Smart Attendance System Using Machine learning

Ankit Chotaliya and Dhainik Suthar

From Vishwakarma government engineering college (VGEC),

382424, chandkheda, Ahmedabad, Guj., India

Abstract

To maintain the attendance record with day-to-day activities is a challenging task. The conventional method of calling name of each student is time consuming and there is always a chance of proxy attendance. The following system is based on face recognition to maintain the attendance record of students. The daily attendance of students is recorded subject wise which is stored already by the administrator. As the time for corresponding subject arrives the system automatically starts taking snaps and then apply face detection and recognition technique to the given image and the recognize students are marked as present and their attendance update with corresponding time and recognize them. Our system is capable to identify multiple faces in real time. At the last attendance sheet is sent to corresponding sir via mail.

Keywords: *Machine learning, python, Image Processing, Face Recognition, OpenCV, Compare_faces, face_landmarks, HOG.*

Introduction

Face recognition has developed into a major research area in pattern recognition and computer vision. Face recognition is different from classical pattern-recognition problems such as character recognition. In classical pattern recognition. there are relatively few classes, and many samples per class. With many samples per class. algorithms can classify samples not previously seen by interpolating among the training samples.

A support vector machine (**SVM**) is a supervised machine learning model that uses classification algorithms for two-group classification problems. After giving an **SVM** model sets of labelled training data for each category, they're able to categorize new text.

Support vector machines (SVMs) are formulated to solve a classical two class pattern recognition problem. We adapt SVM to face recognition by modifying the interpretation of the output of a SVM classifier and devising a representation of facial images that is concordant with a two-class problem. Traditional SVM returns a binary value, the class of the object. To train our SVM algorithm, we formulate the problem in a difference space, which explicitly captures the dissimilarities between two facial images. This is a departure from traditional face space or view-based approaches, which encodes each facial image as a separate view of a face.

We demonstrate our SVM-based algorithm on both verification and identification applications. In identification, the algorithm is presented with an image of an unknown person. The algorithm reports its best estimate of the identity of an unknown person from a database of known individuals. In a more general response, the algorithm will report a list of the most similar individuals in the database. In verification (also referred to as authentication), the algorithm is presented with an image and a claimed identity of the person. The algorithm either accepts or rejects the claim. Or, the algorithm can return a confidence measure of the validity of the claim.

We compare video to all images in database than after if video matched with person than we mark that person's attendance.

Modal building

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.

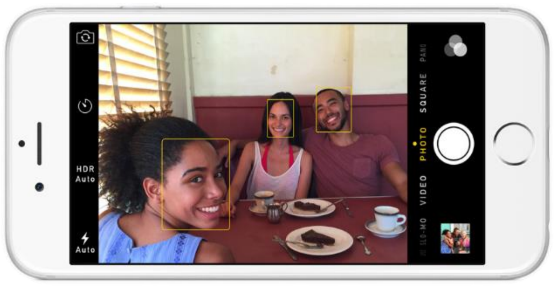
A. Import python modules

First of all, we import some of the python modules like NumPy, OpenCV, dlib, os and face recognition for mark attendance we use datetime module which one is prebuild with python version >=3.9

B. Finding all the 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!

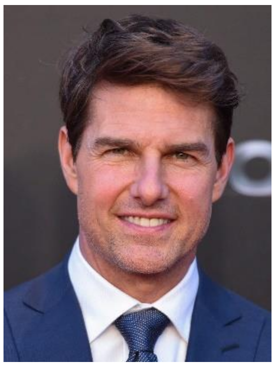
If you've used any camera in the last 10 years, you've probably seen face detection in action:



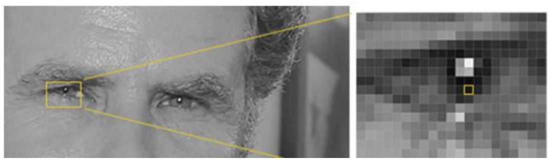
Face detection is a great feature for cameras. When the camera can automatically pick out faces, it can make sure that all the faces are in focus before it takes the picture. But we'll use it for a different purpose — finding the areas of the image we want to pass on to the next step in our pipeline.

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 colour 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.

This might seem like a random thing to do, but there's a really good reason for replacing the pixels with gradients. If we analyse 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!

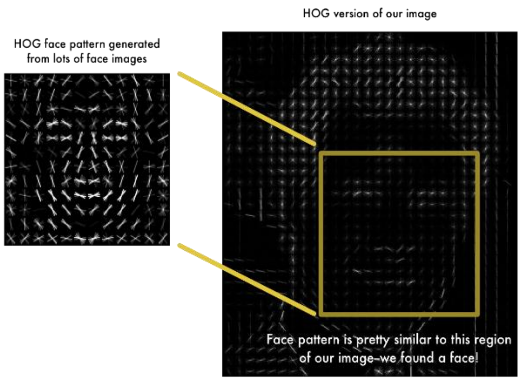
But saving the gradient for every single pixel gives us way too much detail. We end up missing the forest for the trees. It would be better if we could just see the basic flow of lightness/darkness at a higher level so we could see the basic pattern of the image.

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 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 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:

The original image is turned into a HOG representation that captures the major features of the image regardless of image brightness.

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:


HOG version of our image
HOG face pattern generated from lots of face images
Face pattern is pretty similar to this region of our image–we found a face!

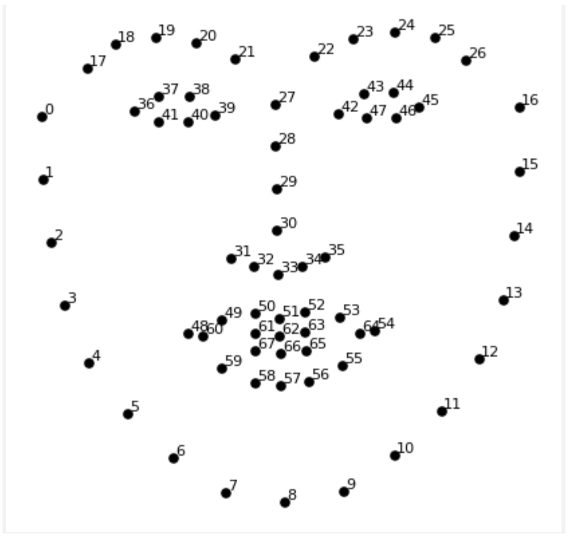
C. Posing and Projecting Faces

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:



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

The perfectly centered result we want

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

D. 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 colour? Something else?

It turns out that the measurements that seem obvious to us humans (like eye colour) 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 (just like we did in Part 3). 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 known person
2. Load another picture of the same known person
3. Load a picture of a totally different person

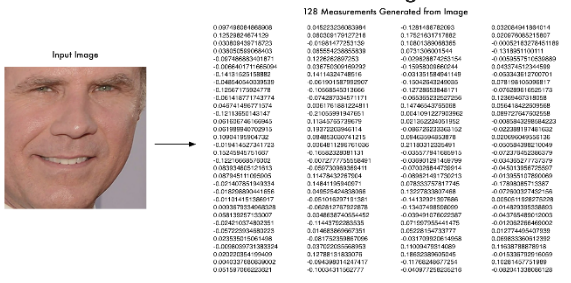
Machine learning people call the 128 measurements of each face an **embedding**. The idea of reducing complicated raw data like a picture into a list of computer-generated numbers comes up a lot in machine learning (especially in language translation). The exact approach for faces we are using was invented 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 Tesla 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 Open Face 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:



```
0.097496084868908     0.045223236083984     -0.1281466782093      0.032084941864014
0.12529824674129      0.060309179127216     0.17521631717682      0.020976085215807
0.030809439718723     -0.01981477253139     0.10801389068365      -0.00052163278451189
0.036050599068403     0.065554238855839     0.0731306001544       -0.1318951100111
-0.097486883401871    0.1226262897253       -0.029626874253154    -0.0059557510539889
-0.0066401711665094   0.036750309169292     -0.15958009660244     0.043374512344599
-0.14131525158882     0.14114324748516      -0.031351584941149    -0.053343612700701
-0.048540540039539    -0.061901587992907    -0.15042643249035     0.078198105096817
-0.12567175924778     -0.10568545013666     -0.12728653848171     -0.076289616525173
-0.061418771743774    -0.074287034571171    -0.065365232527256    0.12369467318058
0.046741496771574     0.0061761881224811    0.14746543765068      0.056418422609568
-0.12113650143147     -0.21055991947651     0.0041091227903962    0.089727647602558
0.061606746166945     0.11345765739679      0.021352224051952     -0.0085843298584223
0.061989940702915     0.19372203946114      -0.086726233363152    -0.022388197481632
0.10904195904732      0.084853030741215     0.09463594853878      0.020696049556136
-0.019414527341723    0.0064811296761036    0.21180312335491      -0.050584398210049
0.15245945751667      -0.16582328081131     -0.035577941685915    -0.072376452386379
-0.12216668576002     -0.0072777755558491   -0.036901291459799    -0.034365277737379
0.083934605121613     -0.059730969369411    -0.070026844739914    -0.045013956725597
0.087945111095905     0.11478432267904      -0.089621491730213    -0.013955107890069
-0.021407851949334    0.14841195940971      0.078333757817745     -0.17898085713387
-0.018298890441656    0.049525424838066     0.13227833807468      -0.072600327432156
-0.011014151386917    -0.051016297191381    -0.14132921397686     0.0050511928275228
0.0093679334968328    -0.062812767922878    -0.13407498598099     -0.014829395338893
0.058139257133007     0.0048638740554452    -0.039491076022387    -0.043765489012003
-0.024210374802351    -0.11443792283535     0.071997955441475     -0.012062266469002
-0.057223934680223    0.014683869667351     0.05228154733777      0.012774495407939
0.023535015061498     -0.081752359867096    -0.031709920614958    0.069833360612392
-0.0098039731383324   0.037022035568953     0.11009479314089      0.11638788878918
0.020220354199409     0.12788131833076      0.18632389605045      -0.015336792916059
0.0040337680839002    -0.094398014247417    -0.11768248677254     0.10281457751989
0.051597066223621     -0.10034311562777     -0.040977258235216    -0.082041338086128
```

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.

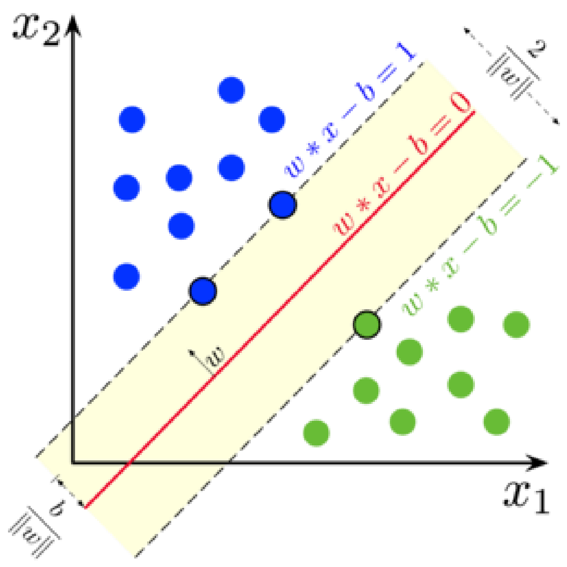
E. Finding the person’s name from the encoding

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.

You can do that by using any basic machine learning classification algorithm. No fancy deep learning tricks are needed. We’ll use a simple linear SVM classifier, but lots of classification algorithms could work.

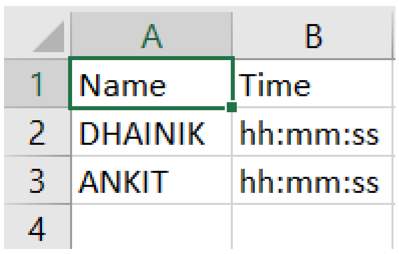
All we need to do is train a classifier that can take in the measurements from a new test image and tells which known person is the closest match. Running this classifier takes milliseconds. The result of the classifier is the name of the person!

SVM ANALYSIS GRAPH



F. Mark Attendance

If image is compared successfully than we write student name in excel sheet



	A	B
1	Name	Time
2	DHAINIK	hh:mm:ss
3	ANKIT	hh:mm:ss
4		

CONCLUSIONS

This work has presented a machine learning model that combines a image load with a face encodings for face recognition prediction and uses as input a Excel sheet and mark the attendance. The SVM architecture can model the local relation of a array encodings of face encodings and their encode list. Results presented in excel sheet show that above figure.

REFERENCES

[1] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[2]https://papers.nips.cc/paper/1998/file/a2cc63e065705fe938a4dda49092966f-Paper.pdf

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

[4] https://en.wikipedia.org/wiki/Support-vector_machine

[5]https://docs.opencv.org/master/d6/d00/tutorial_py_root.html

[6] https://face-recognition.readthedocs.io/en/latest/readme.html