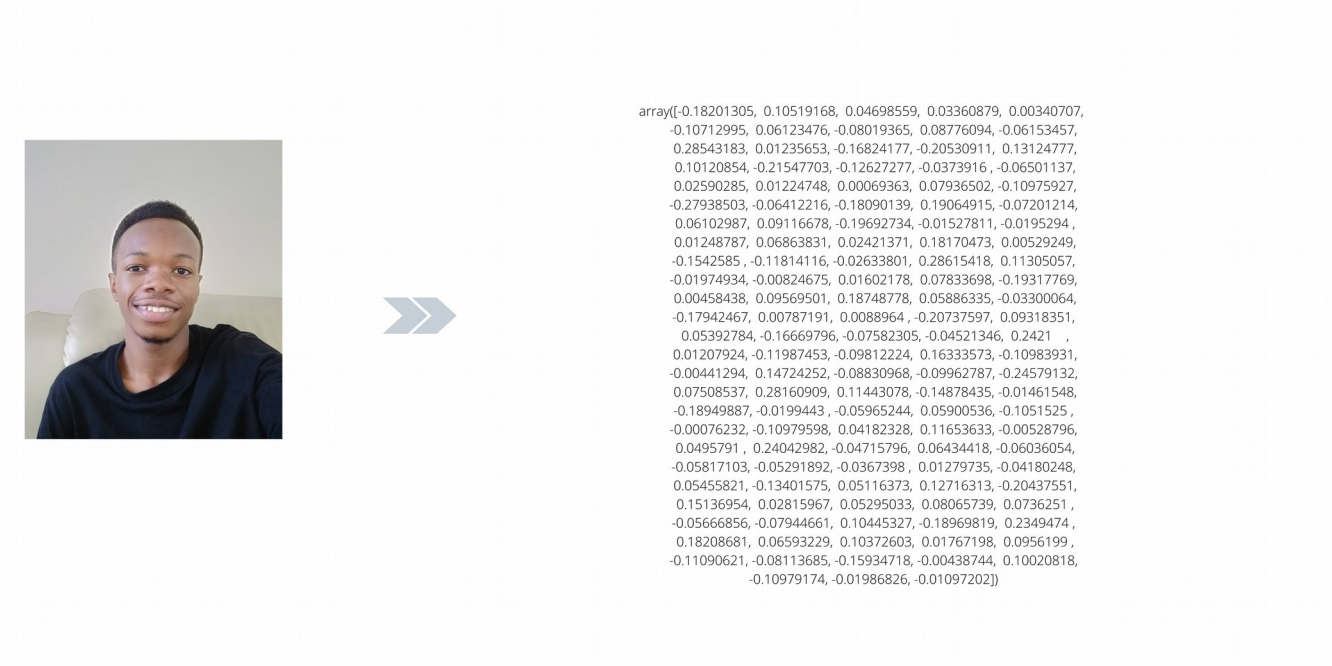

Illustration 10: The resulting 128 measurements from the network

So what parts of the face are these 128 numbers measuring exactly? It turns out that we have no idea. It doesn't really matter to us, all that we care is that the network generates nearly the same numbers when looking at two different pictures of the same person.

VI. Finding if person is authorized or non authorized

This last step is the easiest step in the whole process. All we have to do is find the person in our database of known people who has the closest measurements to our test image. If there exists an image that gives close measurements to the unknown image, then we confirm that the person is authorized, otherwise we display that the person is unauthorized. We can also return the name of the recognized person. I tried to implement both for displaying name of the recognized person, and also for just showing if the person is valid or not. Since this research is based on a quick recognition for authorized personnel only, we will display the 'Authorized' and 'Unknown' messages.

We will use the python library for face recognition developed by Adam Geitgey, 2016. The library contains the network trained on top of openFace model and it comes with a variety of functions such as comparing faces, finding face distances, finding face locations and more other fancy functions.

FINAL OUTPUTS

The final outputs of the implementation gave satisfying results. Although I had to downstream the decision value for confirming a person to be less than 0.4 that is a person should match for at least 75 – 80% and above. This is to make sure the system does not confirm a face that has a little resemblance to one of the faces in the database.

Training Images:

Illustration 12: Steve

Illustration 11: Jackie Chan

Test images:

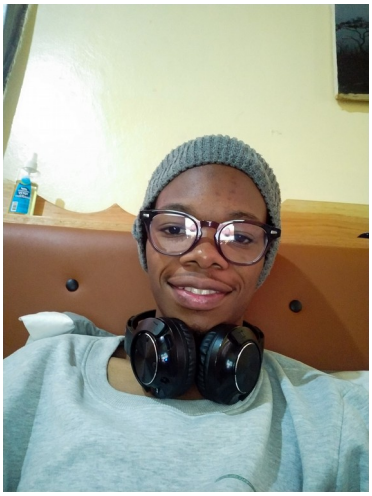

Illustration 13: Steve Test

Illustration 14: Jackie Chan Test

Results:

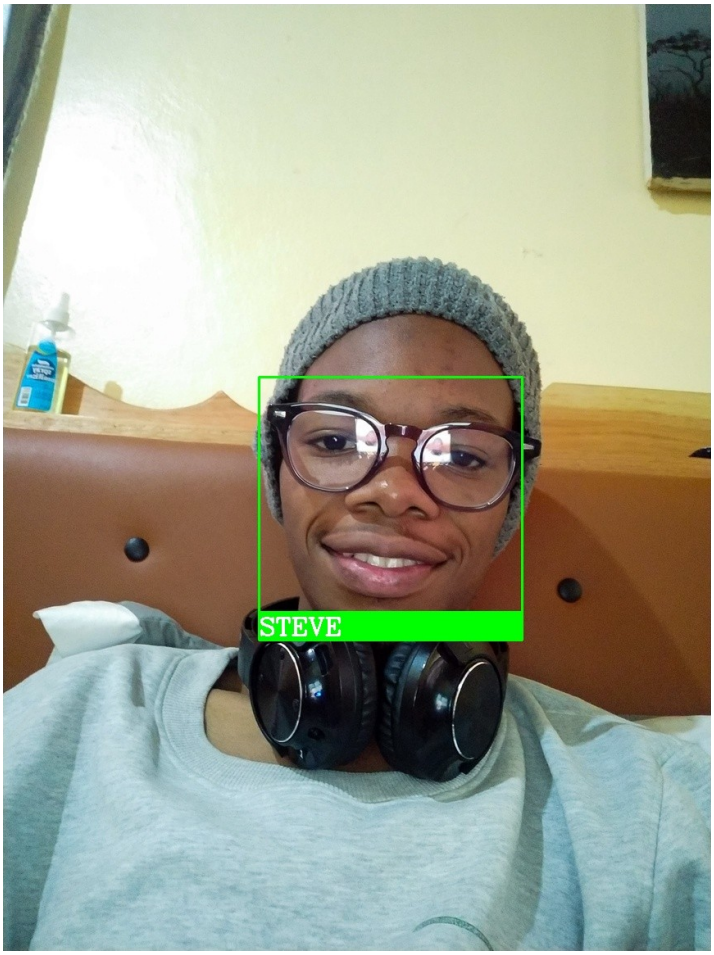

Illustration 15: Processed output

Illustration 16: Processed output

So you can see that the model was able to recognize the faces in spite of the fact that the testing images were somehow different from the training images. It is impressive that it was able to recognize Steve despite the glasses and the beanie hat. It also recognized Jackie Chan while smiling and turned the other way from the direction of the trained image.

Some results from the other code for displaying only Authorized and Unknown.

Illustration 17: Authorized

Illustration 18: Unauthorized