Face Recognition System using Neural networks and SVM

Raghu Puppala

Abstract

Face recognition systems are one of the widely used recognition systems besides fingerprint recognition systems, speech recognition, and iris scanning systems. A face recognition system is a combination of face detection and faces recognition. Face detection involves recognizing all the faces in an image. Face recognition is about determining the name of the person from the face. This is called classification. Facebook uses a similar approach to detect a face and classify (tag with a person name) an image. In real-world face detection, many variations of faces due to lighting, pose, expressions and clutter in an image require a robust detection and recognition system. This project presents an integrated approach to build a complete face detection and recognition system. This is achieved by using Histogram of Gradients(HOG) for detecting faces in an image, face landmark estimation to align the faces, CNN for training features of faces and SVM for classification of face images. This project also aims at learning multiple open source softwares and how to integrate complex systems to create an end to end Face recognition system.

1. Introduction

Face recognition is probably one of the most prominent areas of research. It has a wide range of applications like access control, surveillance, identity authentication and on social networking sites. More and more social networking sites, mobile apps, services, AR applications are being built based on Face recognition systems at its core. During last two decades, automatic face recognition systems have seen considerable progress. Before 2010, these systems were mostly based on feature modeling of the face. Since 2010, deep neural networks have taken computer vision community by storm, significantly improving the state of the art in many applications. One of the main reasons for making it possible is training data and easy availability of processing power.

Face recognition system is really a series of several related problems:

1. Find all the faces in a picture
2. Focus on each face and be able to recognize it even under bad lighting, weird direction or not clear
3. Pick unique features of the face to distinguish from faces of other people.
4. Finally, compare those unique features and classify the image and determine the name of the person.

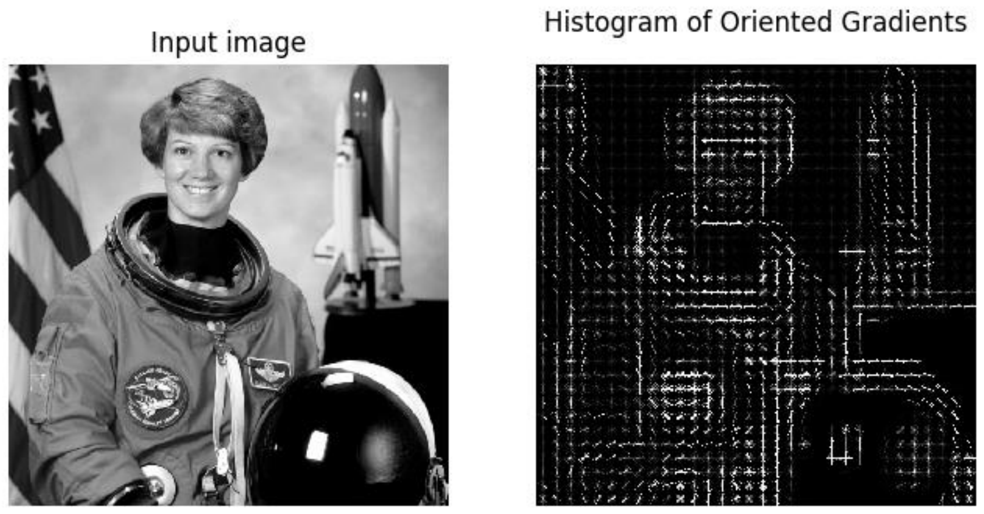
We take it for granted to recognize a face, as our brain is wired to do all of this automatically and instantly. Computers find it really difficult to learn unless we teach it how to learn each step in this process separately.

2. Method

2.1 Finding all the faces

Our first step in our pipeline is face detection. We need to locate faces in an image before we can classify it. Here we are interested in finding areas of images where faces are present. Initial methods developed for face recognition used individual features on the face like nose, eyes or mouth. Later on, new methods used global features of the face like Eigen faces that used Principal Component analysis(PCA). Other methods used Fisherfaces or laplacian faces to extract features from faces. Here we are going to use a method developed in 2005 called Histogram of Oriented Gradients or HOG.

To find the faces in the image, we will convert our images to grayscale as we do not need color information to detect faces in an image. Then we will look at every single pixel and also the pixels surrounding it. Our goal is to determine how dark is the pixel compared to its surrounding pixels. This will help us to draw the direction in which the image is getting darker. Then we want to draw an arrow in the same direction as the direction of pixels getting darker. Repeating this process for every single pixel, we will end up with every pixel replaced by an arrow. These arrows are called gradients and they indicate the flow from light to dark pixels across an entire image. We can understand the basic pattern of the image from the image as shown below.



To get such image, we'll break up the image into small squares of 16x16 pixels each and count number of gradients pointing in each major direction. We will replace that square in the image with the arrow directions that were the strongest. Doing this for the whole image gives 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

Here, the face detector is made using Histogram of Oriented Gradients (HOG) feature combined with a linear classifier, an image pyramid, and sliding window detection scheme. It can be understood as an object detector which is capable of detecting human faces. We run a sliding detection window around the image pyramid. At each position of the detector window, a HOG descriptor is computed for the window. This descriptor is given to a trained classifier to determine if it is a face or not a face [1]. I have used dlib library and modified some parameters to create face detector and based on a paper - HOG for human detection by Navneet Dala and Bill Triggs. Some of the parameters that are used in HOG scheme to build this face detector are:

Cell size = 8

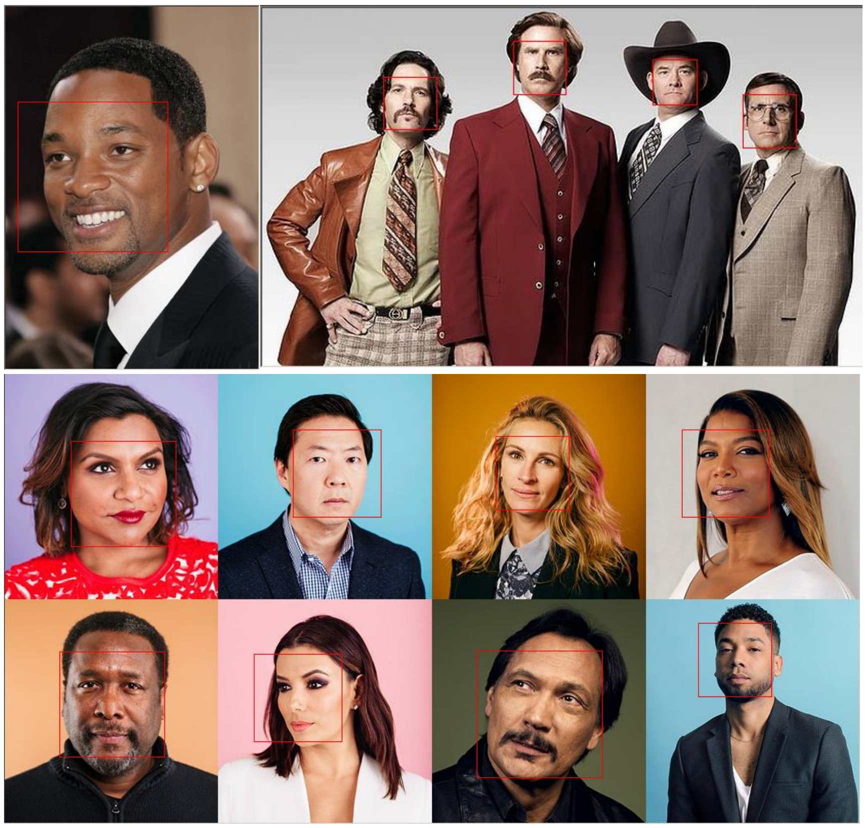
Detection window = 64 x 64

Max pyramid levels = 1000

Min pyramid layer width = 64

Min pyramid layer height = 64

Here is the result of finding faces on few images:

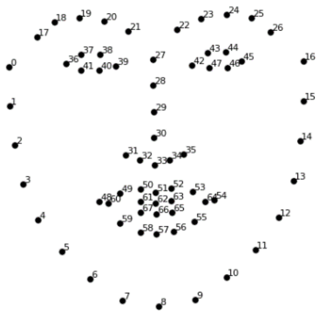


2.2 **Posing and Projecting Faces**

To deal with faces that have different orientations and hence look different, we will try to warp each picture so that the eyes and lips are always in the same place in the image. This helps us to simplify the problem of comparing faces in the next step.

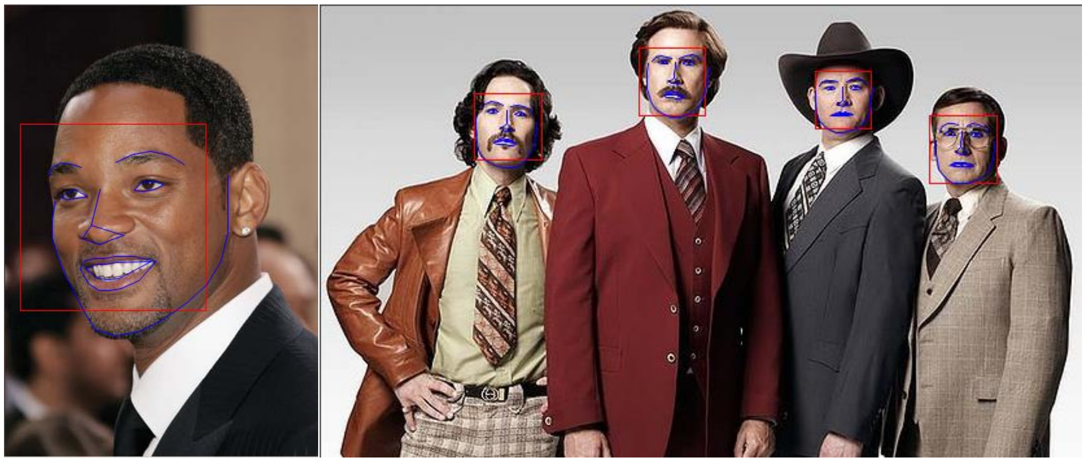
There are different ways to do this. But, we are going to use face landmark technique first developed in 2014 by Vahid Kazemi and Josephine Sullivan. According to them, we can represent a face with 68 specific points called landmarks- like chin, edges of eyes, the inner edge of the eyebrow, etc. This paper shows how an ensemble of regression trees can be used to estimate the face's landmark positions directly from sparse subset of pixel intensities via gradient boosting algorithm with high accuracy. So, we can train a machine learning model to be able to find these 68 landmarks on any face.

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 is shown below:



This image was created by Brandon Amos of CMU who works on OpenFace project. Here, I'm going to use dlib's implementation of the paper by Kazemi et al. They trained a face annotation model on iBUG 300-W face landmark dataset. Using this model, we can annotate our face images and then can center the image as best as possible using basic image transformations.

Results after landmark detection on detected faces are shown below. Blue color is used for landmarks and red for face detection box in the following image:



2.3 **Encoding Faces**

This step deals with recognizing faces and actually telling faces apart. We can train a deep Convolutional Neural Network to generate 128 measurements for each face.

This can be done 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

Then the algorithm looks at the measurements it is currently generating for each of those three images. It then tweaks the neural network slightly so that it makes sure the measurements it generates for #1 and #2 are slightly closer while making sure the measurements for #2 and #3 are slightly further apart. After repeating this step multiple images of different people, the network learns to generate 128 measurements for each person. These 128 measurements of each face are called embedding.

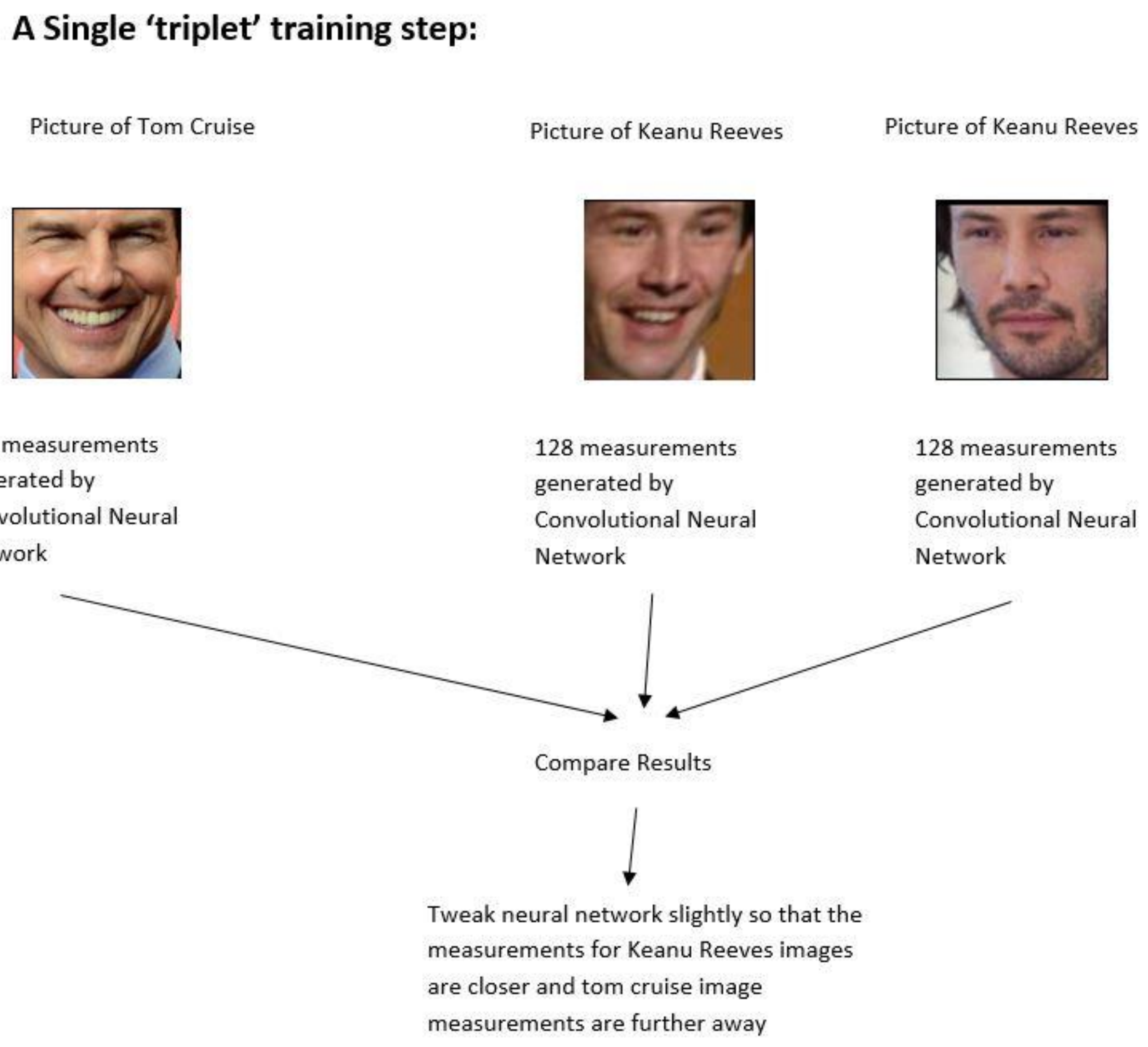
This is based on a paper- FaceNet[4] by Shroff et al used for creating a FaceNet embedding of 128 bytes for each class. According to this paper, FaceNet directly learns a mapping from face images to a compact Euclidean space where distances correspond to a measure of face similarity. This method uses deep convolutional network and it is trained to directly optimize the embedding itself. They use a triplet loss function based on Large Margin Nearest Neighbor(LMNN).

Large Margin Nearest Neighbor (LMNN) is learning a pseudo-metric

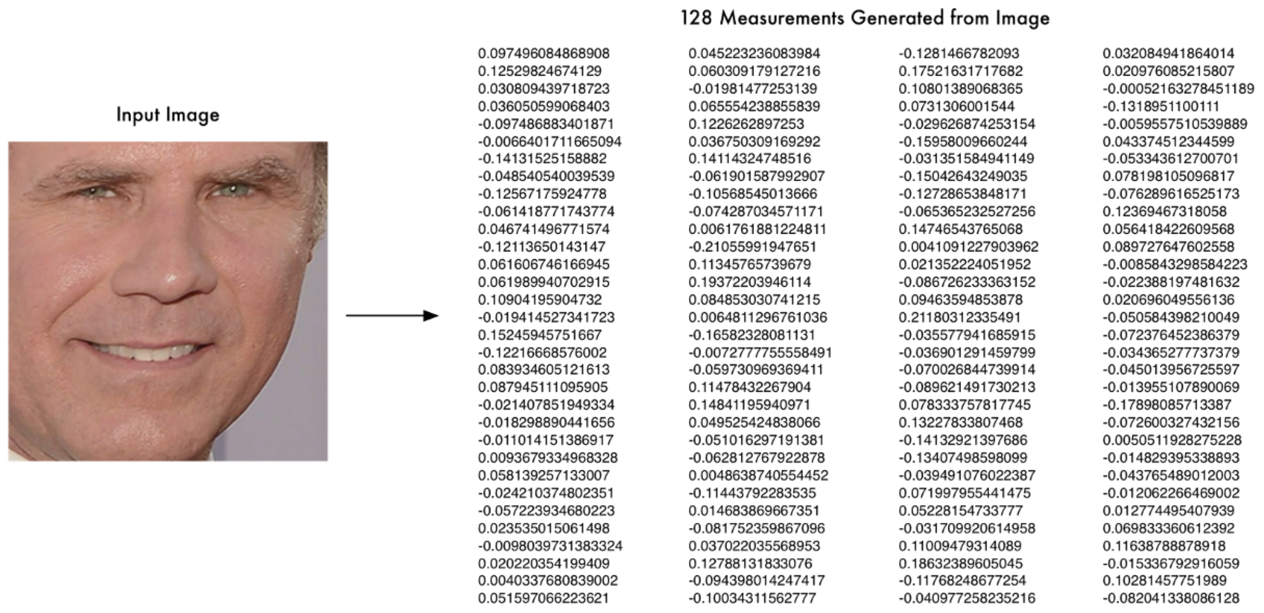
$$d(x,y)=(x-y)M(x-y)T$$

where M is a positive-definite matrix. The only difference between a pseudo-metric and a metric is that d(x,y) = 0 for x=y does not hold.

Here, triplet is (face of person A, other face of person A, face of person which is not A). so, triplet consists of two matching face thumbnails and a non-matching thumbnail and the loss aims to separate the positive pair from the negative pair by a distance margin. The thumbnails are tight crop area of face images without 2D or 3Dalignment except for scaling and translation.


A Single 'triplet' training step:
Picture of Tom Cruise
Picture of Keanu Reeves
Picture of Keanu Reeves
128 measurements generated by Convolutional Neural Network
128 measurements generated by Convolutional Neural Network
128 measurements generated by Convolutional Neural Network
Compare Results
Tweak neural network slightly so that the measurements for Keanu Reeves images are closer and tom cruise image measurements are further away

Training a CNN which outputs face embeddings for each face requires lot of computing power. FaceNet's CNN was trained for around 1000 to 2000 hours on a CPU cluster. But once the network is trained, it can generate embeddings for any face. OpenFace[6] team have trained several networks based on FaceNets acrchitecture. We can use this model to generate face embeddings(128 measurements) for each face image. A sample embedding for a face image looks like shown below:



128 Measurements Generated from 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

We have no idea what parts of the face are these 128 measurements. Actually, it does not matter. The network generates nearly same measurements for images of same person and distinct measurements for images of different person.

2.4 **Finding the person's name from the encoding**

This is the simplest of all the steps in face recognition system. We have to just find the person in our database of known people that matches the 128 measurements of our test image. This can be done by any basic machine learning algorithm like linear Support Vector Classification or logistic regression. We can train a classifier that can take measurements from a new test image and tell us which known person is the closest match.

Here we are going to use Support Vector classification to classify a new face image by comparing embedding of test image against our trained dataset. Our trained classifier model helps to classify by comparing the embeddings. Open source machine learning library Scikit Learn has SVC package to implement this classification part. I have used this package to classify the face embedding as one of the classes from our training data. The implementation is based on libSVM. The kernel used here is linear as the number of features(n) are greater than number of classes(m), n>m. if n<m then Gaussian kernel would be a better option.

3. **Execution and Findings**

To put it in simple terms we can implement a face recognition as follows:

- Encode a picture using the HOG algorithm to create a simplified version of the image. Using this simplified image, find the part of the image that most looks like a generic HOG encoding of a face.

- Figure out the pose of the face by finding the main landmarks in the face. Once we find those landmarks, use them to warp the image so that the eyes and mouth are centered.
- Pass the centered face image through a neural network that knows how to measure features of the face. Save those 128 measurements.
- Looking at all the faces we've measured in the past, see which person has the closest measurements to our face's measurements. That's our match!

3.1 **Face detection and alignment**

I took a small subset of LFW dataset consisting of face images of 50 famous personalities (50 classes). Each class has at least 5 images. This is our training data. We make a subfolder for each person we want to recognize. This is saved in ./training-images folder. We run face detection and alignment algorithm on training data set to create a cropped and aligned version of each of our test images. This is made possible by OpenFace package. All the aligned images are stored in ./aligned-images. This was able to detect multiple faces in an image but it was not so good to detect faces with occlusions. So, I had to exclude some images which had occlusions. The aligning step was very quick as it was basic image transformations. All the cropped images are cropped properly without missing any part of the face.

3.2 Generate Embeddings

We are going to use **a** trained CNN model developed by OpenFace project team based on FaceNet architecture. This helps in avoiding spending too many hours for training a CNN on face images to generate an embedding for a face image. So we can use this trained model to generate an embedding. After this step we will have an embedding for each face image. This can be used to build a classification model which can classify based on distance between the embeddings. Here an embedding is a vector of length 128. All the embeddings in a csv file located in ./generated-embeddings sub-folder.

3.3 Training

Here I used a linear Support Vector classification model using scikit learn package. Here I used a linear SVM as I have 50 classes(m) and 128 features(n). In this case it is better to go for linear model. Linear and Gaussian kernels are the most used kernels. If we are going to add additional classes with additional images per class, such that m>n we can get better model and better results by using Gaussian kernel.

3.4 Recognize new images(testing)

We can try new different images of the persons in our training data. The process followed here is the same as the one which we followed for creating a classification model. We will run the face detection and alignment algorithm from step 3.1 on the test image to detect all the faces present in the image. Embeddings for all the faces generated and compared with passed to the classifier. It basically computes the probability of possibility of outcome of each face. This is the prediction for the test image. The system outputs the name of the person predicted.

4. Conclusion

We created a simple end to end face recognition system which is pretty simple and fast to implement and can be run on a personal computer with CPU. I have used multiple open source softwares like libSVM, Scikit-learn, OpenFace and dlib libraries. After spending good amount of time on windows machine, I realized it is better to implement such projects on Linux machines. So, I implemented this on Ubuntu machine which made the job simple and fast.

This system will always make a prediction even if the face of the person is not present in the training dataset. In real application we would look at the confidence score and throw away predictions with poor confidence as they would most likely be wrong. The main problem in this type of systems design is the test face may have difference in view point or scale or illumination, when compared with trained faces, these factors will affect the recognition accuracy. To improve the recognition accuracy one way is train the database with more number of faces with different viewpoints, different scales and with different illumination levels. This will lead to large database size. This system is based on very small training set and the system got good prediction rate if more than 10 images of each person are used. So basically, the system would improve a lot by increasing the dataset. Also, the model worked well for faces with different poses and illumination. But, it failed badly in cases of occlusions and faces with hats. So, I excluded images which had hats or turbans. The same model can be used to classify faces in a video.

Future work will focus on better understanding of error cases, further improving the model, and also reducing the mode size and reducing the feature size of embedding if possible.

5. References

1. Histograms of Oriented Gradients for Human Detection Navneet Dalal and Bill Triggs INRIA Rhone-Alps, ˆ 655 avenue de l'Europe, Montbonnot 38334, France [2005]
2. Deep Face Recognition Omkar M. Parkhi, Andrea Vedaldi vedaldi, Andrew Zisserman
3. One Millisecond Face Alignment with an Ensemble of Regression Trees Vahid Kazemi and Josephine Sullivan KTH, Teknikringen.
4. FaceNet: A Unified Embedding for Face Recognition and Clustering Florian Schroff, Dmitry Kalenichenko, James Philbin [2015]
5. Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78
6. OpenFace. Free and open source face recognition with deep neural networks. https://cmusatyalab.github.io/openface/
7. Machine Learning is Fun! Part 4: Modern Face Recognition with Deep Learning by Adam Geitgey. https://medium.com/@ageitgey/machine-learning-is-fun-part-4-modern-face-recognition-with-deep-learning-c3cffc121d78