# Introduction

“Face is the mirror of the mind, and the eyes gateway to the soul.” Faces in addition to reflecting what is going on in the heads and hearts also serve a primal function for living beings - differentiating between an acquaintance and a stranger. Faces create the first impression and they get registered in our brain. More often than not, a living being would recognize if they are seeing a face for the second time. This creates the basis of social cohesion, especially among humans.

Living beings can identify whether they have seen a face before or not. Most likely they would also remember the names and other details if they have been given that information. Wouldn’t it be cool, if machines could also learn to do the same? In fact, the machines can be trained to do these tasks very well. In this blog, we would learn about the methods and algorithms which power such techniques which are termed face detection and recognition.

Some of the popular applications and use cases of face detection and recognition are as follows:

1. Smartphone and personal computer access control
2. Entry access control to a premise to only a selected people
3. People based photo categorization e.g. in Google Photos
4. Automated name identification / tagging of people in photos e.g. on Facebook

# Face Verification vs Recognition

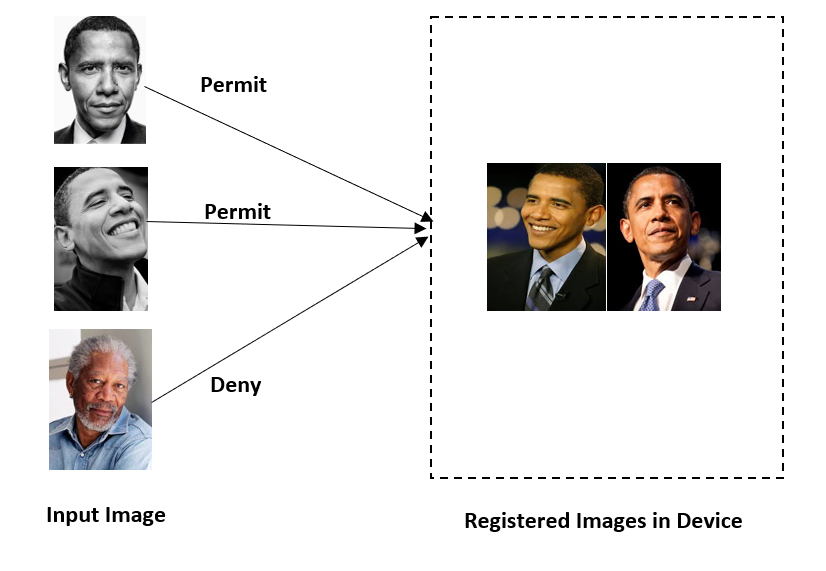
Although not obvious at the face of it, there are a lot of differences between face detection and recognition.

## Face Verification

Face Verification encompasses identification of the same face every time. A face is registered at the beginning in the device. If the face of the same person as the one registered is shown to the device, the face detection algorithm returns a positive detection. If the face of another person is shown, the face detection algorithm should ideally return a negative detection. Due to this property, face detection is primarily used as a more convenient alternative to device/app passwords. This ensures that only the owner of the device (whose picture is registered initially) can access the device and sensitive information.

The noticeable characteristics of face detection are:

1. 1)Registered image(s) of one person
2. Output is a positive or negative detection to a shown image and thus permitting or denying access
3. In ML terms, it is framed as a supervised binary classification problem



**Fig.1: Scope of Face Verification**

## Face Recognition

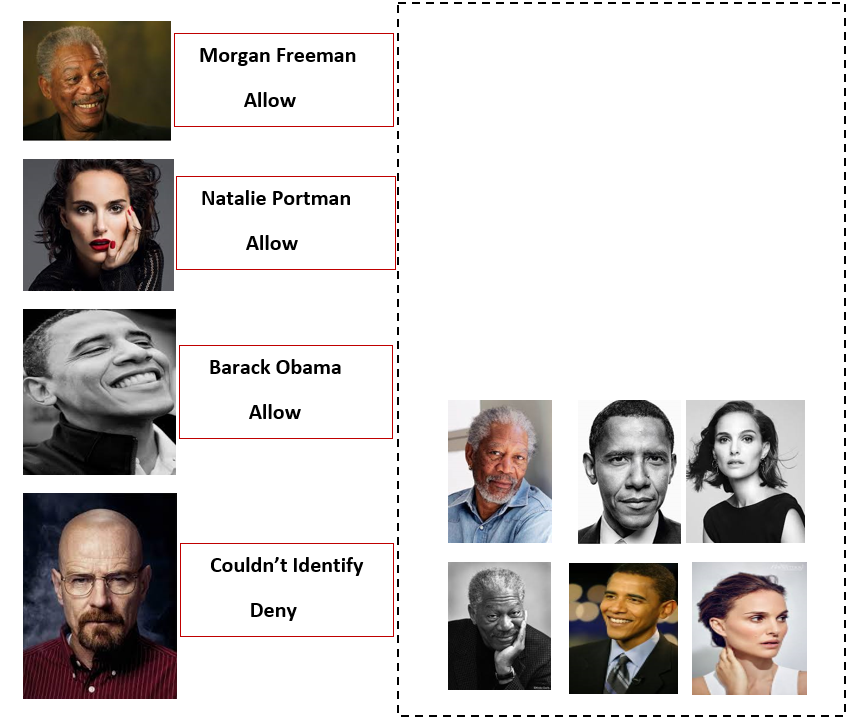
Face recognition, on the other hand, deals with identifying a person’s identity and access from a group of people. A set of people are registered in the device with their image(s). When a new input image is shown, the face recognition algorithm identifies which person is this from among the registered people and shows their details. If the shown image is not of one of the registered persons, then algorithms should deny access.

If an input image contains objects other than faces, then this task is performed in two steps:

1. Identify the Regions of Interest with faces and draw bounding boxes around them
2. Recognise the name of the person

The noticeable characteristics of face detection are:

1. Registered image(s) of multiple people
2. Output is identification of the person from set of registered people
3. In ML terms, it is framed as a multi-class unsupervised algorithm



**Fig.2: Scope of Face Recognition**

# Image Processing Basics **-Sandhya**

## Representing images numerically - Image as an array of pixels

Any object detection or image processing algorithm performed on an image requires it to be represented in a numerical array format. For text data, term-document matrix and term frequency-inverse document frequency (TF-IDF) are used to vectorise (create numerical arrays) the data. In case of an image, pixel values are used to represent an image.

For a 100X50 pixel RGB image, there would be

i) 5000 pixel values in 1 channel

ii) 3 channels each for –red, blue and green.

Hence, if you flatten the image pixels as one single vector, its length would be 15,000 (5,000 for each of the 3 channels). A grayscale image would contain a single channel. Each pixel value represents the degree of brightness for each channel.

A dataset of multiple images then becomes a 4-dimensional data representing:

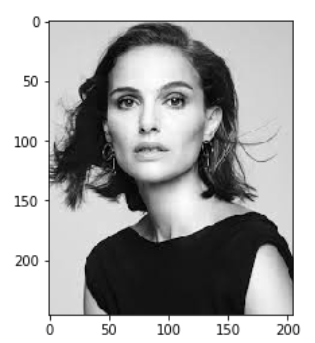
i) Height of image in pixels

ii) Width of image in pixels

iii) Number of channels

iv) Serial Number of image

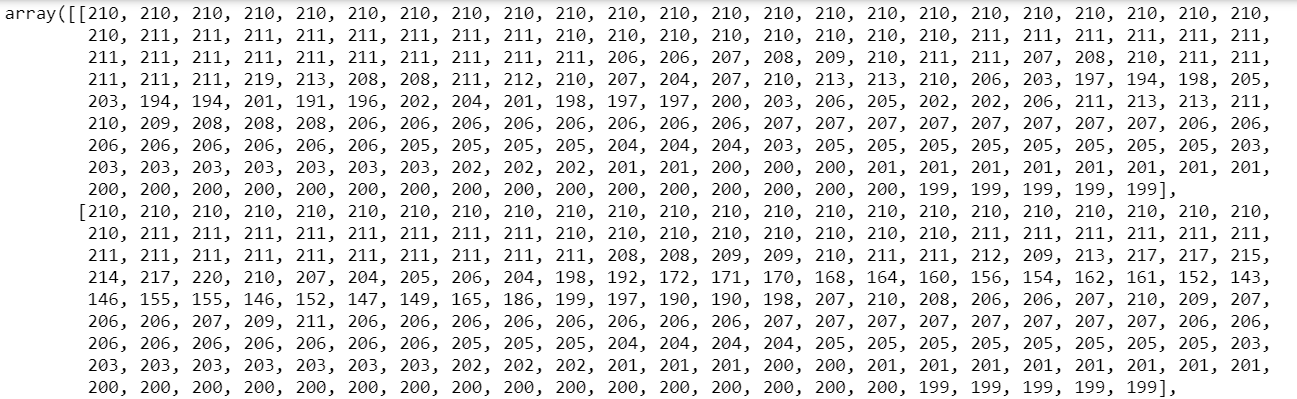
Now we will see how an image array actually looks. This can be easily done using openCV package in Python.



**Fig.3: Image after being read and rendered in openCV**

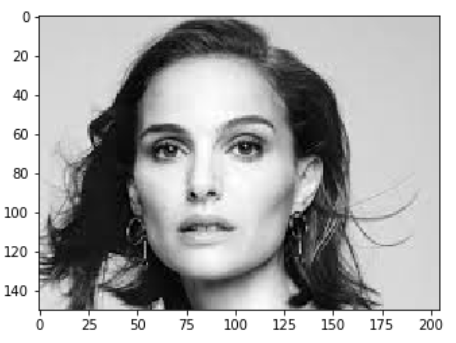
Notice how the two axes ticks denoting the pixel units. The y-axis represents the height, x-axis the width. The exact shape of this image array is (246, 205, 3) meaning height of this image is 246 pixels, width 205 pixels and 3 channels representing Red, Green and Blue channels.

The image below shows the first two pixel units in y-direction (height). Remember, there are 246 such pixel units. Each of the two shows the pixel values (205 values) for the entire width in that pixel unit. Quick exercise, count the number of values inside the square brackets, it should be 205.



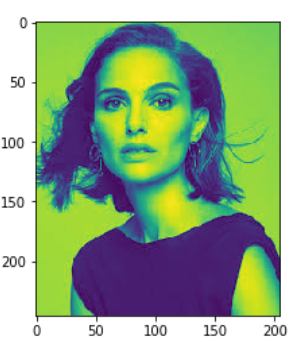
**Fig.4: Image as a matrix of pixels; showing 2 pixel units in y direction**

Once the image is accessed as a matrix, all the mathematical operators can be applied to it. The first and foremost is subsetting i.e. looking at only part of the image. This is useful for cropping of images. For example, if we want to crop only the face, we can filter only 150-160 pixel units in y-direction and all the pixel units in x-direction.



**Fig.5: Cropping only the face by subsetting the first 150 pixel units in y direction**

Another example of subsetting would be to look at only one channel (of Red, Blue or Green) of the image.



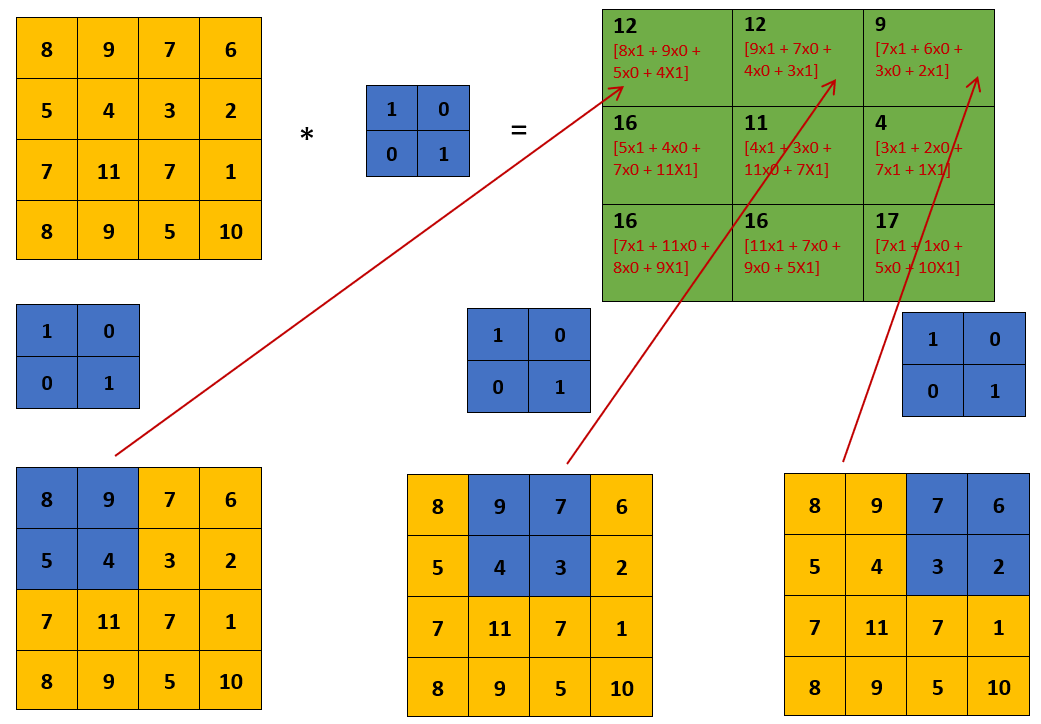
**Fig.6: Subsetting only one channel for the entire image**

## Filters and Convolution - Feature Engineering on images

Convolution is a mathematical operator which is applied on images to detect important features and reduce the number of training parameters.

Before delving deeper, it is imperative to understand the mathematical definition of convolution. It involves an image matrix and a convolution filter.

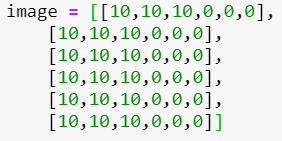
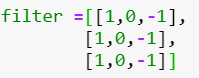
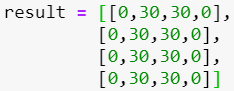
1. Filter is superimposed on the first chunk of the image matrix of the same size as the filter.
2. The pair of numbers from the superimposed portions are multiplied to each other
3. The products are then summed to produce one number
4. Filter is moved one step and to find another chunk of the same size. Steps a, b and c are repeated on this chunk.

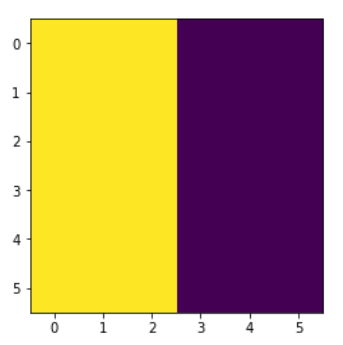
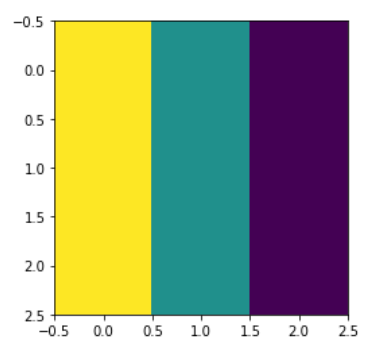
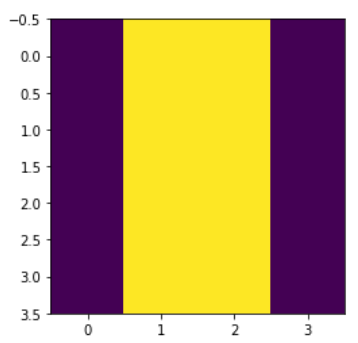


**Fig.7: Explanation of convolution operator**

Note that this example is shown for a convolution with strides of 1 unit i.e. the filter moves only 1 unit in x and y directions to find the next overlap. The strides can be defined to be more than 1. A high stride decreases the size of the output much more than a convolution with single stride. This is generally done when it is intended to reduce the size of output drastically in a single step.

Intuition for convolution - represent a large filter-sized area by the weighted average of the elements. This might lead to discovery of a local representational pattern and feature.

**Fig.8: Visual representation of effect of convolution on images**

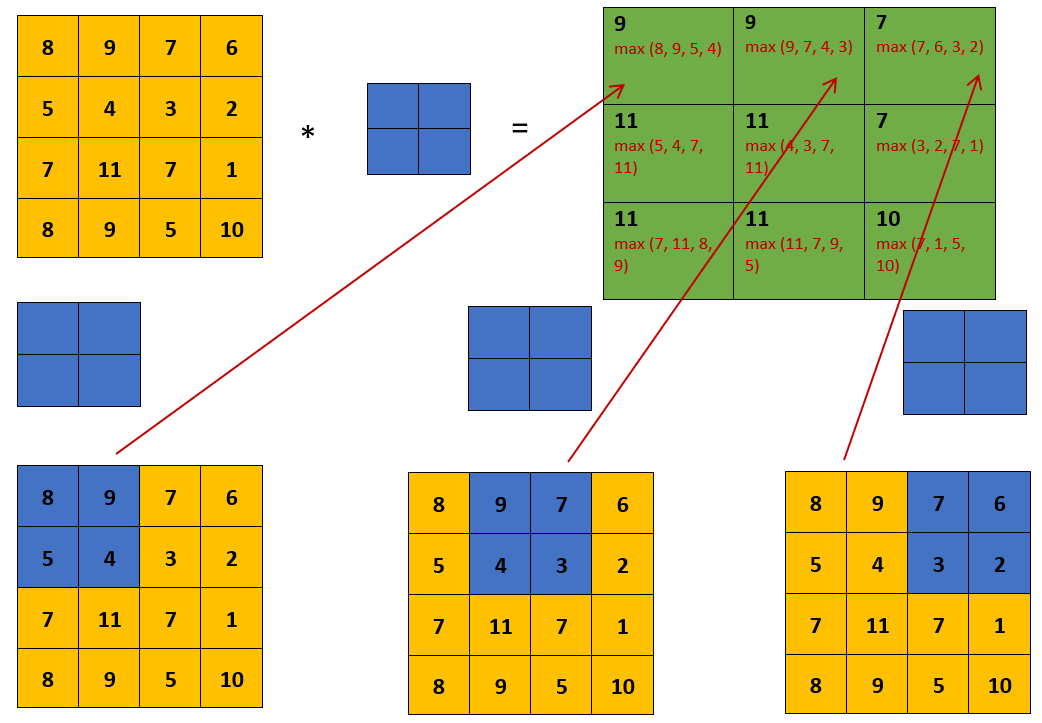
The image above shows the effect of convolution on images. The filters like this are used for edge detection. The example above is an example of vertical edge detection.

## Pooling

Pooling is very similar to convolution but instead of calculating sumproduct of the superimposed elements, we either take maximum, minimum or average of these elements.

The below image shows the calculation for max pooling. The min and average pooling just replace the max with min and average for the calculation. The shape of the output matrix is same as convolution but the values are very different. Hence, it can lead to very different feature discoveries than convolution.

Intuition for pooling - represent a large filter-sized area by the most/least/average prominent element and suppress the others. This might lead to discovery of the most prominent or ubiquitous pattern.



**Fig.9 Explanation of Max Pooling operator**

## Distance measures for arrays/matrices

An array of n-dimension can be seen as a point in an n-dimensional plane. A point in very familiar 2D plane is represented as A(x,y) or more generally as A(x1, x2).

The Euclidean distance between two points A(x11, x21) and B A(x12, x22) in a 2D plane is given by

D = {(x12 - x11)2 + (x22 - x21)2}½

*superscripts refer to the point number/index; subscripts refer to the axis number/index*

Similarly, the Euclidean distance between two points A(x11, x21, X31,……..,Xn1) and B(x12, x22, X32,……..,Xn2) in a n-dimensional plane is given by

D = {(x12 - x11)2 + (x22 - x21)2}1/2 + (x32 – x31)2}1/2 + …………..+ (xn2 – xn1)2}1/2

In some applications, a different distance measure called Manhattan distance is used. Instead of summing up square of differences of coordinate values, Manhattan distance sums up the absolute value of differences of coordinate values.

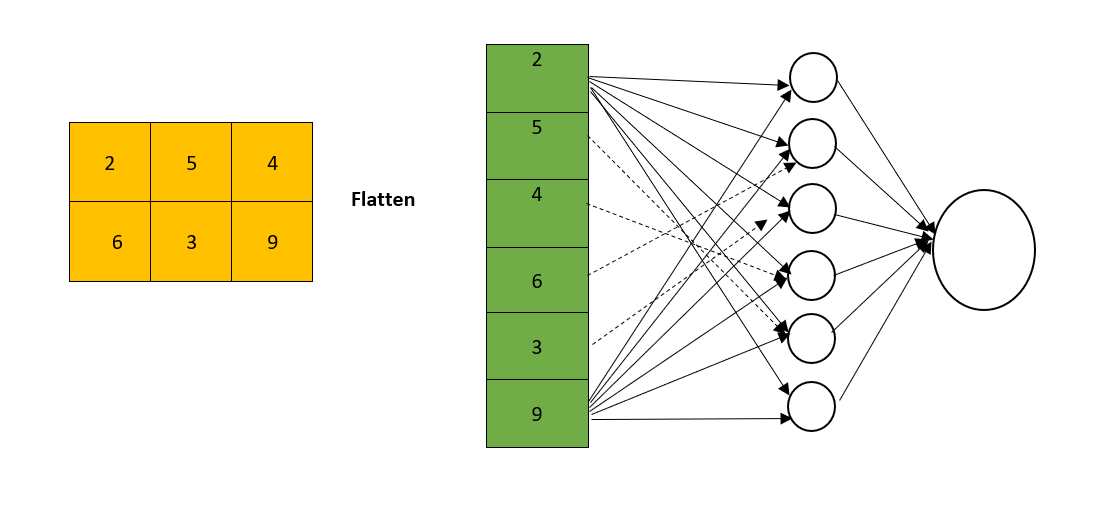
D = { |x12 - x11| + |x22 - x21| + |x32 – x31| + …………..+ |xn2 – xn1|

Distances are a good measure of similarity. The premise is that neighbor points should exhibit similar behavior. IN the context of face detection and recognition, the distance between n-dimensional face embeddings is calculated to gauge the similarity or dissimilarity between the two faces. In most of the cases in the context of face detection, the used distance measure would be one of the Euclidean and Manhattan.

## Fully Connected Layer

This is a part of a neural network which flattens a 2D/3D matrices to a 1D array and connects them to a node so that it becomes a linear combination of those variables in the 1D array.

Fully connected layers are used along with Flatten layers at the end of many convolutional neural networks to get the final output. The output of this layer is generally passed to a softmax layer for a multi-class classification problem. This layer basically converts n-dimensional data to 1D data so that a simple sum of products with trained weights can be calculated as output.



**Fig.10 Explanation of Max Pooling operator**

## Extracting face features - Deep Face, Face Net

# Algorithms

The mathematical computation on images such as face verification, detection and recognition require the following:

1. **Representation of image as a 1D vector of numbers** - achieved by obtaining pixel values of image and running set of convolution, pooling etc. operations followed by a flattening to make it a 1D vector
2. **Distance measure** - A distance measure like Euclidean distance to quantify the similarity or dissimilarity between images (or their numeric representation)

Machine learning algorithms e.g. image classification etc. generally require a lot of examples to train themselves on tasks. But face verification and recognition belong to a class of algorithms which learn to perform tasks by seeing very few (even one) examples. This kind of learning is called One-shot learning. Next we would see how one-shot learning algorithms are applied for these tasks.

# Algorithm behind Face Verification - Siamese Network

The algorithm used to train a face verification model is called Siamese Network. Some of the salient features of this algorithm are listed below.

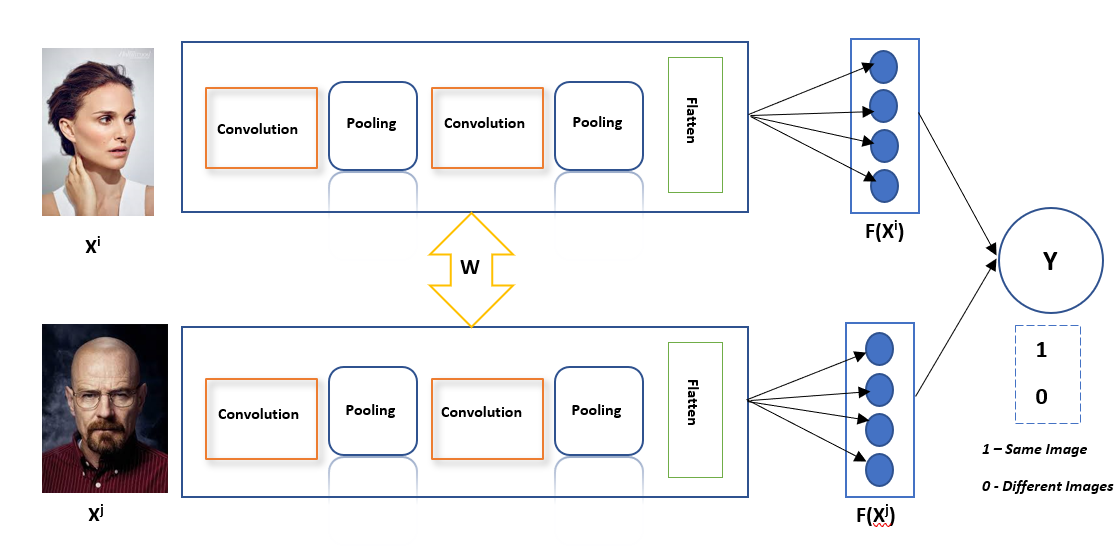
## Training Architecture

The Siamese Network consists of

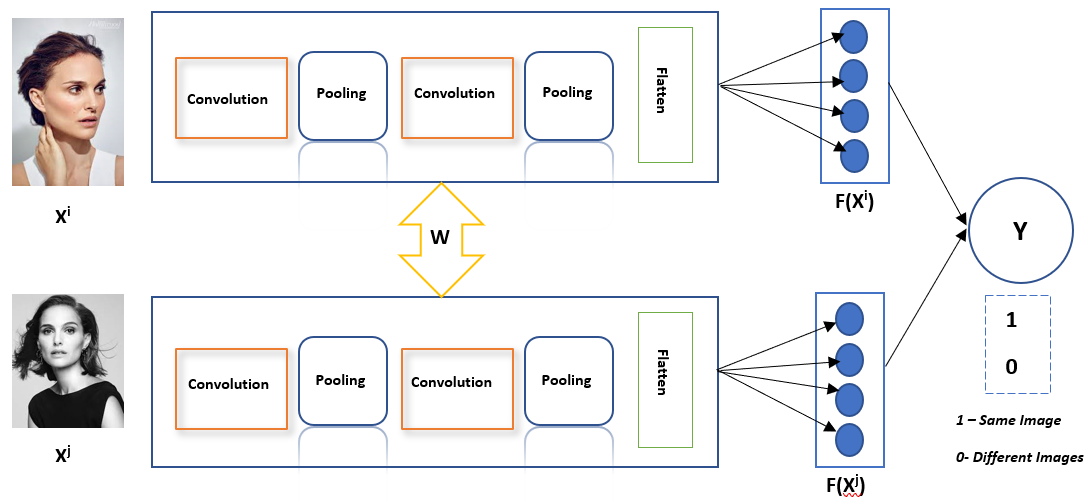
i) two conjoint neural nets (consisting of many convolution and pooling layers) each of which outputs a same-sized flat 1D array from an input image

ii) a weight array shared by the two nets

iii) a distance metric to calculate the similarity/dissimilarity between the two images



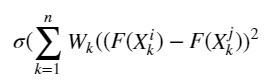
**Fig.11: Schematic of Face Verification training architecture - Siamese Network (different images)**



**Fig.12: Schematic of Face Verification training architecture - Siamese Network (same images)**

## Training Goal

The goal of the one-shot learning/training architecture shown above is to learn the network parameters (convolution, pooling etc. parameters) for best F(Xi) representation of an image such that difference between

****

and Y is minimised. Note that the summation in the expression above is on the elements (dimensions) of the face embeddings. For example, if the face embeddings are 128 element long, k would range from 1 to 128.

In other words, the goal of learning is to obtain face embeddings F(Xi) and F(Xj) for images Xi and Xj such that the distance between them

||F(Xi)-F(Xj)|| is small, if the images are same

||F(Xi)-F(Xj)|| is large, if the images are different

## How to DIY

One-shot training is easier than the traditional neural network training as it requires lesser number of examples of the same type but it still requires a substantial number of Siamese pair of images. Fortunately for us, these models have been trained very well on large number of such pairs by companies like Facebook and are outsourced by them.

These pre-trained models such as DeepFace, FaceNet, MTCNN etc. can be used to obtain the embeddings of the input images as-well-as the registered images. These pre-trained models perform the task of convolution and pooling blocks encased in the blue box in the schematic above. Popular deep learning libraries like TensorFlow and Keras have facilities to call these pre-trained models over images to get their embeddings.

Once the embeddings are obtained, the distance between the input image and the registered image is calculated. If the distance is less than a given threshold then it is considered a positive verification.

dist(F(Xi), F(Xj)) < c → Positive verification

dist(F(Xi), F(Xj)) > c → Verification failed

Distance formulas in Keras and Tensorflow can be used to calculate these distances. A simple UI can be created to register user image(s) at the beginning of the usage.

# Algorithm behind Face Recognition - Triplet Loss

Face recognition uses another distance-based measure called Triplet Loss. Some of the features of a triplet loss algorithm are listed below.

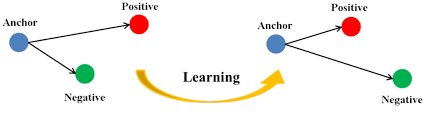
## Training Architecture

The Triplet loss is calculated as follows

i) two different pairs of images are taken - one image called anchor is present in both pairs. Another image in one pair is called positive and in 2nd one it is called negative.

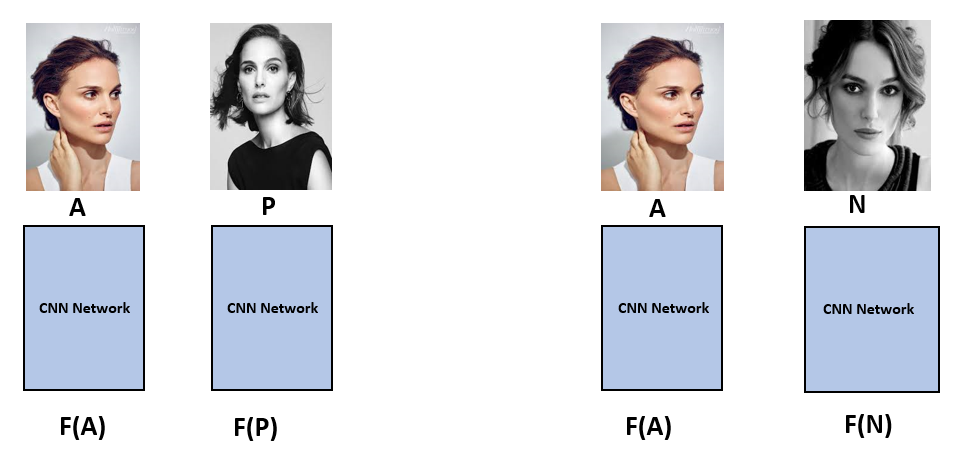
ii) The difference in distances between anchor and positive and anchor and negative is calculated.

iii) The goal of the training is to find face embeddings such that distance between anchor and positive is lesser than the anchor and negative



**Fig.13: The goal of triplet loss learning is to reduce the distance between anchor and positive image and increase the distance between anchor and negative**

From the picture and description above, it should become clear why the algorithm is called a triplet loss- there is a triplet of pictures on which the loss is calculated.



**Fig.14: Schematic of Face Recognition training architecture - Triplet Loss**

In the schematic picture above, A is Anchor image, P is Positive image and N is Negative image. F(A) is the embedding of anchor image, F(P) is the embedding of positive image and F(N) is the embedding of the negative image.

## Training Goal

The goal of the training is to reduce the distance between anchor and positive while increasing the distance between anchor and negative.

This can be mathematically written for one image as

*||F(A) - F(P)||2 + c <= ||F(A) - F(N)||2*

The loss for a given set of triplet images is given by

*L(Ai, Pi, Ni)* = max(*||F(Ai) - F(Pi)||2 - ||F(Ai) - F(Ni)||2  + c, 0)*

The loss for all the triplet samples is given by summation of loss over all the images.

Loss = ∑*L(Ai, Pi, Ni)*

Note that, during training, if A,P,N are chosen randomly,

d(A,P) + α ≤ d(A,N)

is easily satisfied because P and N would be very different from each other and hence from A.

Instead, we should find training examples which are hard to train. In other words, choose triplets such that distance between A and P is approximately similar to distance between A and N.

d(A,P) ~ d(A,N)

## How to DIY

Steps to DIY a Face Recognition system is similar to Face Verification system. A summarised set of steps is as follows:

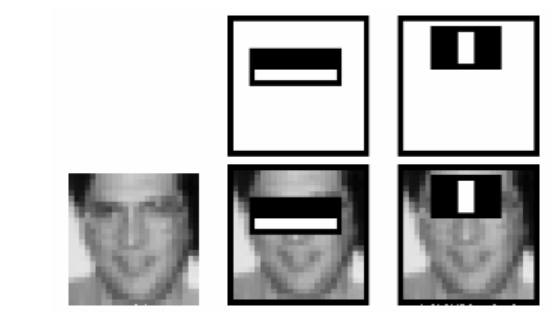
1. Get the face embeddings of the input(query image for which the recognition has to happen) using Deep Face and Face Net (trained on triplet loss) packages in TensorFlow/Keras
2. Calculate the distance of this face embedding from all the registered images embeddings
3. Return the registered image with the least distance from the input image as the match
4. Return the identification of the matched registered image

# Face Detection using Haar Cascade Classifiers

Haar Cascade Classifiers is a machine learning (no neural networks involved) based Face Detection algorithm which is probably the easiest to Do It Yourself.

The salient features of this approach are:

1. It involves vertical and horizontal edge detection using methods described in Fig.8 above
2. It creates features from these edges. Some of the examples of these features for eye detection are shown below. First feature captures the fact that the area below the eyes is less darker than the eye area. The second feature captures the fact that the eyes are darker than the area between them.



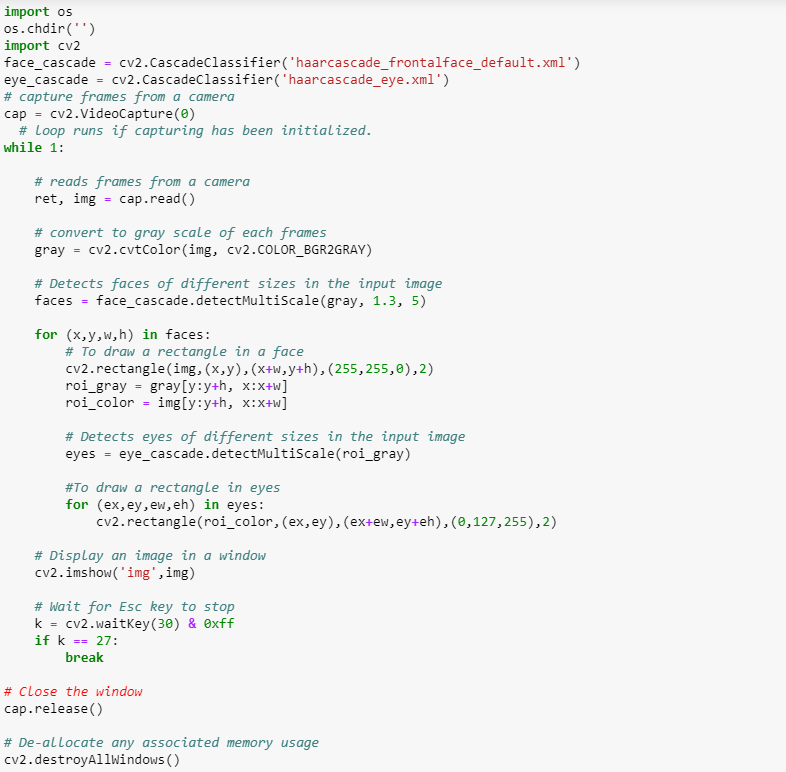
**Fig.15 - Features for detecting eyes used in Haar Cascading Filters. The features are developed based on edge detection**

1. Thousands of such features are created on each image using a moving window on. Each window/feature is classified as containing an eye/face based on a threshold value of the feature
2. The misclassified windows (eyes classified as non-eyes and vice-versa) are given more weightage in the next iteration of training. It follows Adaboost ensemble method of training which boosts a weak classifier to a strong classifier.
3. Haar algorithms uses a cascade of such features described above. The training over features happens in stage. So few features are part of the first stage, few of the second, few of the third and so on. Only the windows which classify as face/eye in the first stage are passed to the 2nd stage and so on.

How to DIY

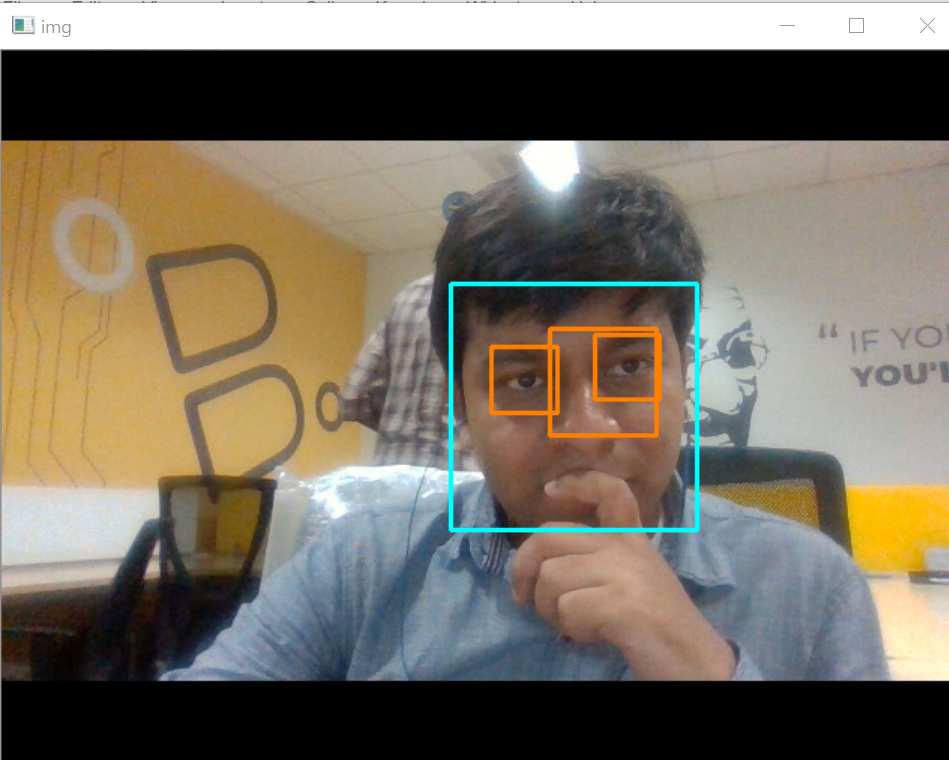
This is probably the easiest to DIY because doesn’t require installation of deep learning frameworks like Keras and Tensorflow. Moreover the trained models are available as XML files.

You can run the code below to draw bounding boxes around face and eye on a live video from your computer camera.



**Fig.16 - The code for detecting face and eyes using a pre-trained Haar Cascade filter**

One screen capture from the resulting video on running the code looks like this



**Fig.17 - The bounding boxes around face and eye detected using Haar Cascade filters**

The XML files needed in this code can be downloaded from the link below

<https://github.com/opencv/opencv/tree/master/data/haarcascades>

And can be kept in the working directory where code is residing.

## Summary

Face detection and recognition applications have been democratized thanks to their increased usage in smartphones and premise entrances as an access control technology. FaceNet and DeepFace are pre-designed and pre-trained convolutional neural networks to obtain contextual and accurate face embeddings. Siamese Network and Triplet Loss are used for face detection and recognition both of which use Euclidean distances between face embeddings (with different training goals) for performing these tasks. Tensorflow and Keras can be used with DeepFace and FaceNet libraries to build these systems of our own.

## Further reading & References

Deep Face - <https://www.cs.toronto.edu/~ranzato/publications/taigman_cvpr14.pdf>

FaceNet - <https://arxiv.org/abs/1503.03832>

Image Processing - <https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_tutorials.html>

Haar Cascade Filter - <https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html>