FACE RECOGNITION SYSTEM

# problem statement

Build face recognition model which gives label (name of person) and rectangle around face detected as output when given path of image (containing single face) of a person as input.

# methodology/pipeline for face recognition

Building a face recognition model comprises of following steps:

## face detection and alignment

Given an image (containing face) with label as input data, first task is to convert Image into matrix form (here we converted to numpy array) to detect faces (if needed face landmarks also) in the image.

## face extraction

Using coordinates (position) of face detected in image, face is then extracted from image and resized to fixed size (here (160,160)).

## feature embedding/extraction

Faces extracted (an array) are then used as input by feature embedding model (CNNs) (here Facenet), which learns from similarities (inverse of distance in Euclidean space) and differences of faces to extract most important features of a face and embed it to some dimensional vector or Euclidean space. Output vector is called embedding. Embeddings are similar for faces of same person but have significant differences for faces of different person. Various methods are available for feature embedding which extracts different combination of features.

## feature classification

features are then classified into different classes using different approaches which depends on the classifier used. Faces can be classified based on embedding distances (similarities) in Euclidean space which is computationally expensive or by remembering boundaries between people as they are already classified or training classifier like SVM to create linear or non-linear boundaries to divide data into classes.

## feature matching/verification (for test data)

Final task is to recognize or find label of input unlabeled test image by matching its feature embeddings with feature embeddings of available data. This is done by classifier which puts the test image into one of the classes to which its feature matches the most. Labels can also be encoded to numbers for performing easier computations.

# methods available

Based on pipeline, various methods available for different steps are provided here:

## face detection and alignment

* **Haar cascade** can be used for real time face detection but can detect only frontal images.
* **Dlib (HOG)** can work slightly on non-frontal images and is fastest method on CPU. It can’t detect small images and can’t handle occlusions.
* **Dlib (CNN)** can work on different face orientations. It is very fast on GPU but very slow on CPU. It can’t detect small faces in image.
* **MTCNN** can also work on various face orientations in image. It detects most accurate, can detect faces at different scales and can handle occlusions. It is slower than other methods.

## feature embedding

There are various methods and pretrained models available for Feature embedding. Some of well-known and accurate methods are compared and discussed below:

* **VGG** uses various architectures which have good accuracy. All of VGG’s hidden layers uses RelU as activation function due to which the decision function is more discriminative. It uses very small receptive fields (3\*3) and convolutional filters (1\*1) which leads to improved performance.
* **Face Recognition API** have easier architecture to implement with some inbuilt libraries required for face recognition and is easier to use. It have good accuracy and works well for real time systems. Today, many facial recognition API’s are available.
* **FaceNet** model uses One shot learning flow which aims to learn about object categories from few training images. It is a deep convolutional network and very large comprehensive dataset is used for training. It gives good accuracy even in scarcity of datasets. Generally, pretrained model is used and not trained from scratch. MTCNN’s implementations are based on FaceNet.

## feature classification and matching

Some well-known methods available for feature classification are:

* **Cosine Similarity** gives the similarity between two vectors as the measure of cosine of angle between two vectors. It gives false results for sparse numeric data.
* **Euclidean distance** is distance-based feature classification method which calculates distance between feature occupying specific location in Euclidean space (represented by embeddings). Similarity is inverse measure of distance. It is suitable for lower dimensional datasets having small number of classes.
* **KNN (K Nearest neighbor)**- It is checked that for a given feature in Euclidean space, out of k nearest neighbors majority belongs to which class. KNN classifier has dimensionality problem.
* **SVM (Support Vector Machine)** gives good performance when number of features is large and number of samples are not very large. It is a robust model for prediction problems. Wrong choice of kernel can lead to large error in prediction.

# methods used

## mtcnn

MTCNN (multi-task cascaded convolutional neural network) is a python library written by github user ipacz. It consists of three stages or layers of CNN namely Proposal network, Refinement network and Output network. MTCNN is used for face detection as it gives most accurate predictions for various facial orientations of image (i.e. good prediction for non-frontal images and for various scales (i.e. small face in large image). Drawback is it may take longer time than other methods for face detection.

## facenet

FaceNet developed in 2015 by Google researchers Schroff et al is a deep neural network. It initially randomly allocates embedding which is location of image in 128D euclidean space. Then it randomly selects anchor image, positive example (image of same person as in anchor image) and negative example (image of different person). Adjusts the network parameters and correspondingly embedding such that positive example is closer to anchor image than negative example. This process is iteratively done until faces of same person are close to each other and far from faces of any other person to learn various facenet parameters. This type of loss is called Triplet loss**.**

## svm classifier with a linear kernel

It creates linear hyperplanes with some margin to divide features into different clusters. Main focus is to maximize margin using nearest points to hyperplane. Final task is to recognize or find label of input unlabeled test image by matching its feature embeddings with those of available data. This is done by SVM classifier which puts the test image into the class to which its feature matches the most.

# results

Accuracy of model for our Bollywood celebrities is:

* Training dataset: 94%
* Test dataset: 91%

# conclusion

Facial recognition is evolving field and is being explored widely recently for various uses like augmented reality, cashless payments, face tagging and security in mobiles. Facial recognition have various real life applications in security and defense, retail marketing, healthcare hospitality and various other sectors. Although the code can recognize faces in image well but there are some possible improvements which can be done in model as follows:

* It can be extended to detect more than one face in a image.
* Tensor libraries and operations can be used instead of iterative loops and numpy arrays (which were used due to limitation of coding knowledge in tensor operations) which could make recognition faster especially in GPU.
* OpenCV module can be used with given model and improvements for real time face detection using webcam.
* When data is very large or very different then transfer learning can be used which is the training the weights of the top layers of the pre-trained model alongside the training of the classifier nodes added for training weights to features associated with our dataset.