Face Recognition System

Purushotham Konanki(11705279)

*Computer Science Engineering*

*Lovely Professional University*

*Phagwara, Punjab, India*

[purushnaidu143@gmail.com](mailto:purushnaidu143@gmail.com)

*Abstract*—Face recognition is a way of recognizing a human face through technology. A facial recognition system uses biometrics to map facial features from a photograph or a video. It compares the information with database of known faces to find match

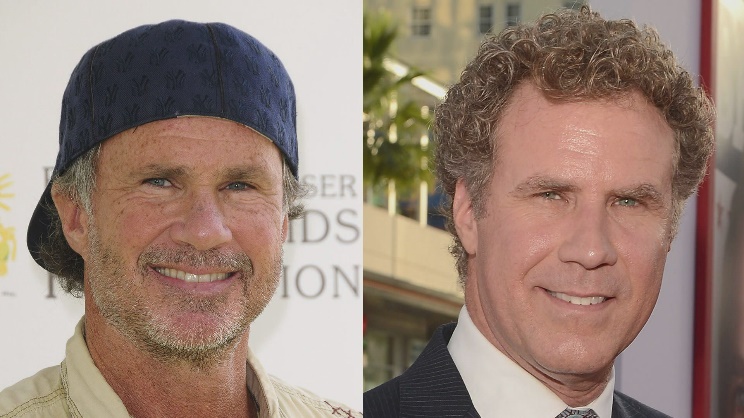
Keywords—Face recognition, openCV, python.

# **Introduction**

Have you noticed that Facebook has developed an uncanny ability to recognize your friends in your photographs? In the old days, Facebook used to make you to tag your friends in photos by clicking on them and typing in their name. Now as soon as you upload a photo, Facebook tags everyone for you like magic

This technology is called face recognition. Facebook’s algorithms are able to recognize your friends’ faces after they have been tagged only a few times. It’s pretty amazing technology — Facebook can recognize faces with 98% accuracy which is pretty much as good as humans can do!

Let’s learn how modern face recognition works! But just recognizing your friends would be too easy. We can push this tech to the limit to solve a more challenging problem — telling Will Ferrell (famous actor) apart from Chad Smith (famous rock musician)!



But face recognition is really a series of several related problems:

First, look at a picture and find all the faces in it

Second, focus on each face and be able to understand that even if a face is turned in a weird direction or in bad lighting, it is still the same person.

Third, be able to pick out unique features of the face that you can use to tell it apart from other people— like how big the eyes are, how long the face is, etc.

Finally, compare the unique features of that face to all the people you already know to determine the person’s name. As a human, your brain is wired to do all of this automatically and instantly. In fact, humans are too good at recognizing faces and end up seeing faces in everyday objects.Computers are not capable of this kind of high-level generalization (at least not yet…), so we have to teach them how to do each step in this process separately.We need to build a pipeline where we solve each step of face recognition separately and pass the result of the current step to the next step.

**Face recognition step by step:**

Let’s tackle this problem one step at a time. For each step, we’ll learn about a different machine learning algorithm. I’m not going to explain every single algorithm completely to keep this from turning into a book, but you’ll learn the main ideas behind each one and you’ll learn how you can build your own facial recognition system in Python using OpenFace and dlib.

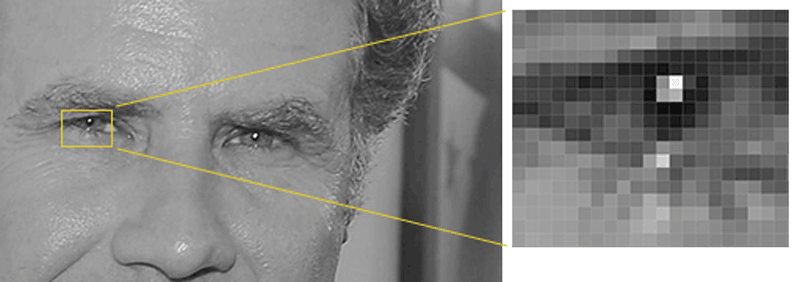
**Step 1: Finding all faces**

The first step in our pipeline is face detection. Obviously we need to locate the faces in a photograph before we can try to tell them apart! If you’ve used any camera in the last 10 years, you’ve probably seen face detection in action. Face detection is a great feature for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. But we’ll use it for a different purpose — finding the areas of the image we want to pass on to the next step in our pipeline.

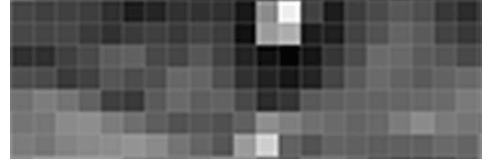
Face detection went mainstream in the early 2000's when Paul Viola and Michael Jones invented a way to detect faces that was fast enough to run on cheap cameras. However, much more reliable solutions exist now. We’re going to use a method invented in 2005 called Histogram of Oriented Gradients — or just HOG for short. To find faces in an image, we’ll start by making our image black and white because we don’t need color data to find faces:



Then we’ll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surrounding it:



Our goal is to figure out how dark the current pixel is compared to the pixels directly surrounding it. Then we want to draw an arrow showing in which direction the image is getting darker:



If you repeat that process for every single pixel in the image, you end up with every pixel being replaced by an arrow. These arrows are called gradients and they show the flow from light to dark across the entire image:



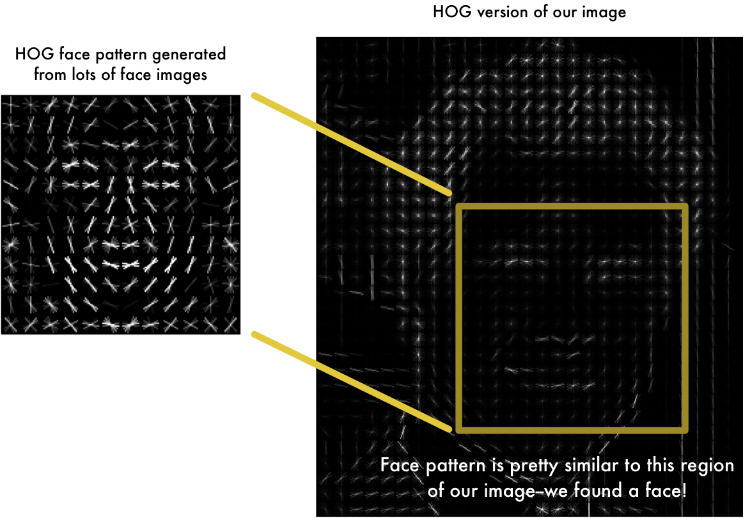
This might seem like a random thing to do, but there’s a really good reason for replacing the pixels with gradients. If we analyze pixels directly, really dark images and really light images of the same person will have totally different pixel values. But by only considering the direction that brightness changes, both really dark images and really bright images will end up with the same exact representation. That makes the problem a lot easier to solve!

But saving the gradient for every single pixel gives us way too much detail. We end up missing the forest for the trees. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

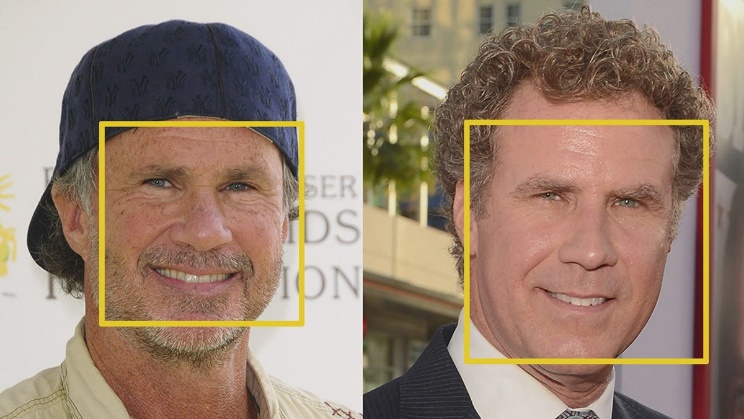
To do this, we’ll break up the image into small squares of 16x16 pixels each. In each square, we’ll count up how many gradients point in each major direction (how many point up, point up-right, point right, etc…). Then we’ll replace that square in the image with the arrow directions that were the strongest.

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:



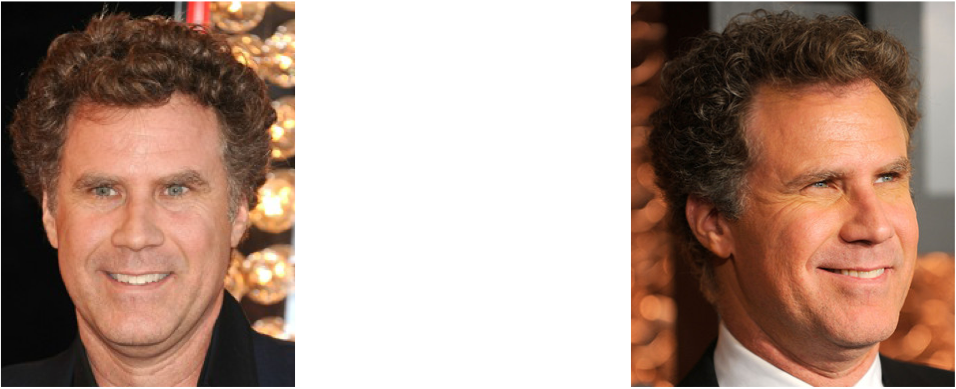
To find faces in this HOG image, all we have to do is find the part of our image that looks the most similar to a known HOG pattern that was extracted from a bunch of other training faces:  


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



**Step 2: Posing and Projecting Faces**

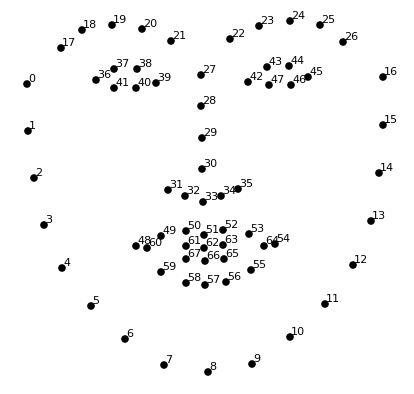
Whew, we isolated the faces in our image. But now we have to deal with the problem that faces turned different directions look totally different to a computer:



To account for 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.

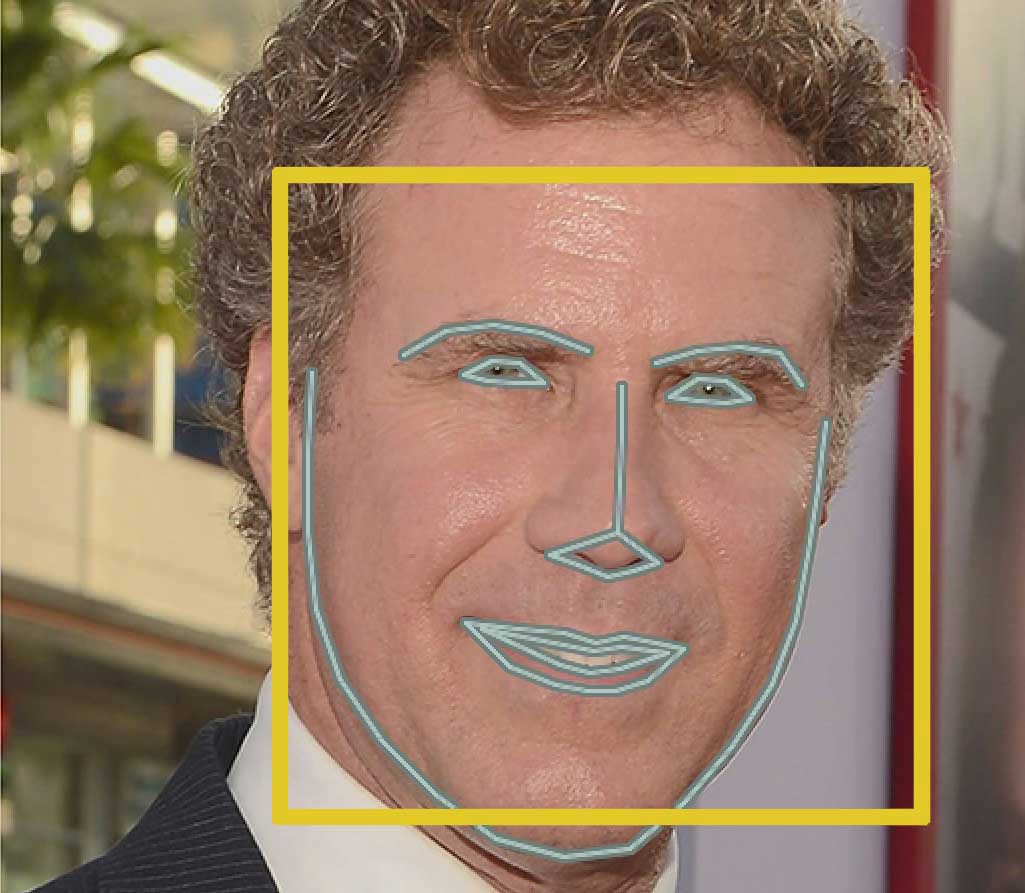
To do this, we are going to use an algorithm called face landmark estimation. There are lots of ways to do this, but we are going to use the approach invented in 2014 by Vahid Kazemi and Josephine Sullivan.

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:

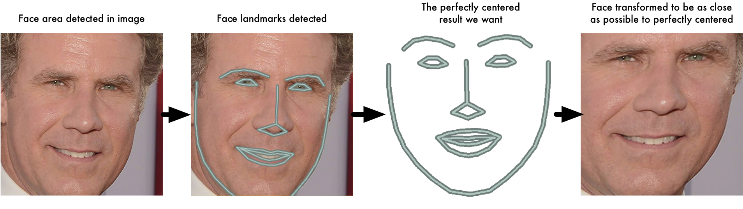


The 68 landmarks we will locate on every face. This image was created by Brandon Amos of CMU who works on OpenFace.

Here’s the result of locating the 68 face landmarks on our test image:



Now that we know were the eyes and mouth are, we’ll simply rotate, scale and shear the image so that the eyes and mouth are centered as best as possible. We won’t do any fancy 3d warps because that would introduce distortions into the image. We are only going to use basic image transformations like rotation and scale that preserve parallel lines (called affine transformations):



Now no matter how the face is turned, we are able to center the eyes and mouth are in roughly the same position in the image. This will make our next step a lot more accurate.

**Step 3: Encoding Faces**

Now we are to the meat of the problem — actually telling faces apart. This is where things get really interesting!

The simplest approach to face recognition is to directly compare the unknown face we found in Step 2 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. Seems like a pretty good idea, right?

There’s actually a huge problem with that approach. A site like Facebook with billions of users and a trillion photos can’t possibly loop through every previous-tagged face to compare it to every newly uploaded picture. That would take way too long. They need to be able to recognize faces in milliseconds, not hours.

What we need is a way 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. For example, we might measure the size of each ear, the spacing between the eyes, the length of the nose, etc. If you’ve ever watched a bad crime show like CSI, you know what I am talking about:

The most reliable way to measure a face

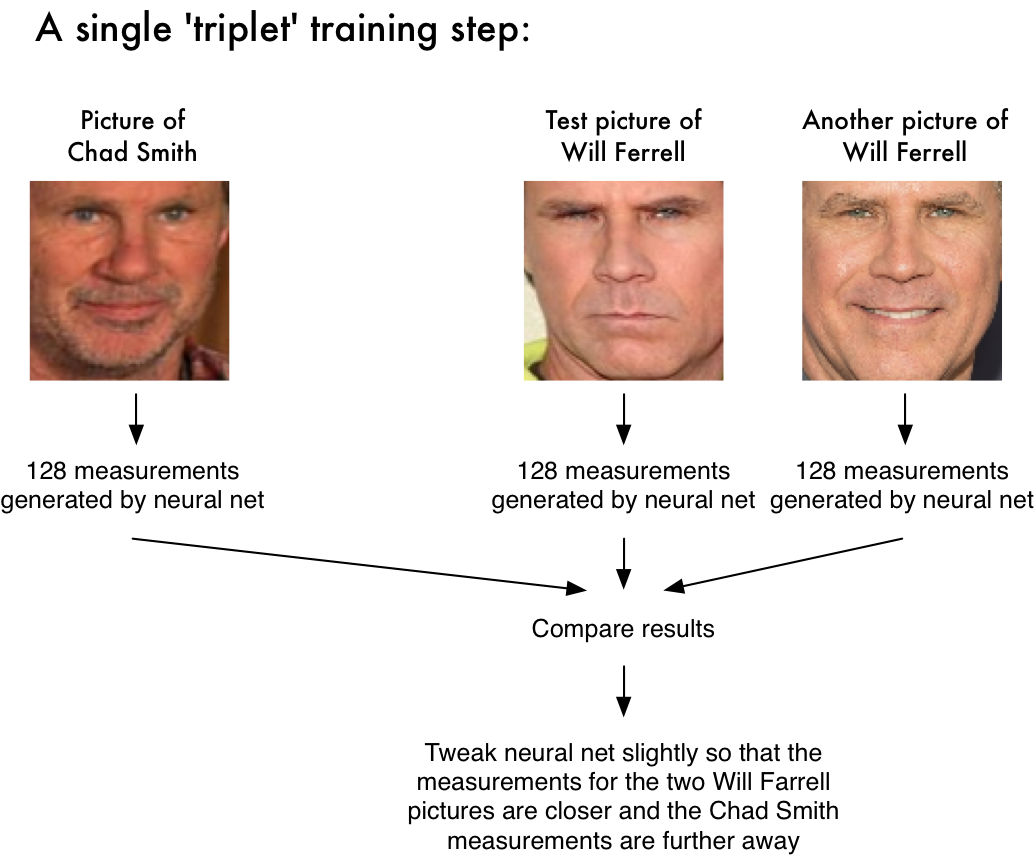
Ok, so which measurements should we collect from each face to build our known face database? Ear size? Nose length? Eye color? Something else?

It turns out that the measurements that seem obvious to us humans (like eye color) don’t really make sense to a computer looking at individual pixels in an image. Researchers have discovered that 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. But instead of training the network to recognize pictures objects like we did last time, we are going to train it to generate 128 measurements for each face.

The training process works by looking at 3 face images at a time:

1. Load a training face image of a known person
2. Load another picture of the same known person
3. Load a picture of a totally different person



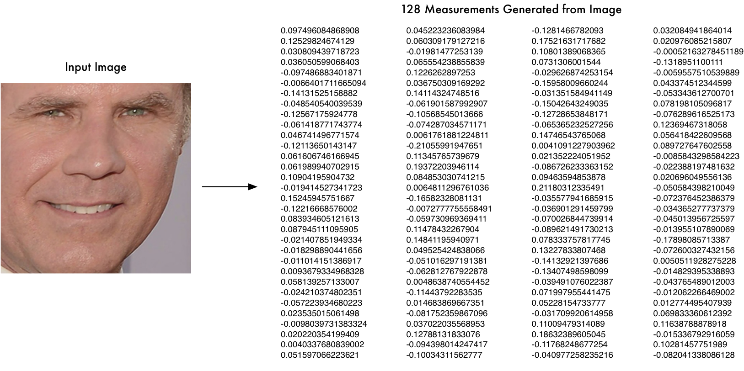
After repeating this step millions of times for millions of images of thousands of different people, the neural network learns to reliably generate 128 measurements for each person. Any ten different pictures of the same person should give roughly the same measurements.

**Encoding our face image**

This process of training a convolutional neural network to output face embeddings requires a lot of data and computer power. Even with an expensive NVidia Telsa video card, it takes about 24 hours of continuous training to get good accuracy.

But once the network has been trained, it can generate measurements for any face, even ones it has never seen before! So this step only needs to be done once. 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 need to do ourselves is run our face images through their pre-trained network to get the 128 measurements for each face. Here’s the measurements for our test image.



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.

**Step 4: Finding the person’s name from the encoding**

This last step is actually 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.

All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

##### **References**

##### I took reference from a website which is helped me at some problems

<https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78>