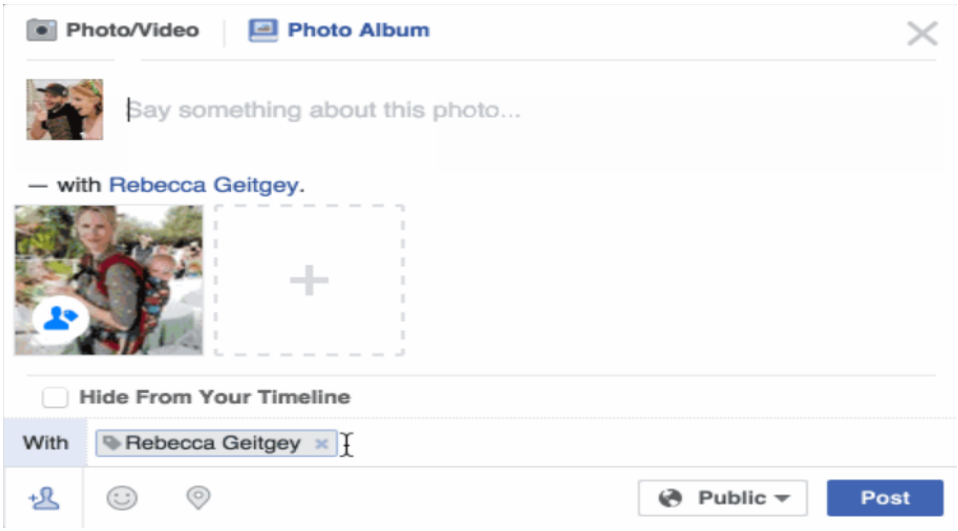
Face Detection vs Face Recognition

Among other tasks made possible through machine learning algorithms, face detection and recognition are a crucial computer vision task. To begin with, both face detection and recognition are co-related yet colloquially different. Face detection is a wider aspect than face recognition and is applied with the help of machine learning. Whether it is about face detection in surveillance, mapping images for medical diagnosis, or deep analysis of human faces in videos for intelligence purposes, studies have led to application of multiple face detection ML approaches such as Viola Jones algorithm, Kohonen approach, Local Binary Pattern Histogram (LBPH), Eigenfaces, Karhunen-Loeve detector. These approaches have helped in refining machine learning algorithms, which are today used in many faces detection softwares and combined with ML models to provide accurate results. ML models apply for both detection and recognition purposes

Face Detection is the process of detecting faces, from an image or a video that doesn't matter. The program doesn't do anything more than finding the faces. But on the other hand, face recognition, the program that finds the faces and it can tell which face belongs to who. So, it is more informational than just detecting them.

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 can recognize your friends' faces after they have been tagged only a few times. It's amazing technology — Facebook can recognize faces with 98% accuracy which is pretty much as good as humans can do!

Face recognition is really a series of several related problems:

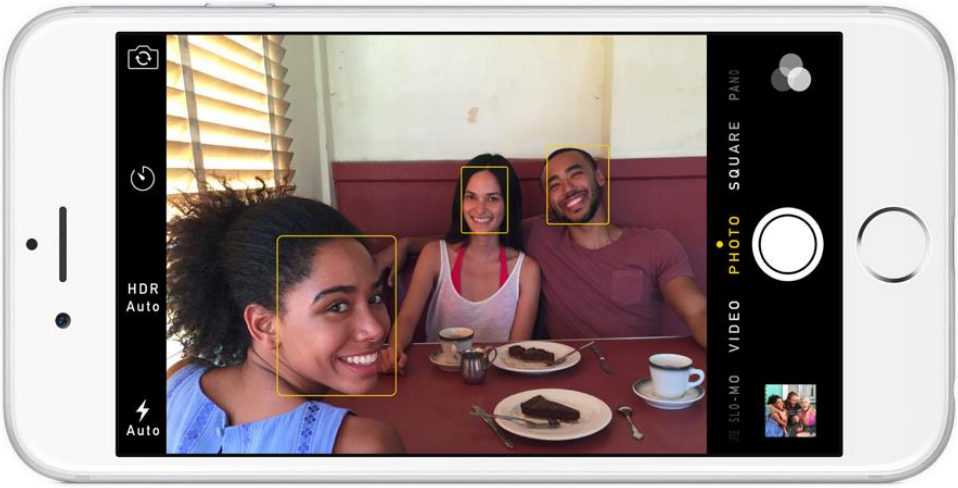
look at a picture and find all the faces in it

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.

compare the unique features of that face to all the people you already know to determine the person's name.

Let's talk about each problem in details….

Problem 1: Finding all the Faces

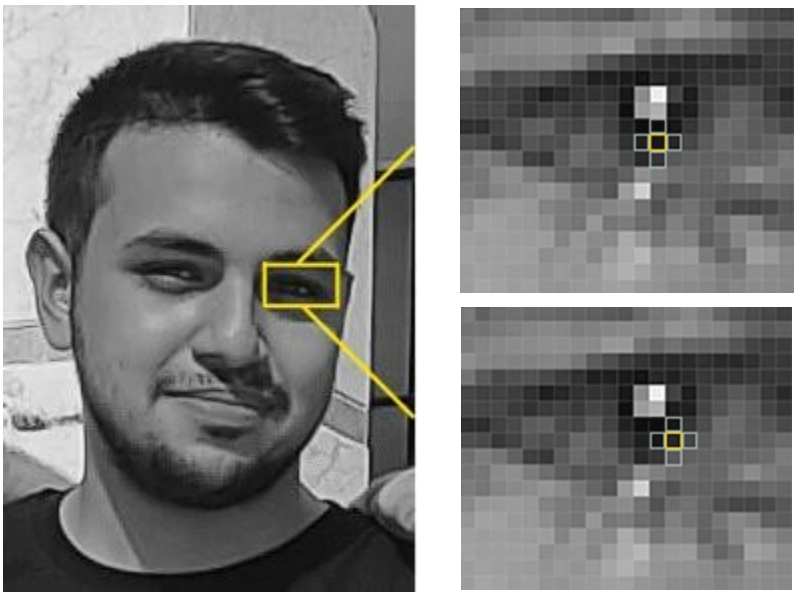


Face detection is a great feature for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. But we'll use it for a different purpose — finding the areas of the image we want to pass on to the next step in our pipeline.

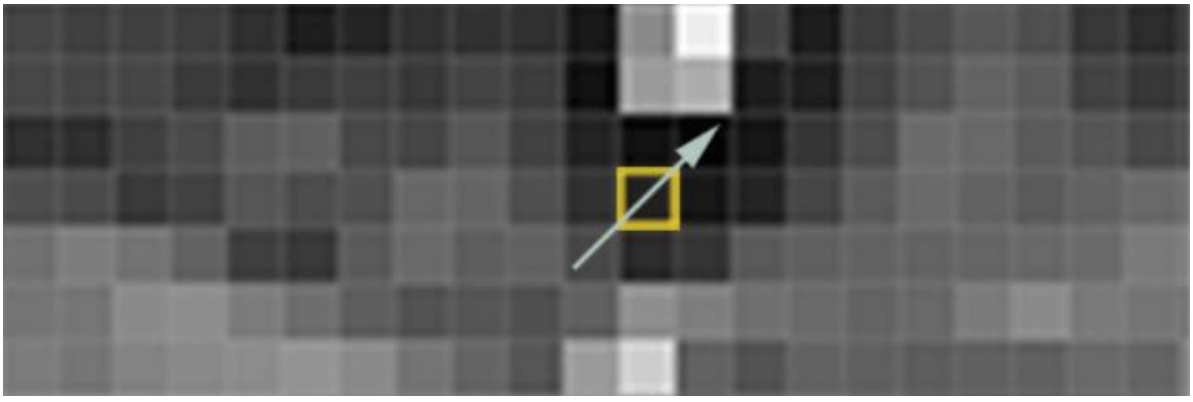
We're going to use a method invented in 2005 called Histogram of Oriented Gradients – HOG

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

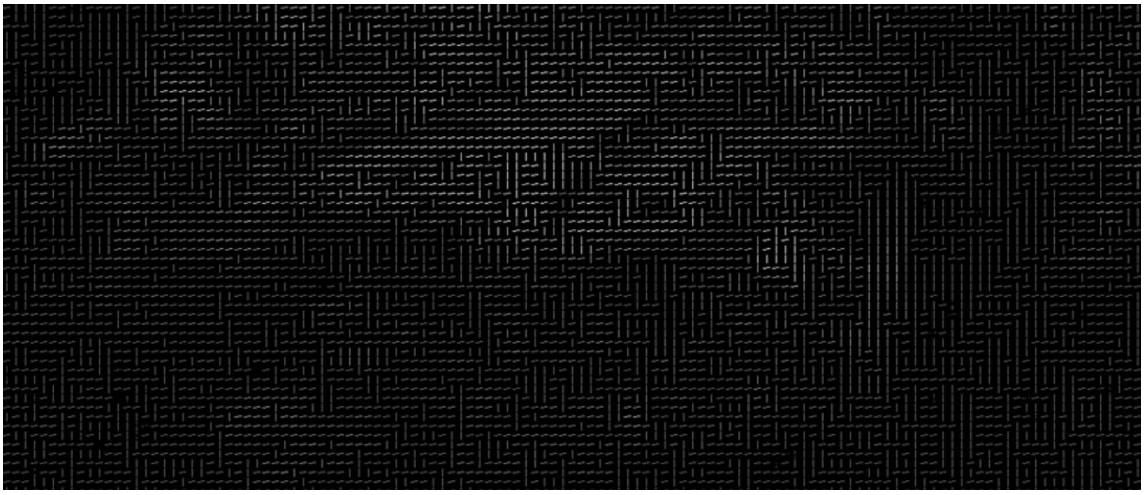
Then we'll look at every single pixel in our image one at a time. For every single pixel, we want to look at the pixels that directly surrounding it:



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:

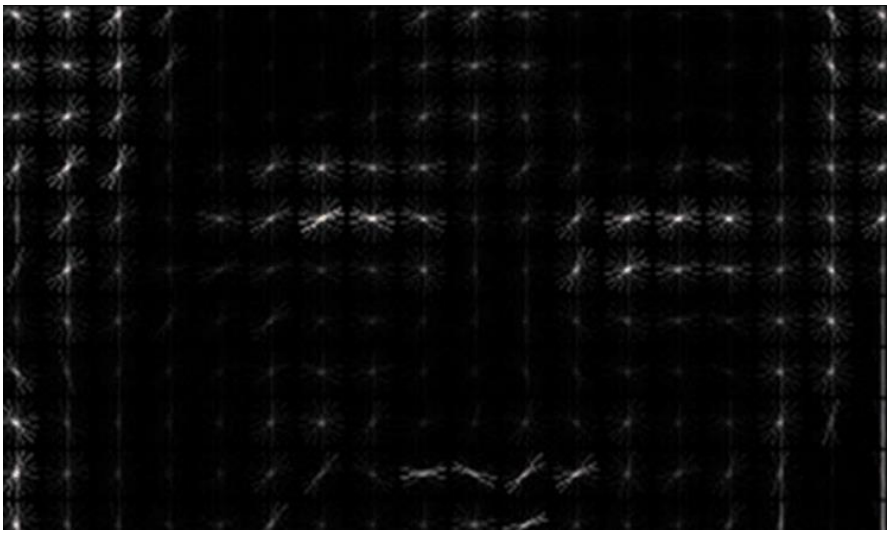


This might seem like a random thing to do, but there's a good reason for replacing the pixels with gradients. If we analyze pixels directly, dark images and light images of the same person will have totally different pixel values. But by only considering the direction that brightness changes, both dark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

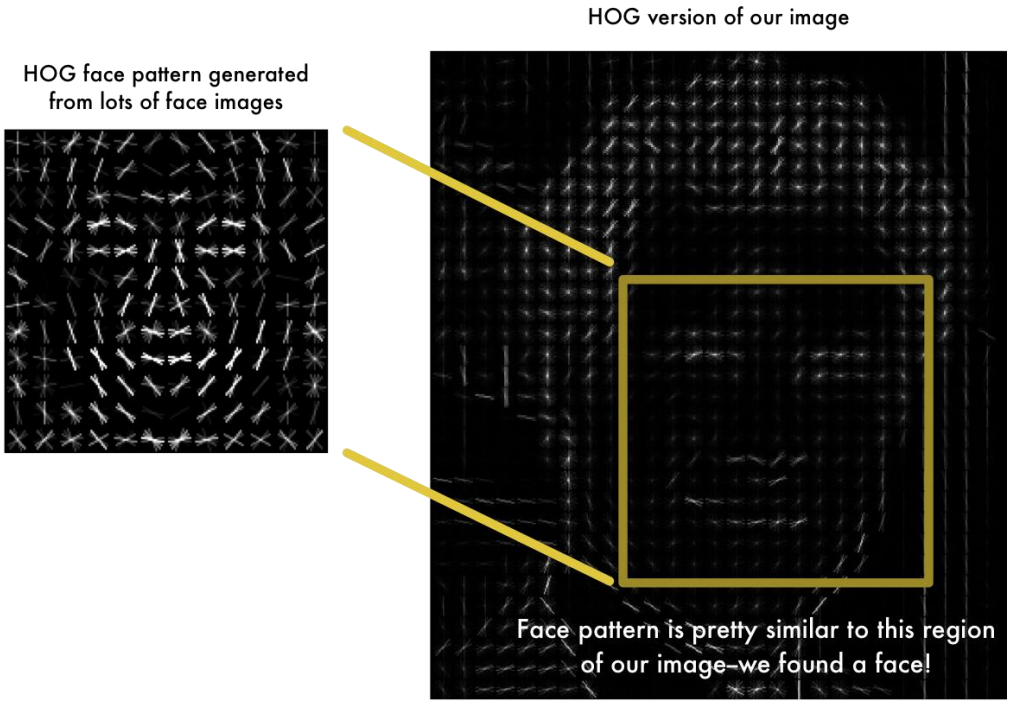
But saving the gradient for every single pixel gives us way too much detail. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

To do this, we'll break up the image into small squares of 16x16 pixels each. In each square, we'll count how many gradients point in each major direction (how many points 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 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 must do is find the part of our image that looks the most like a known HOG pattern that was extracted from a bunch of other training faces:

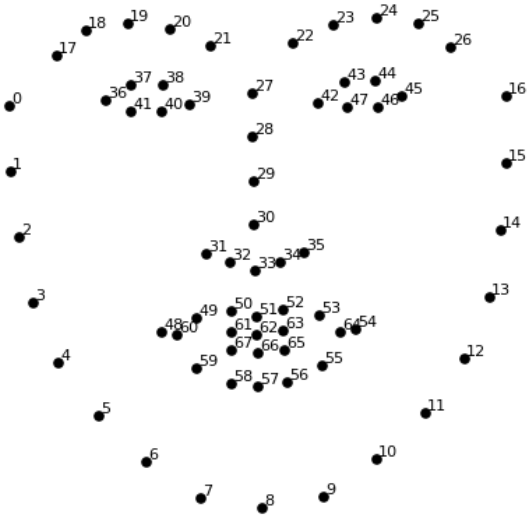


Problem2: Posing and Projecting Faces

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

To do this, we are going to use an algorithm called face landmark estimation. 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:



Problem 3: Encoding Faces

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 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.

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.

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:

Load a training face image of a known person

Load another picture of the same known person

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.

Machine learning people call the 128 measurements of each face an embedding. The idea of reducing complicated raw data like a picture into a list of computer-generated numbers comes up a lot in machine learning (especially in language translation). The exact approach for faces we are using was invented in 2015 by researchers at Google but many similar approaches exist.

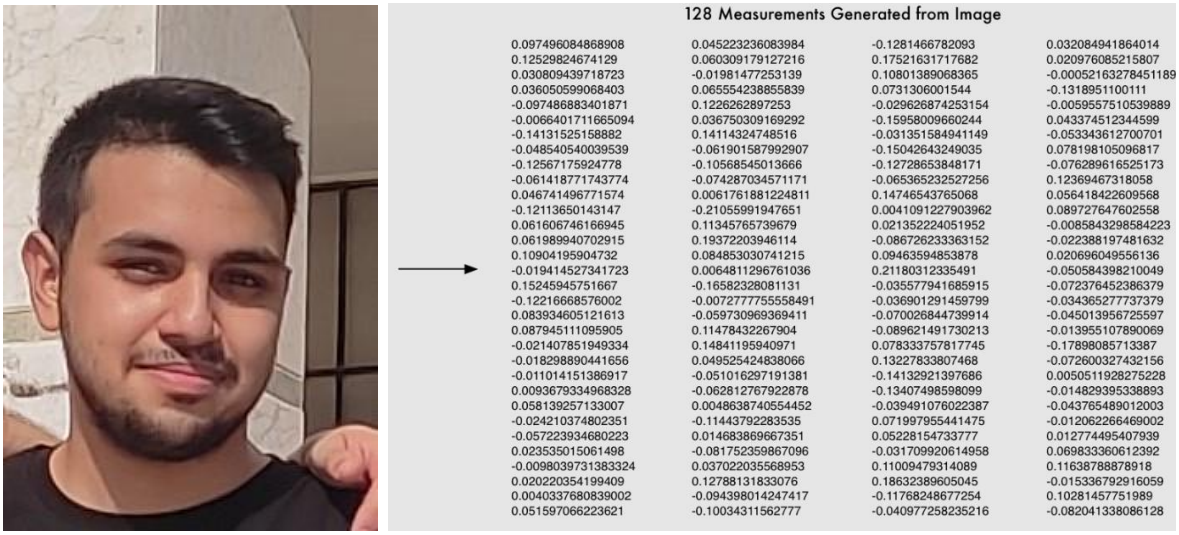
Encoding our face image

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.

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:

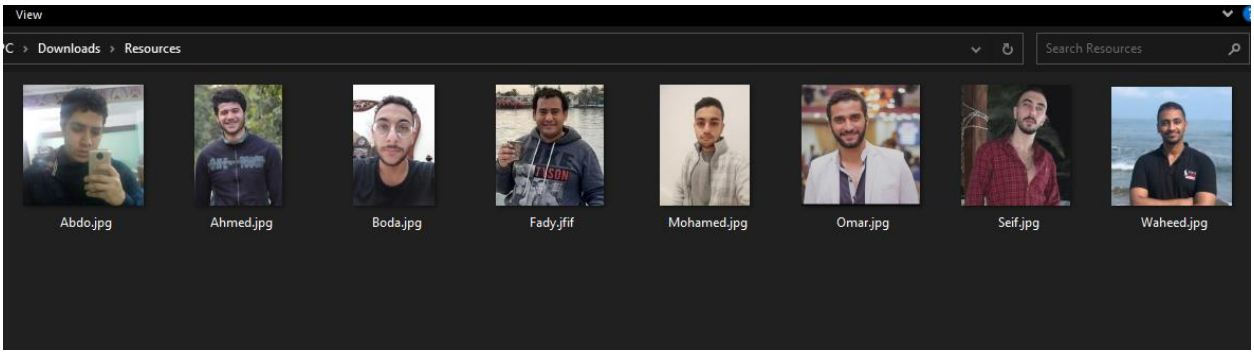


Problem 4: Finding the person's name from the encoding

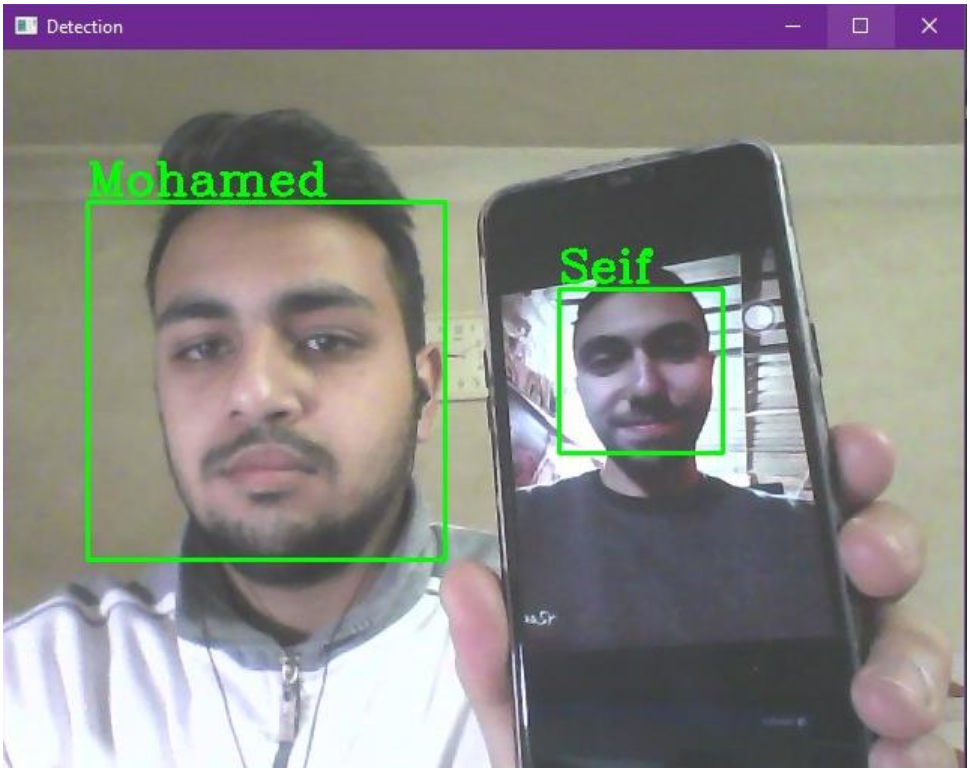
All we must do is find the person in our database of known people who has the closest measurements to our test image.

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!

So, let’s try out our system. First, I trained a classifier with the embeddings of our team.



Then I ran the classifier on every frame of the camera video



It works! And look how well it works for faces in different poses — even sideways faces!

Libraries I used:

OpenCV (Open-Source Computer Vision Library) is an open-source computer vision and machine learning software library.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

Dlib is a modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems. It is used in both industry and academia in a wide range of domains including robotics, embedded devices, mobile phones, and large high performance computing environments.

Face_Recognition :Recognize and manipulate faces from Python or from the command line with the world's simplest face recognition library.

Built using dlib's state-of-the-art face recognition built with deep learning. The model has an accuracy of 99.38% on the Labeled Faces in the Wild benchmark.

It is highly accurate, easy to use, easy to deploy on a server even without an expensive GPU and has rather simple requirements.

Why face_recognition library?

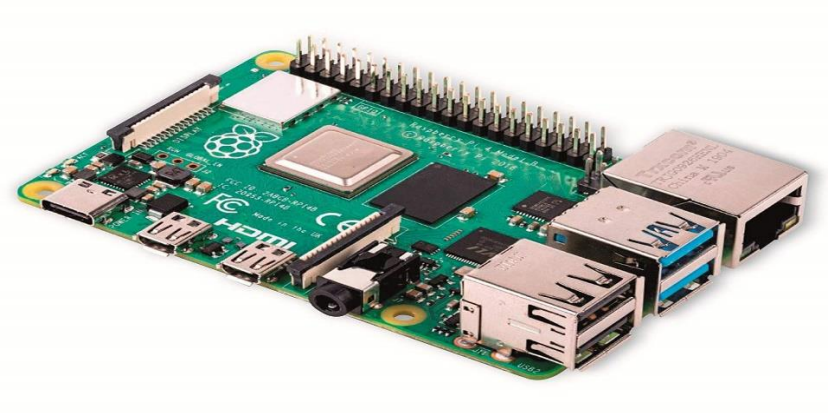
Easier to use than a generalized detection like YOLO and TensorFlow.

Takes much less computation than YOLO and TensorFlow.

We used a minicomputer like Raspberry Pi to run our code on.

What Is a Raspberry Pi?

It is a single-board computer, quite powerful for its small size. The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.



We used Raspberry Pi 4 with 4 GB ram, SD Card 64 GB, and raspberry pi camera v1.3.