# A COMPREHENSIVE STUDY OF THE USE OF EIGENFACES IN FACIAL RECOGNITION SYSTEMS

By Antima Jangid[1], Arshi Rizvi[2], Ashika Deb[3], Disha Verma[4], Kritika Mishra[5], Nancy Dahiya[6], Rhea Ajit John[7] & Tahrima Ansari[8]

*Miranda House, University of Delhi*

***Abstract-*** In the modern world, facial recognition has a challenging scope in the field of image analysis and computer vision. A human face is a complex multi-dimensional structure and demands really good computing techniques for recognition. A facial recognition system is a type of biometric technology, capable of identifying or verifying a person from a digital image or a video source. This paper majorly works with the idea of Eigen Faces and Principal Component Analysis(PCA) in order to achieve facial recognition. It also focuses on the computer algorithm and coding done in python language, for the same. Moreover, in the end, it compares EigenFaces method with other methods such as Local Binary Patterns(LBP), Fisher Faces and Line Edge Maps and explains how they can alternatively perform facial recognition.

***Keywords*** - Facial recognition, Eigen faces, Eigen vectors, Principal Component analysis,

Linear Binary Pattern, Fisher faces, Line Edge Map

[1] B.Sc. Physical Science with Computer Science, 1st year, Miranda House, University of Delhi.

[2] B.Sc. (H) Mathematics, 1st year, Miranda House, University of Delhi.

[3] B.Sc. (H) Mathematics, 1st year, Miranda House, University of Delhi.

[4] B.Sc. (H) Mathematics, 1st year, Miranda House, University of Delhi.

[5] B.Sc. (H) Mathematics, 1st year, Miranda House, University of Delhi.

[6] B.Sc. Physical Science with Computer Science, 1st year, Miranda House, University of Delhi.

[7] B.Sc. Physical Science with Computer Science, 1st year, Miranda House, University of Delhi.

[8] B.Sc. (H) Mathematics, 1st year, Miranda House, University of Delhi.

# Introduction

With advancement of technology, facial recognition systems also have shown encouraging progress. It works by capturing images of a face from up close or a distance, the software involved uses different algorithms to identify the individual or even store the image for future use. Starting from the privacy mechanisms on smartphones to government agencies, facial recognition systems have found a wide array of fields where they can be used. The modern application of face detection technology has attained much attention in different fields for safety and security purposes, business intelligence, and marketing tools. There are various facial recognition software such as Deep Vision AI, FacFirst, Face++, Trueface, SensorTime and many more but all of them use some common algorithms such as Eigenfaces method, fisherfaces method, line edge map method, local binary pattern histograms, Scale Invariant Feature Transform(SIFT), Speed Up Robust Features(SURF). Appearance-based approaches to identification have made a comeback from the early days of computer-vision science, and this may have been aided by an individual approach to face recognition.

This paper principally deals with the use of Eigenfaces algorithm. In this paper, the mathematical aspect of Eigenfaces is explored as well as other concepts that go hand in hand with it. Furthermore, it compares other popular algorithms with eigenfaces method to find out the advantages and disadvantages of one over the other. Finally, it shows a practical implementation of this algorithm in Python. The main goal of this paper is to do a comprehensive study of eigenface algorithm and understand what its benefits and drawbacks are.

## METHODOLOGY

In this paper we have covered different methods of facial recognition like Eigenfaces and PCA, Fisherfaces, Local Binary Pattern and Line Edge Map. The focus of this paper is on the use of eigenfaces in face recognition. The other topics have been briefly explained in the paper from a comparative perspective. We have reviewed many existing research papers and websites to have complete and deep understanding of the topic and then verified the information through different sources(which are mentioned in the references) was added. Discussions with the mentors and fellow group members helped us to place each point and fact at its correct place so that it can be easily understood by the readers. Apart from the theoretical and mathematical aspects, we have performed the program using python software for which we collected data from an existing database and explained each function and package used in the programming. Also in the programming part, some difficulties were faced by us; for instance, while importing some of the libraries, program showed error and took a lot of time, but we tried to tackle all these problems and also took ideas from different research papers and our mentors during the process. We have tried to frame the structure of the paper in a way that it should be understood just by giving a thorough reading.

# FACIAL RECOGNITION

The face recognition system is similar to other biometric systems. The idea behind the face recognition system is that each individual has a unique face. Just like finger print, face also has many structures and features which are unique.

In a face recognition system, there is a database which consists of the images of the individuals that the system has to recognize. If possible, the database should provide multiple photos of the same person. If the images are selected so that they account for varying facial expressions, lighting conditions, et cetera, the solution of the problem can be found more easily as compared to the case where only a single image of each individual is stored in the database.

A face recognition algorithm processes the captured image and compares it to the images stored in the database. If a match is found, then the individual is identified. If no match is found, then the individual is reported as unidentified.

# 1. EIGEN FACES

An **eigenface** is the name given to a set of eigenvectors when used in the computer vision problem of human face recognition [1].

Sirovich and Kirby (1987) introduced the method of using eigenfaces for identification and using them in face classification by Matthew Turk and Alex Pentland. After going through the paper you will get to know what eigenfaces actually are.

**To understand eigenfaces and the algorithm of facial recognition some terms must be understood. They are:**

## 1.1 Vector Space:

Collection of objects called vectors, which can be added together and multiplied or scaled by numbers [2].

Or it's a set V together with 2 operators that satisfy the 8 axioms.

1st operation: **+**

2nd operation: **.**

So, there are 8 axioms plus 2 closure properties for the above mentioned operations which need not to be discussed here.

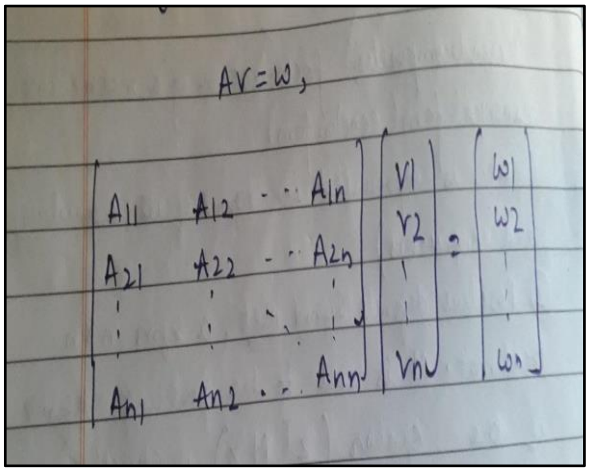
## 1.2 Eigenvectors and Eigenvalues:

Eigenvector is also known as characteristic vector. Eigenvector of linear transformation is a non –zero vector that changes at most by a scalar factor when that linear transformation is applied to it. And the corresponding eigen value is the factor by which the eigenvalue is scaled.

Geometrically, an eigenvector, corresponding to a real non-zero eigenvalue, points in a direction in which it is stretched by the transformation and the eigenvalue is the factor by which it is stretched. The direction is reversed if the eigenvalue is negative.

(In multidimensional vector space, the eigenvector is not rotated)

**For Example:** Consider the linear transformation of n-dimensional vectors defined by an 'n' by 'n' matrix A.



$$Av = w,$$

$$\begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix}$$

***Fig.** 1: n-dimensional vectors defined by an ‘n’ by ‘n’ matrix A.*

If it happens that ‘v’ and ‘w’ are scalar multiples that is, if

$$Av = w = xv$$

Then v = eigenvector of the linear transformation A and x will be eigenvalue. Then,

$$Av = xv$$
$$Av-xv=0$$
$$(A-xI)v=0$$

## 2. PRINCIPLE COMPONENT ANALYSIS

Principle Component Analysis(PCA) is a mathematical method used to reduce the number of face-recognition variables. The main aim of PCA is to implement the system for a particular face and distinguish it from a large number of stored faces with variations.

PCA does dimensionality reduction and seeks special features in the training images and eliminates identical characteristics from the face image(explained in the algorithm). By getting unique features our recognition task gets easier and simpler.

### 2.1 Process of PCA

One of the approaches of PCA used in facial recognition is Eigenface approach. Recognition is performed by projecting the test image onto the subspace spanned by eigen faces and then conducting further classification by measuring minimum euclidean distance. The work done by PCA is explained in algorithm [3].

### 2.2 Algorithm:

The recognition process involves two steps:

**1.Initialization Process (Training process)**- Initialization Process involves acquiring initial set of face image called as training set and calculating Eigenface from training image set.

**2.Recognition Process (Testing process)** - Recognition process involves calculating a set of weights based on the input image of training set and Eigenfaces by projecting input image on each of the Eigenfaces . Determine if the image is known or unknown by calculating minimum Euclidian distance.

Here is a flowchart which shows the complete process of face recognition:

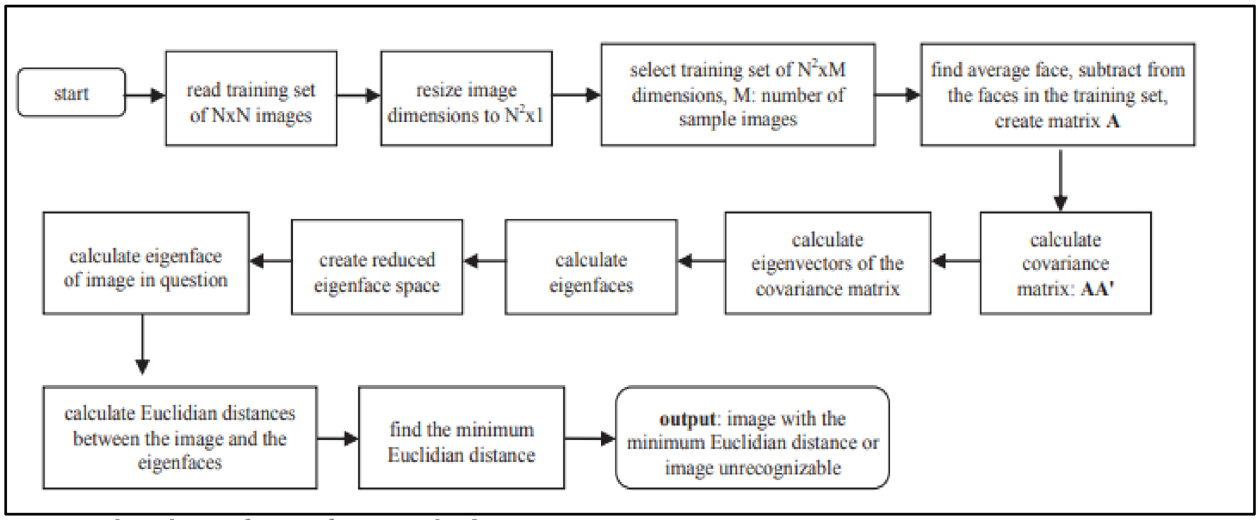

start
read training set of NxN images
resize image dimensions to N²x1
select training set of N²xM dimensions, M: number of sample images
find average face, subtract from the faces in the training set, create matrix A
calculate covariance matrix: AA'
calculate eigenvectors of the covariance matrix
calculate eigenfaces
create reduced eigenface space
calculate eigenface of image in question
calculate Euclidian distances between the image and the eigenfaces
find the minimum Euclidian distance
output: image with the minimum Euclidian distance or image unrecognizable

*Fig. 2: Flowchart of Eigenfaces method*

## 2.3 Training algorithm:

The following image helps to understand the procedure more clearly:

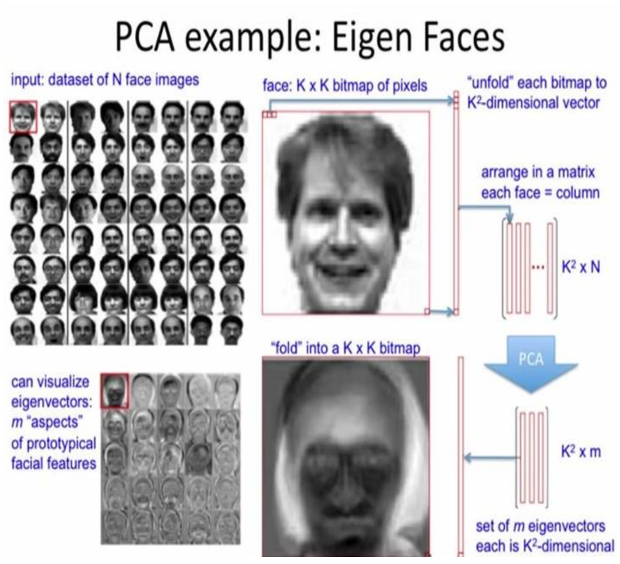

PCA example: Eigen Faces
input: dataset of N face images
face: K x K bitmap of pixels
"unfold" each bitmap to K²-dimensional vector
arrange in a matrix each face = column
K² x N
PCA
"fold" into a K x K bitmap
can visualize eigenvectors: m "aspects" of prototypical facial features
K² x m
set of m eigenvectors each is K²-dimensional

*Fig 3.: using PCA to get Eigenfaces*

1. **First we have to read all the training images**(let's consider we have a dataset of i face images).

*Fig4. : input database of i images of dimensions n x n (taken from online database)*

After that we have to convert each **[n x n] image matrix** into **[(n x n) x 1] image vector (Here n x n means the no. of rows becomes n$^2$)**

[Some of you may not know how an image is represented as a matrix and then converted into a vector] Go through the following paragraph to get an idea of this:

An image is seen as a picture made up of pixels. For most images, pixel values are integers that range from 0 (black) to 255 (white). Even if you view the full-size image, it is difficult to see the individual **pixel** intensities.

The pixel value at any point denotes the image intensity at that spot, and that's also called the **gray level.**

Look at the images given below to get an idea of it:

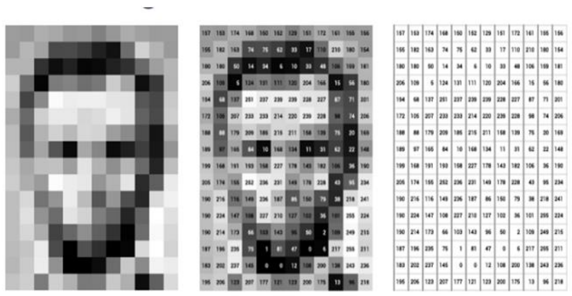


*Fig5.: pixel values of gray level image(Google image)*

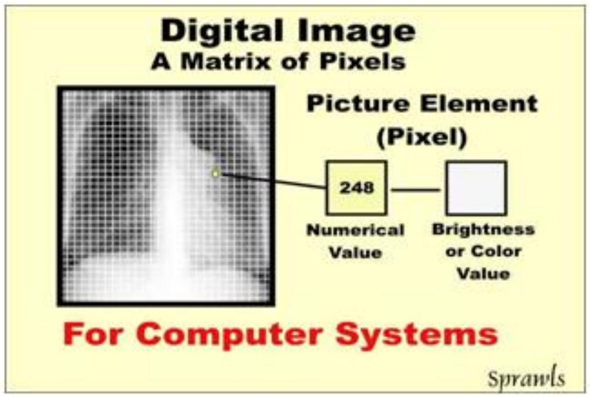
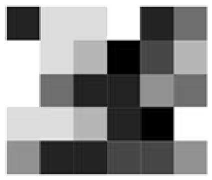


*Fig.6a: Pixels in a picture.*



$$\Longleftrightarrow \begin{bmatrix} 6 & 1 & 1 & 0 & 6 & 4 \\ 0 & 1 & 2 & 7 & 5 & 2 \\ 0 & 4 & 6 & 6 & 3 & 4 \\ 1 & 1 & 2 & 6 & 7 & 0 \\ 3 & 6 & 6 & 5 & 5 & 3 \end{bmatrix}$$

*Fig6b. : Converting image into an image matrix*

**(**Flatten image matrix to vector). For example, Images are converted from 256 by 256 images into a single dimension vector of size 65,536.

$$I = \begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1n} \\ x_{21} & x_{22} & \cdots & x_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}_{m \times n} \xrightarrow{\text{CONCATENATION}} \begin{bmatrix} x_{11} \\ \vdots \\ x_{1n} \\ \vdots \\ x_{2n} \\ \vdots \\ x_{mn} \end{bmatrix}_{N\times 1} = x.$$

N mentioned above is the total no. of elements in the image matrix here it is n*n.

This conversion is important because we need a 2-D square matrix to compute eigenvectors.]

2. **Now we have to construct a low-dimensional linear subspace that best explains the variation in the set of face images**.

So, then next step is to normalise the face vectors . **Normalisation** removes the common features that these ‘i’ faces share together, so that each face will be left behind with only its unique features. We will do normalisation by following steps:

**a. First calculate average face vector as:**

$$\frac{sum\ (all\ image\ vectors)}{number(images)}$$

The average of the image set is calculated as[4]:

$$\psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i$$

where $\psi$: average image, M: number of images, $\Gamma_i$: image vector.

The average face vector will contain the average feature of whole training set.

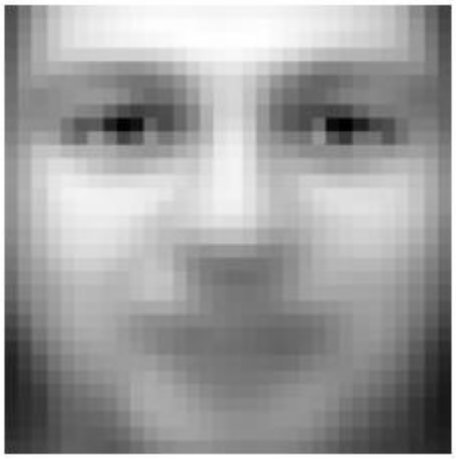


*Fig7.: this image shows the mean image of the training set we have taken as an example above*

**b. Then subtract average_face_vector from every image vector to get the variance matrix** [5].

Put each (image_vectors — average_face_vector) in matrix forming:

**A = [(n x n) x i]** matrix

(where i = number of images).

The difference between each image and the average face is calculated as:

$$\phi_i = \Gamma_i - \psi, \quad i = 1, 2, \ldots, M$$

where $\phi_i$ is the difference between the image and the average image.

Here we will be left with the normalised face vector. All the normalised face vectors are stored in a matrix A of dimension :

[(n x n ) x i]

3. **The next step is to calculate the covariance matrix** (A covariance matrix is a square matrix which gives the covariance between each pair of elements of a given eigenvectors of training image set.) **and find its eigenvectors and eigenvalues as:**

$$C = \frac{1}{M}\sum_{n=1}^{M}\varphi_n\varphi_n^T = AA^T,$$

$$Ce_i = \lambda_i e_i.$$

If we calculate covariance matrix using above formula , it will be of dimension $N^2$ X $N^2$ which will be very large , for example if original image is of 50 X 50 then C will be of 2500 X 2500 then it will create 2500 Eigenvectors and each Eigenvector will be of size 2500 X 1 .

As the goal of PCA is to represent each image as a linear combination of K selected Eigenvectors where K < i. If i is 100 (suppose) then from Covariance Matrix we have to find 100 or less than 100 Eigenvectors from 2500 Eigenvectors which will be a huge amount of calculation.

So the solution of this problem is **dimensionality reduction.**

One of the theorems in linear algebra states that the eigenvectors $\mathbf{e_i}$ and eigenvalues $\lambda_i$

can be obtained by finding eigenvectors and eigenvalues of matrix **C1 = $A^T$ A** (dimensions i×i) as:

If $\mathbf{v_i}$ and $\mu_i$ are eigenvectors and eigenvalues of matrix C1 then:

$$A^T A v_i = \mu_i v_i .$$

Multiplying both sides of equation with $A$ from the left, we get:

$$A A^T A v_i = A \mu_i v_i ,$$

$$A A^T (A v_i) = \mu_i (A v_i),$$

$$C(A v_i) = \mu_i (A v_i).$$

From above statement, it can be concluded that C1 and C have same eigenvalues and eigenvectors related to each other as:

$$\mathbf{e_i = Av_i \text{ and } \lambda_i = \mu_i}$$

So, now instead of calculating eigenvectors and eigenvalues directly from the matrix $AA^T$ we can calculate them using the above method.

4. **The eigen vectors are sorted based on their eigen values, and the K-Best among them are chosen.**

The direction that captures the maximum covariance of the data is the eigenvector corresponding to the largest eigenvalue of the data covariance matrix. Furthermore, the top k orthogonal directions that capture the most variance of the data are the k eigenvectors corresponding to the k largest eigenvalues. These selected k eigenvectors will represent the whole training set.

**To understand it more clearly, look at the following example shown through images:**

Let's consider we have following image set as input database:

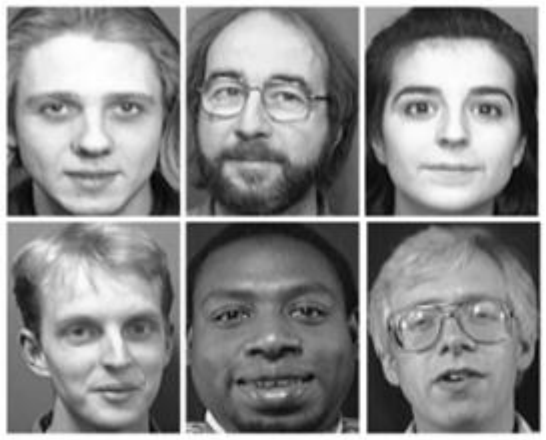


*Fig8.: input database*

**First three eigenfaces**(eigenvectors corresponding to the highest eigenvalues) **look like this** (They somewhat resembles the faces of the training set):

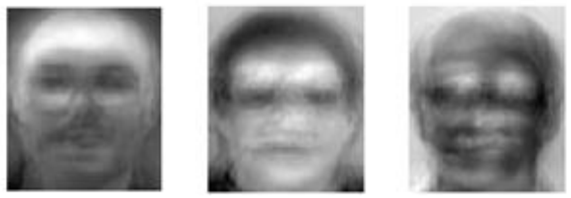


*Fig9.: first 3 eigenfaces*

**Last three eigenfaces**(corresponding to the smallest eigenvalues) **look like this**(They do not carry important information about the images from the training base because they correspond to small eigenvalues ):

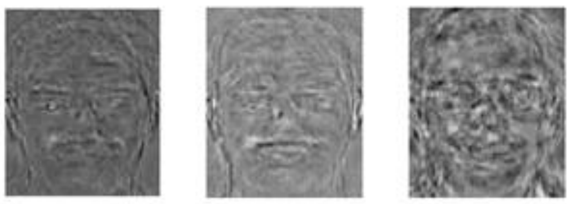


*Fig10.: Last 3 eigenfaces*

So, the eigenfaces corresponding to the highest eigenvalues are retained. Those eigenfaces define the face space. The eigenfaces obtained are known as **Principal components.** And the process is known as Principal Component analysis.

5. **Now obtained K Eigenvectors have to be converted into their original dimensionality of face vector i.e. $n^2$ X 1**.

We can change the dimensionality by linear algebra:

Let $\mathbf{U_i}$ = ith Eigenvector in higher dimensionality space (i.e., original dimensionality space).

$\mathbf{V_i}$ = ith =Eigenvector in lower dimensionality space.

$$\mathbf{U}\boldsymbol{i} = \mathbf{A}\ \mathbf{V}\boldsymbol{i}$$

This will return Eigenvector (Eigenface) in higher dimensionality.

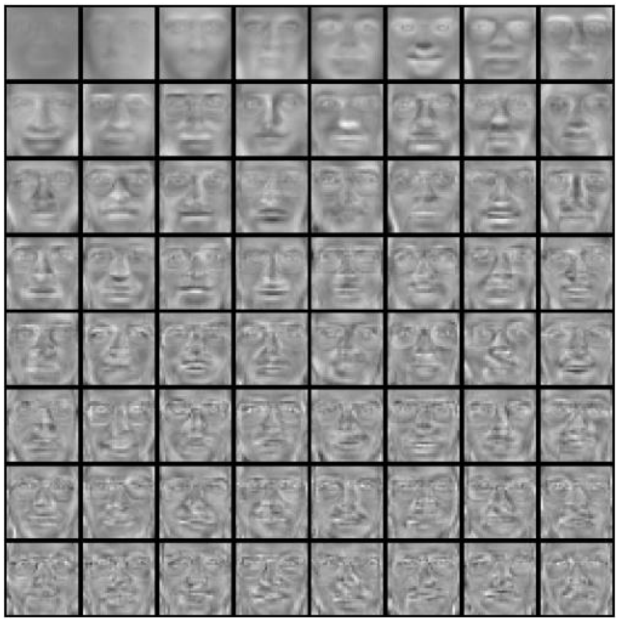
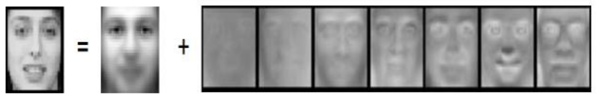


*Fig11.: Top eigenvectors or eigenfaces of the database taken in the starting.*

6. **Now we will project each training image (let's say $x_i$) onto subspace spanned by principal components ($u_i$)** .

We will represent face image(of training set) as a weighted sum of k Eigenfaces + mean image.



$x_i$ = mean image + W1u1+W2u2+W3u3….

Weights $W_1, W_2, W_3, \ldots W_k$ are the weights w.r.t. k Eigenfaces. (Weight means the combination of that particular eigenface in the image). Each image in training set will have its own weight vector which will be stored in a vector $\mathbf{y_i}$ by the system. All the vectors of weights of training images are stored in a matrix Y.

**This step completes our Training Part.**

## 2.4 Testing algorithm:

Here is a flowchart of the process involved:

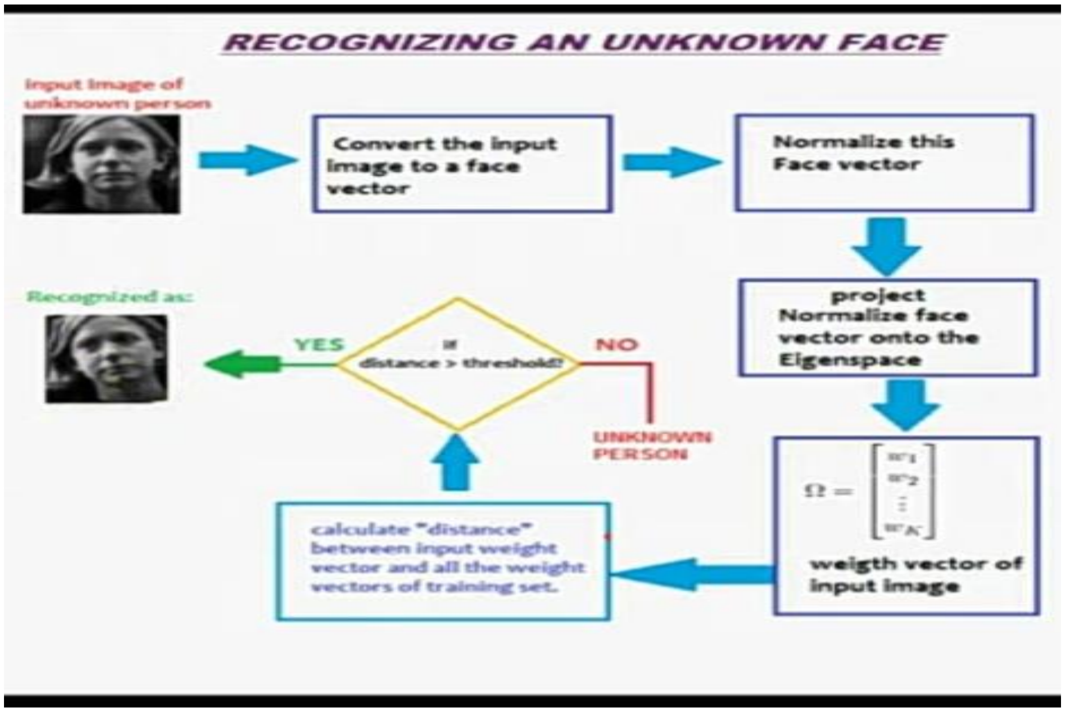

RECOGNIZING AN UNKNOWN FACE
input image of unknown person
Convert the input image to a face vector
Normalize this Face vector
project Normalize face vector onto the Eigenspace
weigth vector of input image
calculate "distance" between input weight vector and all the weight vectors of training set.
if distance > threshold?
YES
NO
UNKNOWN PERSON
Recognized as:

*Fig12.: flowchart of recognizing an unknown face*

1. **For testing, convert the input test image into face vector** (say **P**) as same as explained in the training process.
2. **Then, normalise the face vector** (subtract average face vector from each face vector of training set). Let normalised face vector be **T**.
3. **After that, project the normalised vector T onto eigenfaces and calculate the weights as:**

$$\mathbf{T} = W1\mathbf{u1}+W2\mathbf{u2}+w3\mathbf{u3}+.....$$

where **u1**,**u2**,**u3**,... are eigenfaces.

4. **The weights calculated are stored in a vector** (say **V).** Classification is done by determining the distance d(P,yi) between **V** and each vector **yi** of matrix Y.

The most common is the **Euclidean distance**, but other measures may be used (like Manhattan distance). Formula for Euclidean distance:

$$d(P,yi)=\sqrt{\sum_{i=1}^{D}(a_i-b_i)^2}=\|P-yi\|$$

where $a_i$ and $b_i$ are the corresponding elements of vectors **P** and **yi** respectively.

5. **Recognition**: If the minimum distance between test face and training faces is higher than a threshold Ө, the test face is considered to be unknown, otherwise

it is known and belongs to the person for whose the distance is minimum.

The program requires a minimum distance between the test image and images from the training base which is called ϴ. There is no formula for determining the threshold. The most common way is to first calculate the minimum distance of each image from the training base from the other images and place that distance in a vector **H**. Threshold is taken as 0.8 times of the maximum value of vector **H .**

$$\Theta = 0.8 * \max(\mathbf{H})$$

## 3. LOCAL BINARY PATTERNS

Apart from the other methods we use in facial recognition, Local Binary Patterns is one of the oldest and more popular. This approach was introduced in 1996 by Ojala et al[6].

***Local Binary Pattern(LBP) is a simple but very powerful texture operator which labels pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number[7].***

Texture is an important characteristic of images. It provides information about physical properties of objects like smoothness, roughness, or difference in surface reflectance such as color [8]**.** The term “operator” refers to a mapping function that can transform an image from one form to another.

In 2004, Ahonen et al.[9] successfully applied the LBP operator to face recognition by dividing an image into regions from which LBP features were extracted and concatenated into enhanced feature vectors. These LBP features consist of binary patterns that describe the surrounding pixels in the regions. The characteristics obtained from the regions are concatenated into a single characteristic histogram forming a representation of the image The characteristics obtained from the regions are concatenated into a single characteristic histogram forming a representation of the image. The images can then be compared by measuring the similarity via the distance between their histograms using various mathematical techniques.

### 3.1 Working of LBP

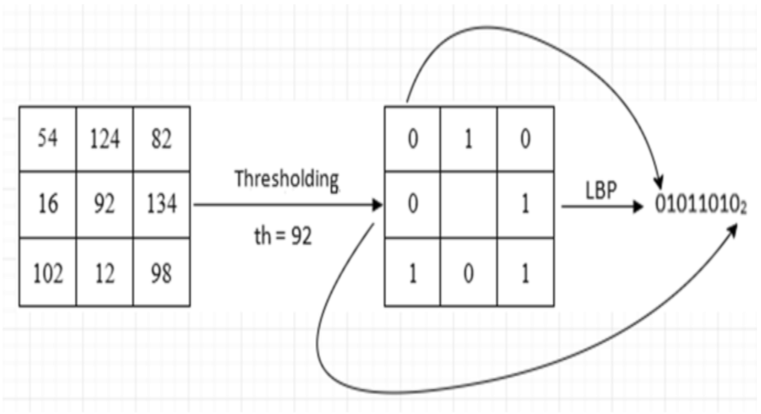


*Fig. 13: LBP operator on center pixel as a threshold*

The LBP has been of two types .The original LBP operator, introduced by Ojala et al, worked with eight neighbors of a pixel, using the value of this center pixel as a threshold. If a neighbor pixel has a higher or equal gray value than the center pixel, then a 1 is assigned to that pixel, else it gets a 0.

The LBP code for the center pixel is then produced by concatenating the eight ones or zeros to a binary code in the clockwise manner.

Here the entry in the centre of the 3x3 matrix, that is, 92 is considered as the threshold. For every pixel in the image, the relation between a center pixel and its neighborhood is encoded as an **LBP value.**

Later on, this technique was improved and made to work with neighborhoods of different sizes. In this case a circle is made with radius R from the center pixel. P sampling points are taken at the edge of this circle, and compared to the center pixel value. Interpolation (bilinear) is necessary to obtain the values of all sampling points in the neighbourhood for any radius and any number of pixels. For neighborhoods the notation (P, R) is used.

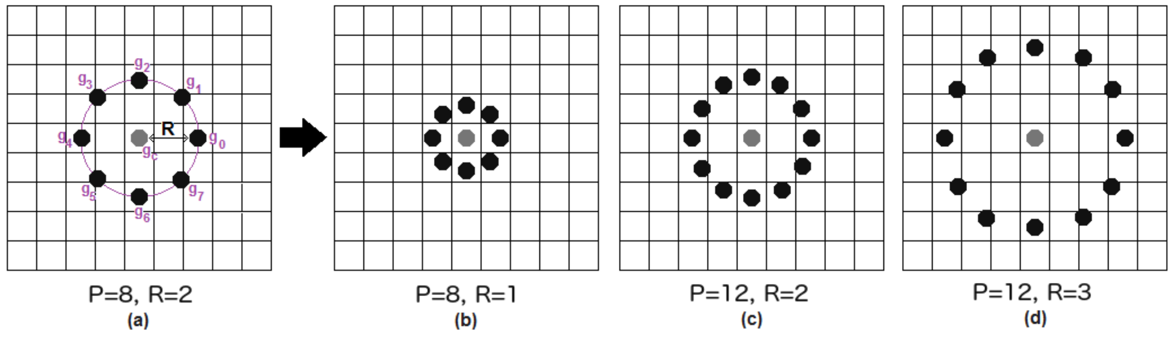


*Fig. 14 : Circularly neighbor-sets for three different values of P and R*

If the co-ordinates of the center pixel are ($x_c$, $y_c$) then the co-ordinates of its P neighbors ($x_p$, $y_p$) can be determined with the sines and cosines at the edge of the circle with radius R:

$$x_p = x_c + R\cos(2\pi\pi p/P)$$

$$y_p = y_c + R\sin(2\pi\pi p/P)$$

If the gray value of the center pixel is $g_c$ and the gray values of his neighbors are $g_p$, **with $p = 0, ..., P - 1$, then the texture. T can be defined in the local pixel ($x_c$, $y_c$) neighbourhood as**:

$$T = t(g_c, g_0, . . . , g_{P-1})$$

Next we do is, describe the texture by subtracting the value of the center pixel from the values of the points on the circle.

$$T = t(g_c, g_0-g_c, . . . , g_{P-1}-g_c)$$

$$T\approx (g_0-g_c, . . . , g_{p-1}-g_c)$$

Because $t(g_c)$ defines an image 's overall luminance, which is unrelated to the texture of the local image, it does not provide useful information for texture analysis.

**In case a point on the circle has a higher or the same gray value than the center pixel, a 1 is assigned to that point, and else it gets a 0:**

**$T \approx (s(g_0-g_c), \ldots, s(g_{P-1}-g_c))$**

**Where,**

$$s(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

**Once the Local Binary Pattern value for every pixel is calculated, the feature vector of the image can be constructed from histograms**. These histograms can then be used to measure the similarity between the images, by calculating the distance between them using various methods.

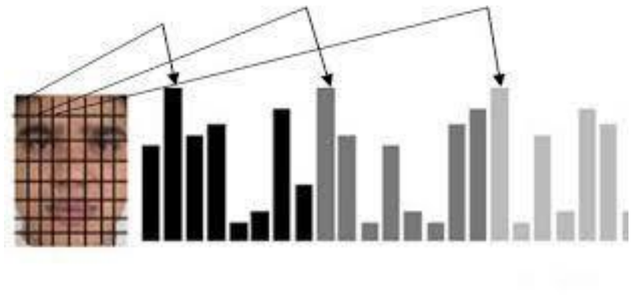


*Fig. 15: Face image divided into 64 regions, with for every region a histogram*

Here, a face image is divided into $8^2 = 64$ regions. For every region a histogram with all possible labels(histogram) is constructed. This means that every bin in a histogram represents a pattern and contains the number of its appearance in the region. **The feature vector is then constructed by concatenating the regional histograms to one big histogram.** The feature vector is effectively a description of the face on three different levels of locality: the labels contain information about the patterns on a pixel-level; the regions, in which the different labels are summed, contain information on a small regional level and the concatenated histograms give a global description of the face.

**Next, we Perform recognition of the test image using the nearest 'neighbor classifier' where the similarity measure can be any of the following: *histogram intersection, log-likelihood statistics, or Chi-squared***[10].

For example, Chi-squared can be computed by:

$$\chi^2(S,M) = \sum_{b=1}^{B} \frac{(S_b - M_b)^2}{(S_b + M_b)}$$

where S and M denote sample and model distribution. B is the number of bins in the distribution, $S_b$, and $M_b$ correspond to the probability of bin b in the sample and model distribution.

**A Chi- squared value of 0 indicates a perfect match with numbers closer to 0 indicating a better match than larger values.** This is exactly how Local Binary Pattern (LBP) works.
Given below is the flowchart of the whole alogorithm. :

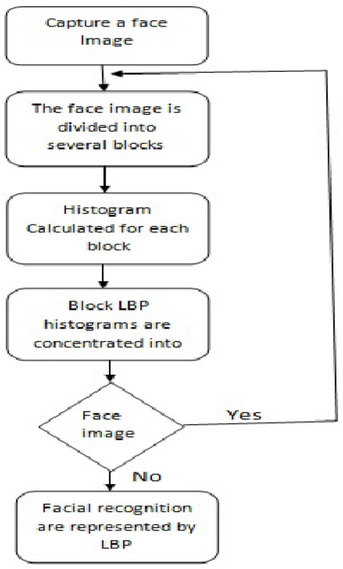

Capture a face Image
The face image is divided into several blocks
Histogram Calculated for each block
Block LBP histograms are concentrated into
Face image
Yes
No
Facial recognition are represented by LBP

*Fig. 16: Flowchart of the LBP Process*

# 4. COMPARATIVE STUDY OF DIFFERENT FACIAL RECOGNITION METHODS

In this section we will see a brief comparison between different facial recognition techniques. Namely, Eigenfaces method versus Fisherfaces method and Eigenfaces method versus line edge map method.

## 4.1 Eigenfaces vs. Fisherfaces

To compare eigenfaces and fisherface method, we need to consider two important things: **between-class scatter** and **within-class scatter**.

- **Between-class scatter**: Suppose there are m classes. Then the scatter among the classes, say, $C_1, C_2, \ldots, C_n$, is the between-class scatter. The formula for the between-class scatter matrix is:

$$\mathbf{S}_b = \sum_{k=1}^{m} n_k(\boldsymbol{\mu}_k - \boldsymbol{\mu})(\boldsymbol{\mu}_k - \boldsymbol{\mu})^T$$

where, m is the number of classes, $\boldsymbol{\mu}$ is the overall sample mean, and $n_k$ is the number of samples in the kth class.

- **Within-class scatter**: Suppose there is a class with n elements. Then the scatter among each of the $x_1, x_2, \ldots, x_n$ elements is the within-class scatter. The formula for the within-class scatter matrix is:

$$\mathbf{S}_w = \sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu}_{y_i})(\mathbf{x}_i - \boldsymbol{\mu}_{y_i})^T$$

Where, $\boldsymbol{\mu}_k$ is the sample mean of the kth class and $y_i$ are the class labels.

The basic difference can be understood from the picture given alongside:

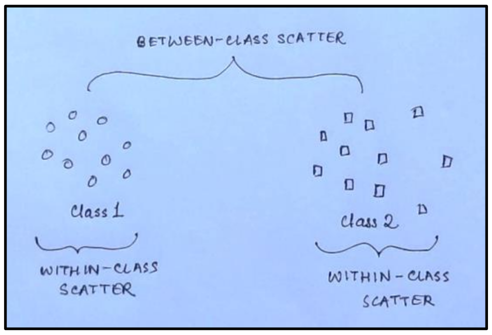


*Fig. 17: Difference between between-class and within class scatter; clearly the within-class scatter in class 2 is greater than in class 1.*

In simple terms, scatter can be described as distance between components.

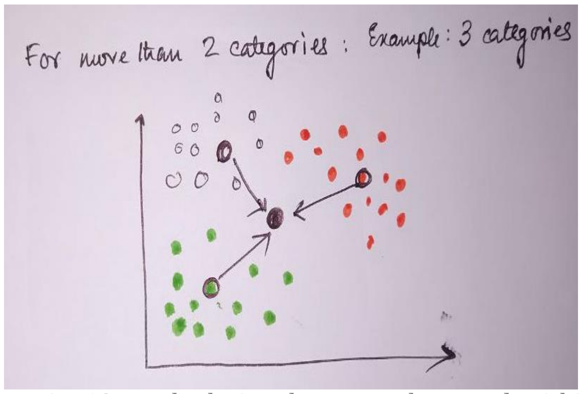


*Fig 18.: calculating between-class and within-class scatter for more than 2 categories*

Consider the image(fig 18), there are 3 categories (green, red and black) mapped on an XY plane.

- Consider a point central to all of the data (say M) (The large black dot). Each category, (green, red and black) has their own respective central points (say x, y, z). These individual central points are nearly equidistant from other points in their respective categories. So x, y, z are also nearly equidistant from M.

- Now, maximize the distance between each category's central point(x, y, z) and the data central point (M). This is called between-class scatter as different categories are involved. Simultaneously, minimize the distance between each category's points and their central point (like for class X whose central point is x and points are $\{x_1,x_2,x_3,\ldots\}$ we take distance between x and $x_i$). This is within-class scatter for each category.

As described earlier in this paper, for **Eigenfaces** method, we consider a set of n sample images $\mathbf{\{x_1,x_2, \ldots,x_n\}}$, taking values in an n-dimensional image space, and assume that each image belongs to one of c classes $\mathbf{\{X_1,X_2,\ldots,X_c\}}$ . We also consider a **linear transformation** mapping the original n-dimensional image space into an m-dimensional feature space, where $\mathbf{m < n}$. The images are the **basis** of the original matrix. Linear transformation is defined on these as $\mathbf{y_k=W^Tx_k}$.

Then after applying the linear transformation, the scatter of the transformed feature vectors $\mathbf{\{y_1,y_2,\ldots, y_N\}}$ is $\mathbf{W^TS_TW}$ where $\mathbf{S_T}$ is the total scattered matrix.

In PCA, the projection $W_{opt}$ is selected to maximize the total scatter matrix's determinant of the projected samples. The scatter being maximized is due to **both between-class and within-class scatter**. Now, between-class scatter is useful for classification but within-class scatter is unwanted information. Resources are required for running this algorithm and a greater amount of wasted, unwanted information would mean waste of resources. So, this is a drawback of this approach.

Variations in images also include illumination changes. Thus if Principal Component Analysis is presented with images of faces under varying illumination, the projection matrix $\mathbf{W}_{opt}$ will contain principal components (that is, Eigenfaces) which retain the variation due to lighting even in the projected feature space. As a result, the points in the projected space will not be well clustered. Therefore, important information useful for discrimination may be lost.

**Fisherfaces** method is based on **Fisher's Linear Discriminant(FLD).** The main idea of FLD is to find a projection to a line such that samples from different classes are well separated. It maximizes the ratio of between-class scatter to that of within-class scatter. FLD achieves a greater between-class scatter than PCA which makes the classification simplified and also amount of unwanted information is reduced.

In short, the basic difference between PCA and FLD is that PCA optimizes the transformation matrix by finding the largest variations in the original feature space whereas, FLD pursues the largest ratio of between-class variation and within-class variation when projecting the original feature to a subspace, i.e, it finds a line of projection that allows for maximum separation between known categories.

After applying these methods on datasets(as seen in the research)[11], it was concluded that fisherfaces method is more accurate for classification over variation in lighting and expression.

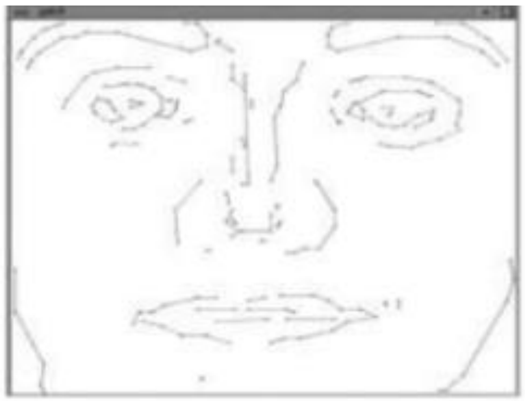
## <u>4.2 Eigenfaces vs. Line edge map</u>

As already discussed, the eigenfaces algorithm calculates the eigenvectors of the covariance matrix of the set of face images.

### Face recognition using line edge map

This algorithm describes a new technique based on line edge maps(LEM) and proposes a line matching technique to make this task possible. LEM uses **physiologic features from human faces** to solve the problem(mouth, nose and eyes).The facial images are first transformed to gray-level images in order to calculate the similarity of human faces. Gray-level picture refers to the black and white image, that is, the monochrome image.

Then these gray level images are encoded into binary edge maps using algorithm called **Sobel Edge Detection algorithm [12]**. Edges in image usually refer to the areas with strong intensity contrast. In simple words, they refer to a jump or sudden change in the intensity level of one pixel to the next.



The Sobel edge detection algorithm is used to encode images into a binary edge map. The algorithm uses a Sobel filter to detect the edges. It works by calculating the gradient of image intensity at each pixel in the image and returns edges at those points where the gradient, i.e, the steepness, of the image is maximum.

An important part of the algorithm is the **Line Segment Hausdorff Distance (**LHD) described to accomplish an accurate matching of face images. The main characteristic of this algorithm is its **flexibility of size, position and orientation**. Given two LEMS $M^1$ which contains the test face database and $T^1$ which contains the input images, the LHD is represented by a **vector** $\mathbf{d(m^1_i, t^1_j)}$. The elements of this vector represent themselves in three different distance measurements: **orientation distance, parallel distance** and **perpendicular distance** respectively. The **function** $\boldsymbol{\theta(m^1_i, t^1_j)}$ represents the smallest intersection angle between lines $m^1_i$ and $t^1_j$.

### Conversion of image into matrix using LEM

First, all unnecessary background area in the image is removed. Next, the face is cropped out from the image. **Difference of Gaussian** is applied to cropped image for edge detection[13].The difference of Gaussian which is a feature enhancement algorithm. It involves the subtraction of a blurred one vision of an ingenious image from another un-blurred version of the first. Then the edges are detected.

*< Fig. 19. Example of a face using LEM. The features are visible but quite simplified.*

After this an adjacency matrix generated and if edge is present in a grid, its value is detected as 1 and absence of it is detected as 0.

The process of edge detection significantly reduces the amount of data and filters out unused information. Thereafter it presents the important structural properties of an image.

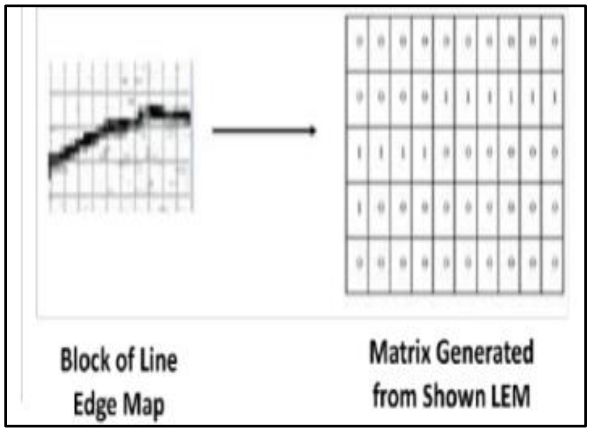


*Fig. 20. Presence of edge is evaluated to 1 and absence of it is evaluated to 0 in the adjacency matrix.*

The advantage of this method is that by measuring the parallel distance, we choose the **minimum distance between edges**. It helps when line edge is strongly detected and the other one not. Yet it also has a weakness; it can confuse lines momentarily, and not detect similarities that should be detected. This method's sensitiveness to illumination changes is low, and the algorithm has low memory requirements of because the kind of data used.

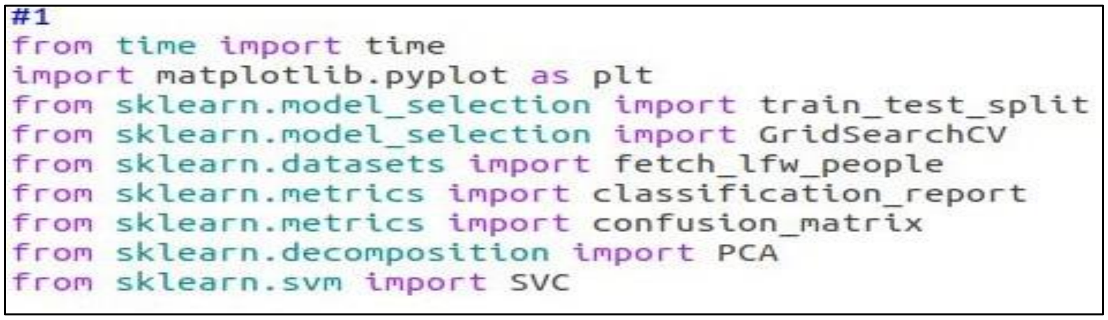
Deriving results from the research[12] we can draw the following comparison between the two methods:

- With eigenface method, high level of accuracy in recognition can be achieved but the quantity of image rejected as unknown increases.
- For the same method, the results show that there is not very much changes with lighting variations; whereas size changes make accuracy fall very quickly as compared to LEM which showed high level of accuracy.
- The LEM algorithm held high rates of correct recognitions for lighting variations.
- LEM is based on face features, whereas eigenfaces uses correlation and eigenvector to do so.

# 5. APPROACH TOWARDS FACIAL RECOGNITION PROGRAMMING IN PYTHON

First, we collected data which is a preprocessed excerpt of the "Labeled Faces in the Wild", aka LFW[14].

Then import packages.



```
#1
from time import time
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.datasets import fetch_lfw_people
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.decomposition import PCA
from sklearn.svm import SVC
```

**time:** It is used to know how much time is taken for executing each step.

**matplotlib.pyplot:** It is a plotting library for the python programming. Its command style functions that make matplotlib works like MATLAB.

**sklearn**: It is a free machine learning library for python.

**train_test_split**: It splits the arrays into training and testing.

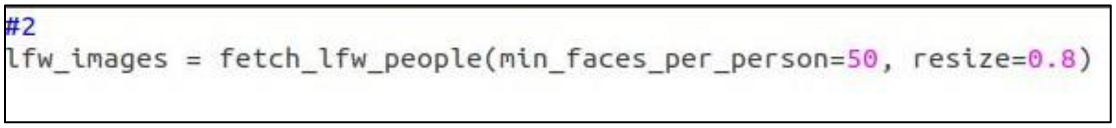
**GridSearchCV**: A machine learning model.

**fetch_lfw_people**: It is loading the lfw dataset.

**classification_report and confusion_matrix**: They are used for giving the proper summary of the output.

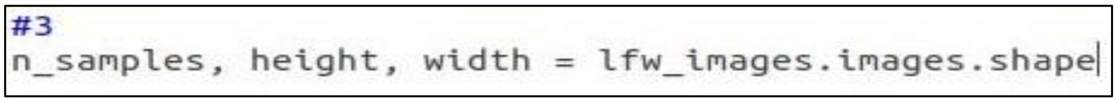
**PCA:** It implements the theoretical algorithm of PCA.

**SVC**: It is used for fitting the data and in return gives the best fit data.



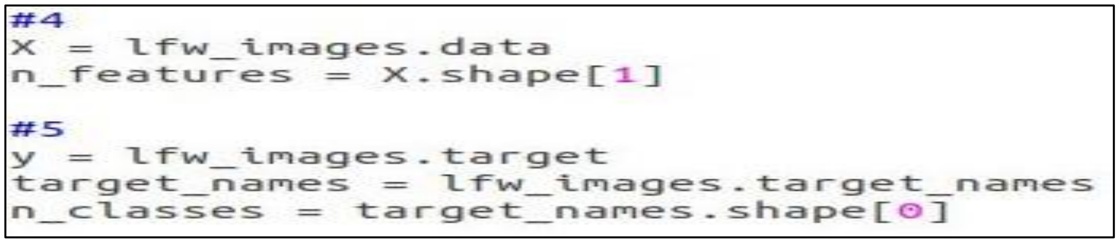
```
#2
lfw_images = fetch_lfw_people(min_faces_per_person=50, resize=0.8)
```

It extracts 50 images of each person from the dataset and resize them to 0.8 of their original value and this is stored in lfw_images.



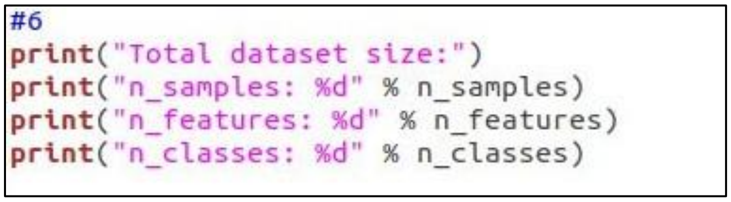
```
#3
n_samples, height, width = lfw_images.images.shape
```

It gives the shape of our dataset.



```
#4
X = lfw_images.data
n_features = X.shape[1]

#5
y = lfw_images.target
target_names = lfw_images.target_names
n_classes = target_names.shape[0]
```

Target function is applied and is stored in y.

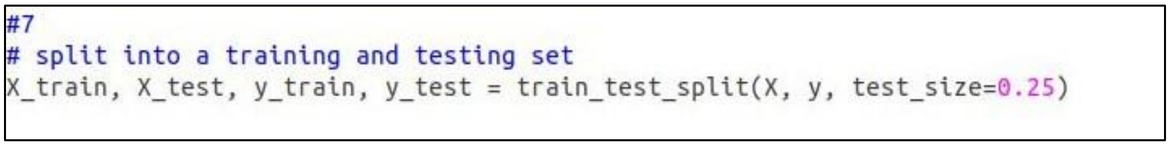


```
#6
print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)
```

**Output of above statement:**

```
Total dataset size:
n_samples: 1560
n_features:  7500
n_classes:    12
```

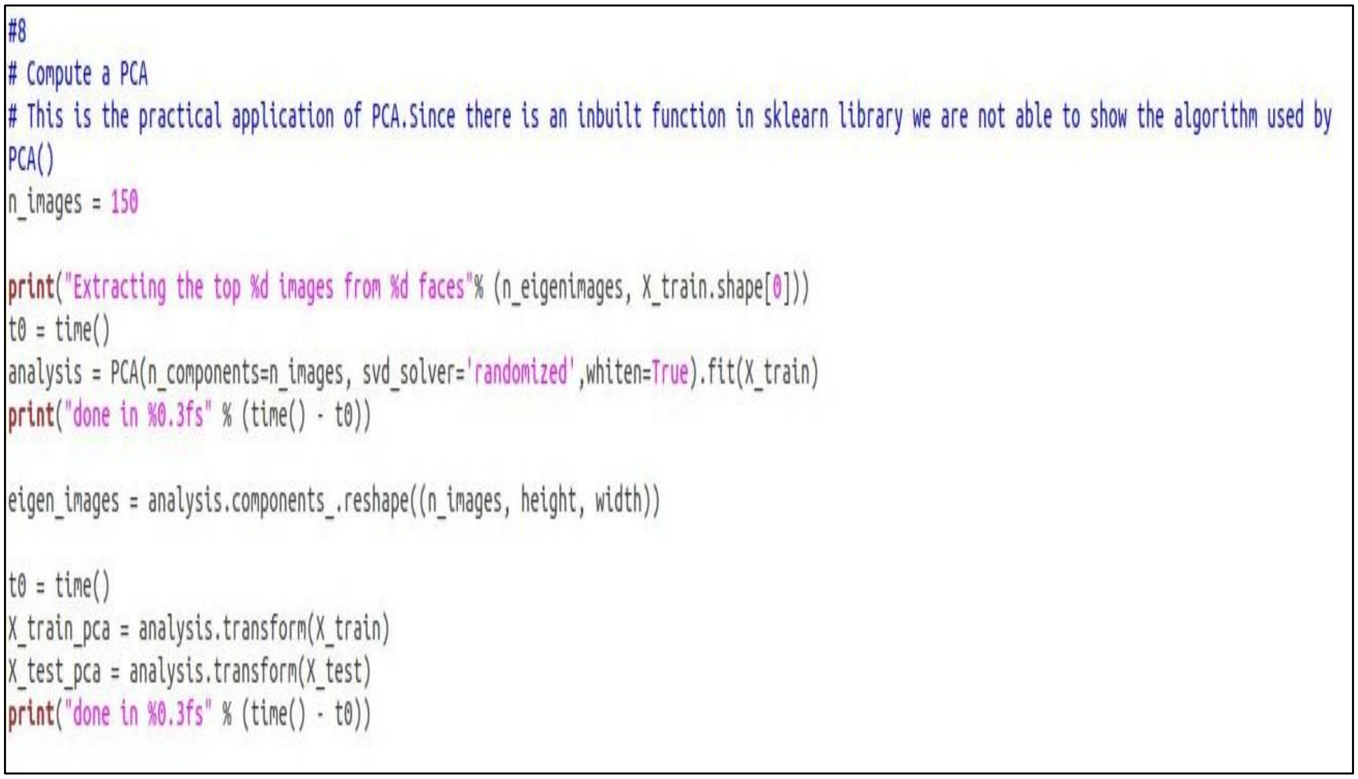
## <u>5.1 Splitting the dataset</u>



```
#7
# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In code 4 and 5 we defined X and y. X is the data that has the features which is used for the prediction of y also called target data. In this part of the code we are splitting X and y using the function train_test_split with the test size of 0.25.

## 5.2 Computing PCA and Eigenfaces



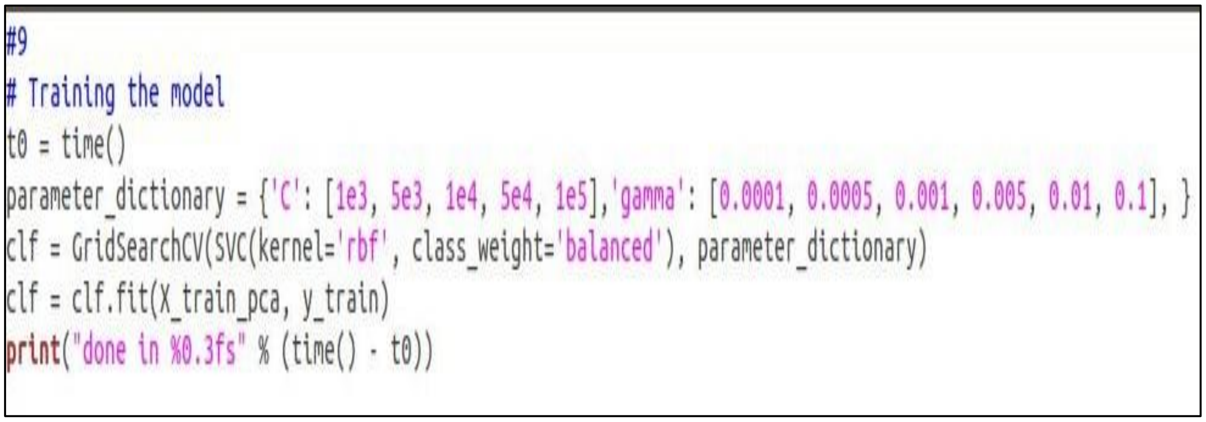
```
#8
# Compute a PCA
# This is the practical application of PCA.Since there is an inbuilt function in sklearn library we are not able to show the algorithm used by
PCA()
n_images = 150

print("Extracting the top %d images from %d faces"% (n_eigenimages, X_train.shape[0]))
t0 = time()
analysis = PCA(n_components=n_images, svd_solver='randomized',whiten=True).fit(X_train)
print("done in %0.3fs" % (time() - t0))

eigen_images = analysis.components_.reshape((n_images, height, width))

t0 = time()
X_train_pca = analysis.transform(X_train)
X_test_pca = analysis.transform(X_test)
print("done in %0.3fs" % (time() - t0))
```

The above code takes 150 images from the X_train dataset and stores in the variable called n_images.

After this we determine the PCA of n_images. This is done by using the function PCA() which takes the following parameters:

- **n_components**: Number of components to keep
- **svd_solver**: It's a machine learning parameter and takes randomized value.
- **whiten**: It takes the value True which ensures that the outputs achieved aren't correlated.
- **fit(X_train)**: It is a machine learning function which is implementing PCA on X_train dataset.

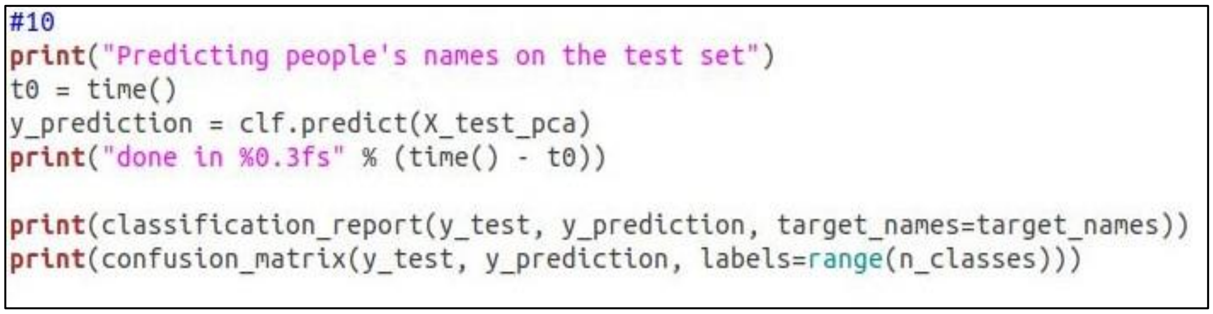
The result is stored in a variable called **analysis.** Then we compute eigenfaces using analysis variable and function called components_reshape. The function changes the height and width of the result we obtained using PCA(). This is stored in the variable called eigen_images. After PCA is done we modify our training dataset and testing dataset using **transform().**



```
#9
# Training the model
t0 = time()
parameter_dictionary = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(SVC(kernel='rbf', class_weight='balanced'), parameter_dictionary)
clf = clf.fit(X_train_pca, y_train)
print("done in %0.3fs" % (time() - t0))
```

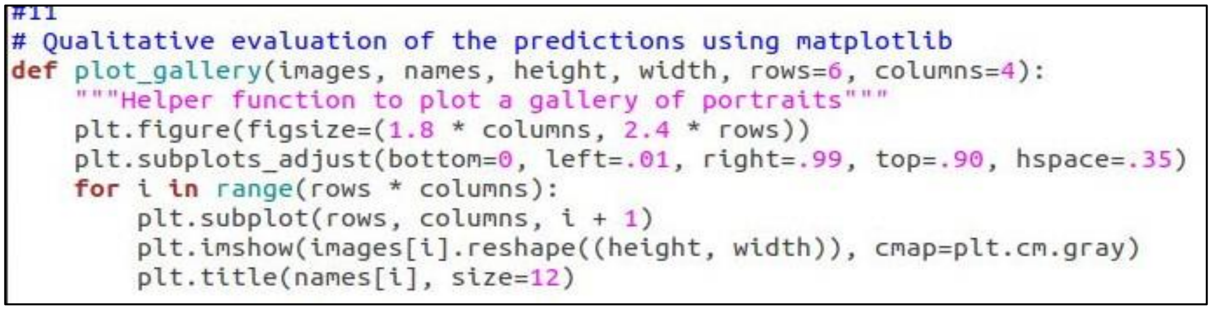
The code 9 is used for training the dataset X_train_pca. We have a parameter dictionary with defined set of values of 'C' and 'gamma'. These values help in providing the accurate output. Then clf which is name for the model, is defined which stores the output received using the model type **GridSearchCv**. Then again the value of clf is changed and now it stores the output of the model for the dataset **X_train_pca** and **y_train**.

## 5.3 Evaluation of the model on the test set



```
#10
print("Predicting people's names on the test set")
t0 = time()
y_prediction = clf.predict(X_test_pca)
print("done in %0.3fs" % (time() - t0))

print(classification_report(y_test, y_prediction, target_names=target_names))
print(confusion_matrix(y_test, y_prediction, labels=range(n_classes)))
```

The code is testing our training algorithm. A classification report and confusion matrix is formed.



```
#11
# Qualitative evaluation of the predictions using matplotlib
def plot_gallery(images, names, height, width, rows=6, columns=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * columns, 2.4 * rows))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(rows * columns):
        plt.subplot(rows, columns, i + 1)
        plt.imshow(images[i].reshape((height, width)), cmap=plt.cm.gray)
        plt.title(names[i], size=12)
```

This code 11 uses **matplotlib** library. It is used for using MATLAB like

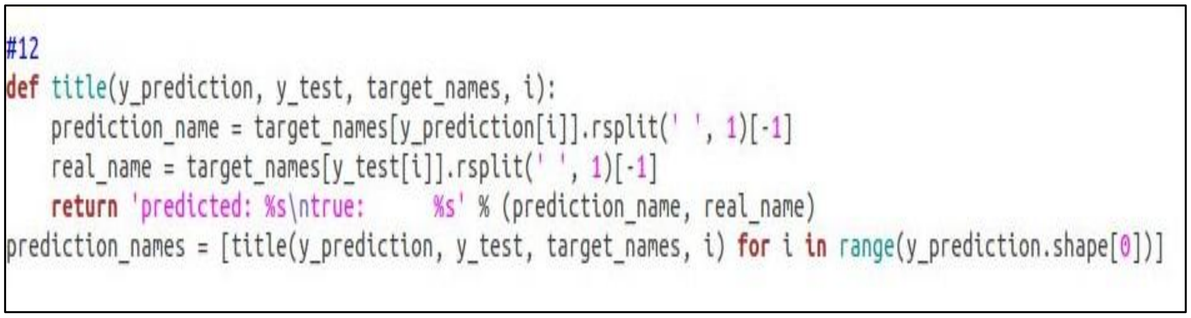
interface in python. **plot_gallery** is a user defined function having images, names, height, width, rows=6 and columns=4 as parameter.

**images** is used as an object for **fetch_lfw_people()** class in segment 2 and is getting values from eigen_images, predicted_images, names are getting it's values from **title** function which will be explained in segment 12.

We are using **figure()** which will plot our images according to the parameters provided. The **plot.subplot_adjust()** is used to tune our images from edges, bottom or used for adjusting the space between the images. 'for' loop is used when we want to iterate on a sequence/range. Here range is rows x columns and the variable is 'i'.

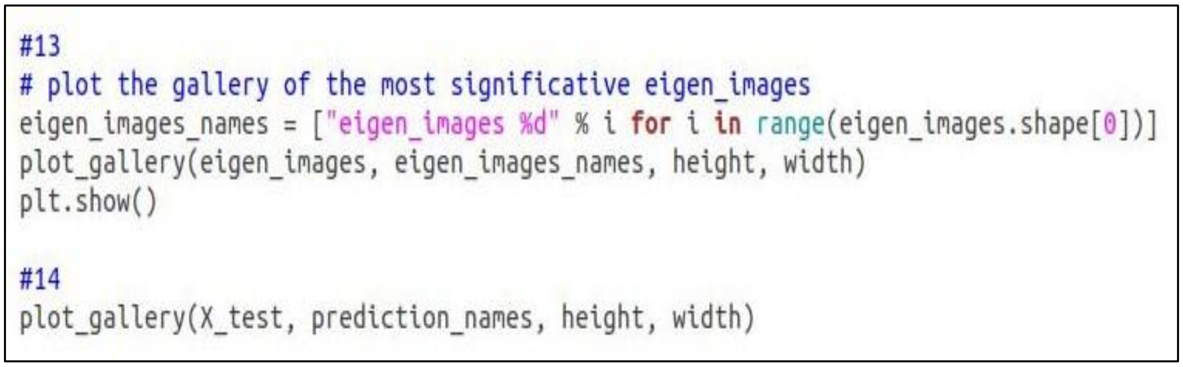
Inside the loop we have three functions:

- **plt.subplot()** is used to create common layout in a single cell.
- **plt.imshow()** is used to show data as an image, here data=parameters of the function, the first parameter **reshape()** gives a new shape to an array without changing its data and the second parameter is the font-size of the text.
- **plt.title()** returns a string that represents text itself. The first parameter is actual title text string, the second paramtere is the font size of the text.



```
#12
def title(y_prediction, y_test, target_names, i):
    prediction_name = target_names[y_prediction[i]].rsplit(' ', 1)[-1]
    real_name = target_names[y_test[i]].rsplit(' ', 1)[-1]
    return 'predicted: %s\ntrue:      %s' % (prediction_name, real_name)
prediction_names = [title(y_prediction, y_test, target_names, i) for i in range(y_prediction.shape[0])]
```

This code is again a user defined function with name **title** with parameters **y_prediction, y_test,target_names,i**. they were already defined in segment 10, 7 and 5 respectively. This function has 2 variables **prediction_name** and **real_name**. As the name suggests, 1st variable contains predicted names of the image with split() while 2nd variable contains real names. We call this function from the variable where 'i' have for loop, which will give values of each row of y_prediction.

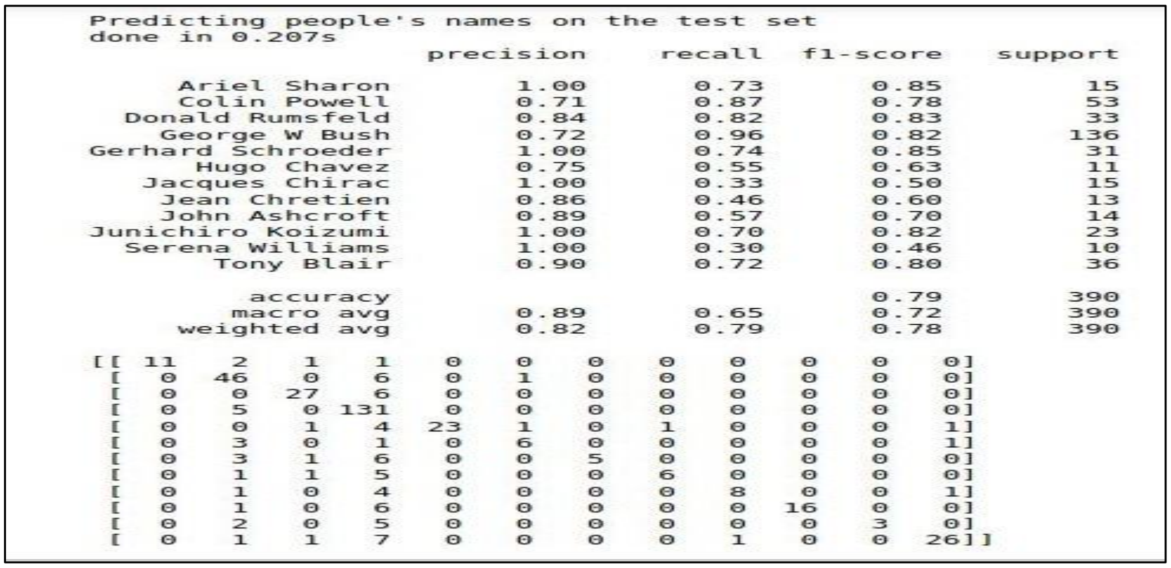


```
#13
# plot the gallery of the most significative eigen_images
eigen_images_names = ["eigen_images %d" % i for i in range(eigen_images.shape[0])]
plot_gallery(eigen_images, eigen_images_names, height, width)
plt.show()

#14
plot_gallery(X_test, prediction_names, height, width)
```

We are storing the values of each row of the eigen_images in eigen_images_names and then calling our user defined function plot_gallery with it and displaying all images with the plt.show(). We are calling user defined function **plot_gallery** for prediction_names, i.e, for getting names from code 12.

## <u>5.4 Outputs</u>

This is the output from code 10.



```
Predicting people's names on the test set
done in 0.207s
                   precision    recall  f1-score   support

     Ariel Sharon       1.00      0.73      0.85        15
     Colin Powell       0.71      0.87      0.78        53
  Donald Rumsfeld       0.84      0.82      0.83        33
    George W Bush       0.72      0.96      0.82       136
Gerhard Schroeder       1.00      0.74      0.85        31
      Hugo Chavez       0.75      0.55      0.63        11
   Jacques Chirac       1.00      0.33      0.50        15
    Jean Chretien       0.86      0.46      0.60        13
    John Ashcroft       0.89      0.57      0.70        14
Junichiro Koizumi       1.00      0.70      0.82        23
  Serena Williams       1.00      0.30      0.46        10
       Tony Blair       0.90      0.72      0.80        36

         accuracy                           0.79       390
        macro avg       0.89      0.65      0.72       390
     weighted avg       0.82      0.79      0.78       390

[[ 11   2   1   1   0   0   0   0   0   0   0   0]
 [  0  46   0   6   0   1   0   0   0   0   0   0]
 [  0   0  27   6   0   0   0   0   0   0   0   0]
 [  0   5   0 131   0   0   0   0   0   0   0   0]
 [  0   0   1   4  23   1   0   1   0   0   0   1]
 [  0   3   0   1   0   6   0   0   0   0   0   1]
 [  0   3   1   6   0   0   5   0   0   0   0   0]
 [  0   1   1   5   0   0   0   6   0   0   0   0]
 [  0   1   0   4   0   0   0   0   8   0   0   1]
 [  0   1   0   6   0   0   0   0   0  16   0   0]
 [  0   2   0   5   0   0   0   0   0   0   3   0]
 [  0   1   1   7   0   0   0   0   1   0   0  26]]
```

**Confusion matrix:** It is used to estimate the accuracy of the algorithm applied in the field of machine learning. Each row in the confusion matrix is called as class.

The number of correct values in each class lies in the diagonal of confusion matrix. Hence there are 12 classes in this confusion matrix. The diagonal of this matrix tells us the correct output we have achieved using the applied algorithm.

For example:

In class 1 we have tota l 16 images. Out of 16 images 11 images were correctly recognised.

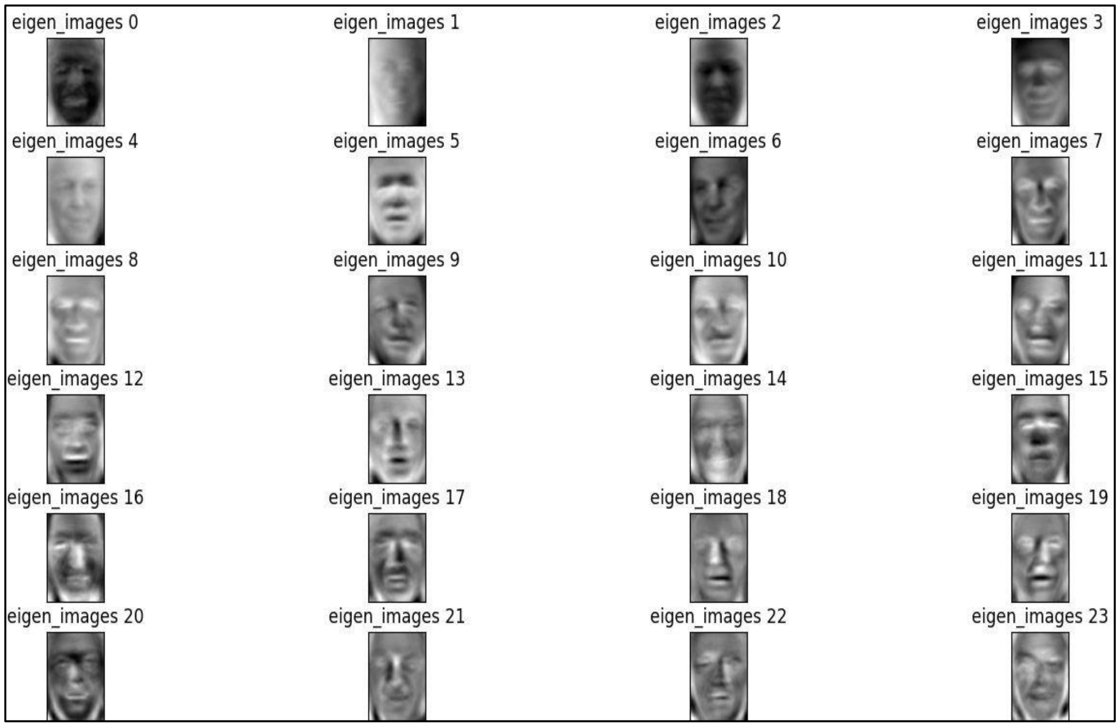
In class 2 we have total 63 images. Out of 63 images , 55 images were correctly recognised

The best model is the one which yields the diagonal matrix.

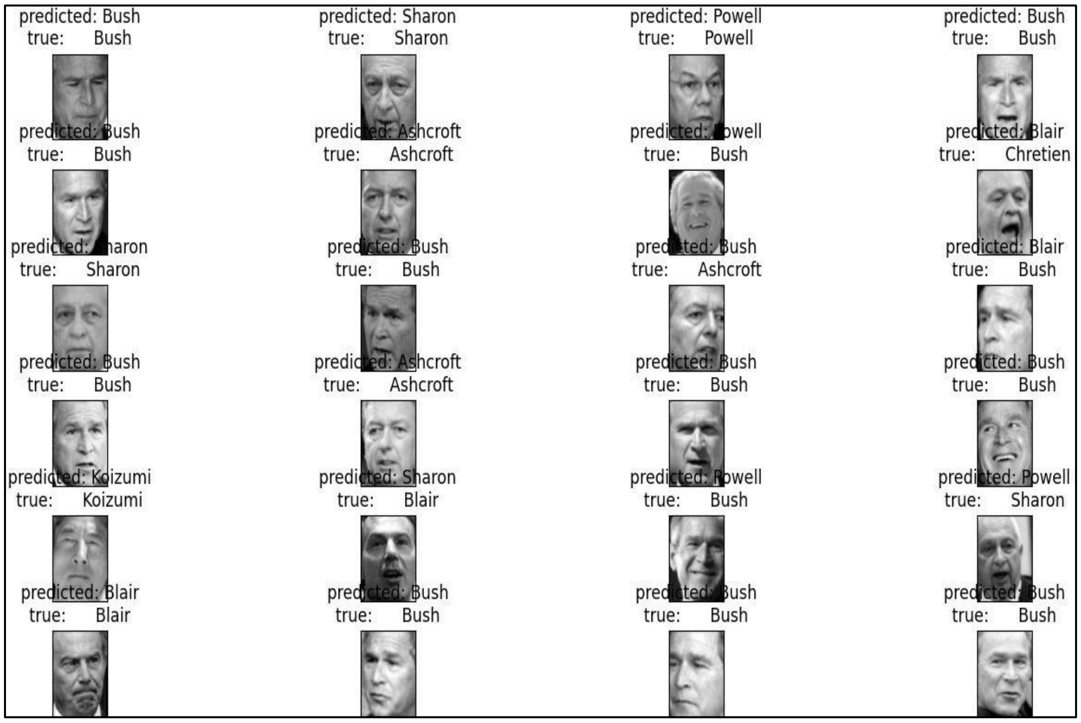
<u>**Precision , Recall , f1-score** and **Support**</u>

1) **Precision:** It is the ability of the program not to yield the false output.
2) **Recall**: It is the ability of the program to yield all the positive samples
3) **F1-score**: It is the harmonic mean of precision and recall.

4) **Support:** It is the number of occurrences in the dataset that predicted the values correctly.



Eigen images from code 13(up)
Prediction images from code 14(down)



## 6. SCOPE and CHALLENGES

With increasing security concerns in every sector, the Face Recognition System has become very popular. Nevertheless, due to some of the system's disadvantages, it doesn't fulfill all user requirements as face lock in android mobiles can be fooled by the picture of the person. This biometric system has the potential for future changes that will increase the degree of safety; thus the essential information will be more protected. This system has scope of development particularly during the covid-19 situation where most of the work is carried out online and protection or secrecy is the most important factor for all, such as online exams and online attendance. This can be further developed for use in ATMs, lockers, banks, and for criminal identification etc. There are other challenges including lighting conditions impacting certain devices, various angles, various gestures, etc which needs to be improved. Furthermore, in this pandemic situation where wearing masks has become the most important thing, the system should be designed in such a way that the person with the masks can also be identified by using the person's hairline details and eye distance etc. This system would be better to use instead of a PIN or fingerprint particularly in this pandemic situation where one should avoid touching items. Thus, it will be a good starting point for research projects on face recognition as it has the scope for improvement and can be used in many fields and increases security level.

## CONCLUSION

The aim of this paper was to study the use of one of the most important topics of linear algebra i.e., eigenvalues and eigenvectors in the field of facial recognition.

A thorough study of Eigenfaces and comparing the method with others like Linear Binary Pattern, Fisherfaces and Line Edge Map, has brought forth a number of advantages and drawbacks of the system. Perfecting this approach still remains a challenge. Eigenfaces method is advantageous as raw data is used and no knowledge of facial geometry is required. This is because it uses correlation and eigenvectors for recognition, whereas in a method like LEM the facial features are used directly. PCA’s dimensionality reduction is an advantage. Additionally, an approach like LBP is more like a patch descriptor which is limited to images while PCA can be extended to audio as well. Overall, the recognition is simple and efficient compared to other matching approaches.

However, the experimental results and the research carried out in this paper also demonstrate some serious limitations of eigenface method. One of the main shortcomings is that the method is very sensitive to scale, therefore, a low-level pre-processing is still necessary for scale normalization. Secondly, its recognition rate decreases for recognition under varying pose and illumination which can be solved by both LEM and fisherfaces. Next, there is the problem of dealing with expression and glasses. Our program’s image dataset consisted mainly of frontal views. Eigenface approach cannot handle that very accurately. Apart from this, a lot of unwanted information is generated unlike in LBP or Fisherfaces or LEM. Recognition is more efficient when the number of face classes is larger than the dimensions of the face space, i.e, when we take a larger database of images.

Other approaches mentioned in this paper, were found to be robust and produced accurate results, as understood from existing research. However, they have their own drawbacks as well along with huge scope for future work. There is much work to be done to realise methods that can mirror the way the human mind recognises faces.

## ACKNOWLEDGEMENTS

We wish to thank Miranda House, University of Delhi for organising the Summer Workshop for the purpose of investigative projects in multidisciplinary contexts that afforded us this incredible opportunity to not only learn about things outside the textbooks but also brainstorm ideas and contribute to existing research. This research work would not have been possible without the invaluable support of our mentors, Mr. Naveen Gupta, Ms. Apeksha Verma and Mr. Mohit Kumar Macchal, faculty members of the department of Mathematics, Miranda House, University of Delhi therefore, we would like to acknowledge their constant support.

## REFERENCES:

[1] https://en.wikipedia.org/wiki/Eigenface

[2] https://en.wikipedia.org/wiki/Vector_space

[3] Liton Chandra Paul, Abdulla Al Sumam, "*Face Recognition Using Principal Component Analysis Method*", International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume 1, Issue 9, November 2012

[4] M. Turk and A. Pentland, "*Eigenfaces for Recognition*", Journal of Cognitive Neuroscience, vol. 3, no. 1, pp. 71-86, 1991, hard copy

[5] Deval Shah, "*Face Recognition using eigenfaces technique.*", available at https://medium.com/@devalshah1619/face-recognition-using-eigenfaces-technique-f221d505d4f7

[6] T. Ojala, M. Pietikainen and D. Harwood, "*A comparative study of texture measures with classification based on feature distributions*" Pattern Recognition vol. 29, 1996.

[7] Kelvin Salton do Prado , "*Face Recognition: Understanding LBPH Algorithm*", available at https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

[8] D. He and L. Wang, "*Texture unit, texture spectrum and texture analysis,*" IEEE Transactions on Geosciences and Remote Sensing, vol. 28, no. 4, pp. 509-512, Jul 1990.

[9] T. Ahonen, A. Hadid and M. Pietikainen. "*Face recognition with local binary patterns,*" in Proc. ECCV, 2004, pp. 469–481.

[10] C.H. Chan, J. Kittler and K. Messer. "*Multi-scale local binary pattern histograms for face recognition,*" in International Conferences on Biometrics, 2007, pp. 809-818.

[11] Peter N. Belhumeur, Joao P. Hespanha, and David J. Kriegman. "*Eigenfaces vs. Fisherfaces: Recognition Using Class Specific Linear Projection*" in IEEE transactions on pattern analysis and machine intelligence, vol. 19, no. 7, july 1997.

[12] https://homepages.cae.wisc.edu/~ece533/project/f06/orts_rpt.pdf

[13] Rohil, Harish & Kaushik, Pankaj. (2014). "*Adjacency Matrix based Face Recognition Approach*". International Journal of Computer Applications. 98. 22-28. 10.5120/17299-7740.

[14] https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html

[fig2] Müge Çarıkçı , Figen Özen, Haliç University, Electrical and Electronics Engineering Department, Şişli, Istanbul, Turkey, "*A Face Recognition System Based on Eigenfaces Method*",

[fig3] https://pdf.sciencedirectassets.com/ Dhairya Parikh,"*Advancements in Computer based Facial recognition systems*", available at https://medium.com/coinmonks/from-the-rand-tablet-to-differentiating-identical-twins-aa4ba6031bb0

[fig5] https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html

[fig6a] http://www.sprawls.org/resources/DICHAR/module.htm

[fig6b] https://math.stackexchange.com/questions/3637803/how-to-use-independent-set-problem-for-image-edge-detection

[fig7] http://www.cs.unc.edu/~lazebnik/spring09/lec22_eigenfaces.pdf

[fig8] Marijeta Slavkovic, Dubravka R. Jevtic, "*Face recognition using eigenface approach*", Serbian Journal of Electrical Engineering, DOI:10.2298/SJEE1201121S, Published 2012

[fig9] Marijeta Slavkovic, Dubravka R. Jevtic, "*Face recognition using eigenface approach*", Serbian Journal of Electrical Engineering, DOI:10.2298/SJEE1201121S, Published 2012

[fig10] Marijeta Slavkovic, Dubravka R. Jevtic, "*Face recognition using eigenface approach*", Serbian Journal of Electrical Engineering, DOI:10.2298/SJEE1201121S, Published 2012

[fig11] http://www.cs.unc.edu/~lazebnik/spring09/lec22_eigenfaces.pdf

[fig13] https://link.springer.com/article/10.1007/s11042-020-08864-z

[fig14] Kelvin Salton do Prado , "*Face Recognition: Understanding LBPH Algorithm*", available at https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b https://globaljournals.org/GJCST_Volume13/1-Face-Recognition-using-Local.pdf

[fig16] Rahim, Md & Hossain, Md & Wahid, Tanzillah & Azam, Md. (2013). "*Face Recognition using Local Binary Patterns (LBP)*" available at https://www.researchgate.net/figure/Flowchart-of-the-LBP-Process-Flowchart-of-the-Proposed-System_fig4_331645500

[fig19] Rohil, Harish & Kaushik, Pankaj. (2014). "*Adjacency Matrix based Face Recognition Approach*". International Journal of Computer Applications. 98. 22-28. 10.5120/17299-7740.

[fig20] Rohil, Harish & Kaushik, Pankaj. (2014). "*Adjacency Matrix based Face Recognition Approach*". International Journal of Computer Applications. 98. 22-28. 10.5120/17299-7740.

**Additional References**:

- https://youtu.be/_lY74pXWlS8
- https://youtu.be/SaEmG4wcFfg
- Marijeta Slavković , Dubravka Jevtić, "*Face Recognition Using Eigenface Approach*", SERBIAN JOURNAL OF ELECTRICAL ENGINEERING Vol. 9, No. 1, February 2012, 121-130 121

**For the dataset**:

- http://vis-www.cs.umass.edu/lfw/lfw-funneled.tgz

**For program code**:

- https://pythonprogramming.net/facial-recognition-python/
- https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#introduction
- https://towardsdatascience.com/simple-face-detection-in-python-1fcda0ea648e
- https://scikit-learn.org/stable/auto_examples/applications/plot_face_recognition.html