FACE RECOGNITION USING PCA

DONE WITH THE PRINCIPLE OF DEEP LEARNING .AI

TEAM MEMBERS

*RAJYAVARDHAN GUPTA

*ARCHIT LADHANIA

Aim:

Using Machine Learning Algorithms and supporting flow charts, implement the Face Recognition using PCA (Principal component analysis) technique.

Objective:

The face is primary focus of the attention and plays a major role in identification and establishing the uniqueness of a person from the rest of human society. Security and authentication have become a very important part of any industry, one of the key methods to follow is face recognition. Face recognition is an effective means of authenticating an individual. The recognition system enables to detect changes in the face pattern of an individual to an appreciable extent, the recognition system can also tolerate local variations in the facial expression of an individual. Our approach towards face recognition is based on Principal component analysis (PCA). The system consists of a database of a set of facial patterns for each individual. The characteristic features called 'eigenfaces' are extracted from the stored images using which the system is trained for subsequent recognition of new images.

THEORY

Introduction:

Face is the most memorable part of the body in real life that makes it an important variable. Generally, humans can remember and recognize a person based on his face. However, face is one of a complex variable when viewed from the perspective of computer vision. Human faces have different features and characteristics of each person so that face recognition is very good to be applied in various areas, including entertainment, smart cards, information security etc. Face recognition is the task of identifying an already detected object as a known or unknown face. Often the problem of face recognition is confused with the problem of face detection Face Recognition on the other hand is to decide if the "face" is someone known, or unknown, using for this purpose a database of faces in order to validate this input face.

Although it is clear that people are good at face recognition, but it's not obvious that a human brain can encoded or decoded for every face. Human face recognition has been studied for more than twenty years. Developing a suitable program which can be used digitally in recognizing a face is a quite challenging task, because human faces are complex and are different from each other in every aspect. A big challenge is how to quantize facial features so that a computer should be able to identify a face. The study carried out by many researchers over the past several years indicates that certain facial characteristics are used by human beings to identify faces. F**ace recognition** is the challenge of classifying whose face is in an input image. This is different than **face detection** where the challenge is determining if there is a face in the input image. With face recognition, we need an existing database of faces. Given a new image of a face, we need to report the person's name.

The basic concept of the computer vision as useful information from a single image or a sequence of image, usually used the method the automatic extraction, analysis and learning. One of the best methods to perform facial recognition is to use the Principal Component Analysis (PCA) algorithm. PCA is probably the most popular multivariate statistical technique and it is used by almost all scientific disciplines. It is also likely to be the oldest multivariate technique. Multivariate analysis can simply be interpreted as a method associated with large variables in one or more experiments. This method can extract the necessary characteristic data from various data that we have. PCA is a multivariate technique that analyse a data table in which observations are described by several inter-correlated quantitative dependent variables.

This project will apply the human face recognition system using the eigenface approach. Eigenface is one of the facial recognition methods based on the Principal Component Analysis (PCA) algorithm. PCA involved a mathematical procedure to derive a set of features for face recognition. Face recognition stage begins with face detection process using cascade classifier method, face pre-process, collect and train the face detected and finally the face recognition.

Different approaches for face recognition:

There are two predominant approaches to the face recognition problem: Geometric (feature based) and photometric (view based). As researcher interest in face recognition continued, many different algorithms were developed, three of which have been well studied in face recognition literature. Recognition algorithms can be divided into two main approaches:

1. Geometric: Is based on geometrical relationship between facial landmarks, or in other words the spatial configuration of facial features. That means that the main geometrical features of the face such as the eyes, nose and mouth are first located and then faces are classified on the basis of various geometrical distances and angles between features.

2. Photometric stereo: Used to recover the shape of an object from a number of images taken under different lighting conditions. The shape of the recovered object is defined by a gradient map, which is made up of an array of surface normal (Zhao and Chellappa, 2006)

Popular recognition algorithms include:

1. Principal Component Analysis using Eigenfaces, (PCA)
2. Linear Discriminate Analysis,
3. Elastic Bunch Graph Matching using the Fisherface algorithm

In this project, we would focus on **Principal Component Analysis (PCA)** using Eigenfaces.

A naïve way of accomplishing this is to take the new image, flatten it into a vector, and compute the Euclidean distance between it and all of the other flattened images in our database.

There are several downsides to this approach. First of all, if we have a large database of faces, then doing this comparison for each face will take a while. Imagine that we're building a face recognition system for real-time use. The larger our dataset, the slower our algorithm. But more faces will also produce better results. We want a system that is both fast and accurate. For this, we'll use a neural network. We can train our network on our dataset and use it for our face recognition task.

There's an issue with directly using a neural network: images can be large. If we had a single $m \times n$ image, we would have to flatten it out into a single $m\dot{n} \times 1$ vector to feed into our neural network as input. For large image sizes, this might hurt speed! This is related to the second problem with using images as-is in our naïve approach: they are high-dimensional. (An $m \times n$ image is really a $m\dot{n} \times 1$ vector) A new input might have a ton of noise and comparing each and every pixel using matrix subtraction and Euclidean distance might give us a high error and misclassifications.

These issues are why we don't use the naïve method. Instead, we'd like to take our high-dimensional images and boil them down to a smaller dimensionality while retaining the essence or important parts of the image.

Dimensionality Reduction:

The previous section motivates our reason for using a dimensionality reduction technique. Dimensionality reduction is a type of unsupervised learning where we want to take higher-dimensional data, like images, and represent them in a lower-dimensional space.

All dimensionality reduction techniques aim to find some **hyperplane,** a higher-dimensional line, to *project* the points onto. We can imagine a *projection* as taking a flashlight perpendicular to the hyperplane we're project onto and plotting where the shadows fall on that hyperplane.

Principal Component Analysis (PCA):

PCA is a technique by which we reduce the dimensionality of data points and emphasize variation and bring out strong patterns in a dataset. It's often used to make data easy to explore and visualize. It is a classic tool widely used in the appearance-based approaches for dimensionality reduction and feature extraction in most of the pattern recognition application. Hau T. Ngo et al. described a flexible and efficient architecture for real-time face recognition system based on modular Principal component Analysis method in an environment of FPGA, they showed that modular PCA improves the accuracy of face Recognition when face images have varying expression and illumination. The architecture was able to perform face recognition in 11ms for a database with 1000 face images. The performance of PCA-based face recognition system is quite satisfactory.

For example, consider the space of all 20 by 30-pixel grayscale images. This is a 600-dimensional space because 600 data points are required to represent the values of the 600 pixels. But suppose we only consider images that are valid faces. Such images may lie in a small subspace of the set of all images. If this subspace is a hyperplane spanned by k vectors, then we can represent any point that lies on this plane with only a set of k coefficients. To use PCA to do pattern recognition, you first take a representative sample of data points and find the subspace with the desired dimensionality that best fits the sample data. This step needs to be done only once. Once it has been done, you can determine if a new point is a valid data point by projecting it onto the subspace and measuring the distance between the point and its projection.

Principal component analysis to efficiently represent the pictures of faces. Any face image could be around reconstructed by a small collection of weights for each face and a standard face picture, that is, eigen picture. The weights are obtained by projecting the face image onto the eigen picture. In mathematics, Eigenfaces are the set of eigenvectors which are the set of feature vector or characteristic used in the computer vision problem of human face recognition. The principal component of the distribution of faces or the eigenvectors of the covariance matrix of the set of face image is the Eigenfaces. Each face can be represented exactly by a linear combination of the Eigenfaces. The best M eigenfaces construct an M dimension (M-D) space that is called the “face space” which is same as the image space. PCA for face recognition is based on the information theory approach in which the relevant information in a face image is extracted as efficiently as possible. Further ANN was used for classification. Neural Network notion is used because of its ability to learn from observed data.

Suppose each data point is N-dimensional:

Same procedure applies:

$$\begin{aligned} var(\mathbf{v}) &= \sum_{\mathbf{x}} \|(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}} \cdot \mathbf{v}\| \\ &= \mathbf{v}^{\mathbf{T}}\mathbf{A}\mathbf{v} \quad \text{where } \mathbf{A} = \sum_{\mathbf{x}} (\mathbf{x}-\overline{\mathbf{x}})(\mathbf{x}-\overline{\mathbf{x}})^{\mathbf{T}} \end{aligned}$$

• The eigenvectors of A define a new coordinate system

– eigenvector with largest eigenvalue captures the most variation among training vectors x

– eigenvector with smallest eigenvalue has least variation

• We can compress the data by only using the top few eigenvectors

– corresponds to choosing a "linear subspace" » represent points on a line, plane, or "hyper-plane"

– these eigenvectors are known as principal component vectors

Advantage and disadvantage of PCA

The advantages of PCA are listed below:

1) Lack of redundancy of data given the orthogonal components.

2) Reduced the complexity in face images grouping with the use of PCA

3) Smaller database representation since only the trainee images are stored in the form of their projections on a reduced basis.

4) Noise reduction since the maximum variation basis is chosen and so the small variations in the back ground are ignored automatically.

5)2DPCA over 1DPCA is that the feature vector is now two-dimensional so the problem of dimensionality is greatly reduced.

Disadvantages of PCA are:

1) The covariance matrix is difficult to be evaluated in an accurate manner.

2) The simplest invariance could not be captured even by the PCA unless the training data explicitly provides this information.

3) PCA is a less sensitive to different training data set.

4) Computationally expensive and complex with the increase in data size.

5) Time complexity is high.

Implementation

Approach:

The step by step instructions along with the formulas for the recognition of faces using Principal Component Analysis (PCA) are as follows:

STEP 1: Prepare the Data

The first step is to obtain a set S with M face images. Each image is transformed into a vector of size N and placed into the set.

$$S = \{\Gamma_1, \Gamma_2, \Gamma_3, \,\ldots\ldots\ldots\, ,\Gamma_M\}$$

STEP 2: Obtain the Mean

After obtaining the set, the mean image Ψ has to be obtained as,

$$\Psi = \frac{1}{M}\sum_{n=1}^{M}\Gamma_n$$

STEP 3: Subtract the Mean from Original Image

The difference between the input image and the mean image has to be calculated and the result is stored in Φ

$$\Phi_i = \Gamma_i - \Psi$$

.

STEP 4: Calculate the Covariance Matrix

The covariance matrix C is calculated in the following manner:

$$C = \frac{1}{M}\sum_{n=1}^{M} \Phi_n \Phi_n^T$$

$$= AA^T$$

$$A = \{\Phi_1, \Phi_2, \Phi_3, \ldots\ldots, \Phi_n\}$$

STEP 5: Calculate the Eigenvectors and Eigenvalues of the Covariance Matrix and Select the Principal Components

In this step, the eigenvectors (eigenfaces) ui and the corresponding eigenvalues λi should be calculated. From M eigenvectors, u, only M' should be chosen, which have the highest eigenvalues. The higher the eigenvalue, the more characteristic features of a face do the particular eigenvector describe. Eigenfaces with low eigenvalues can be omitted, as they explain only a small part of the characteristic features of the faces. After M' eigenfaces are determined, the "training" phase of the algorithm is finished. Once the training set has been prepared the next phase is the classification of new input faces. The recognition procedure consists of two major steps:

Transform the New Face The new face is transformed into its eigenface components and the resulting weights form the weight vectors.

$$\omega_k = u_k^T(\Gamma - \Psi)$$

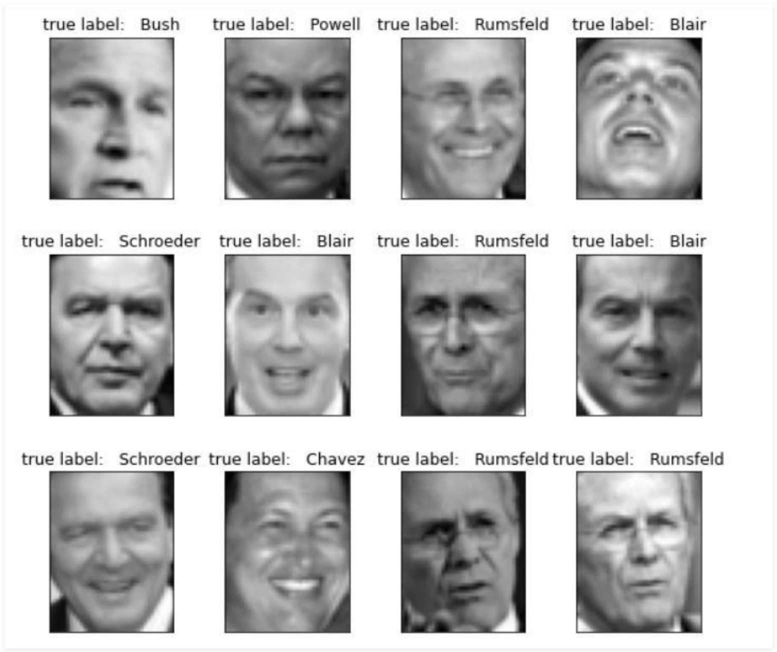
where **ω** = weight, **μ** = eigenvector, **Γ** = new input image, **Ψ** = mean face

The weight vector $\mathbf{\Omega}^{\mathrm{T}}$ is given by,

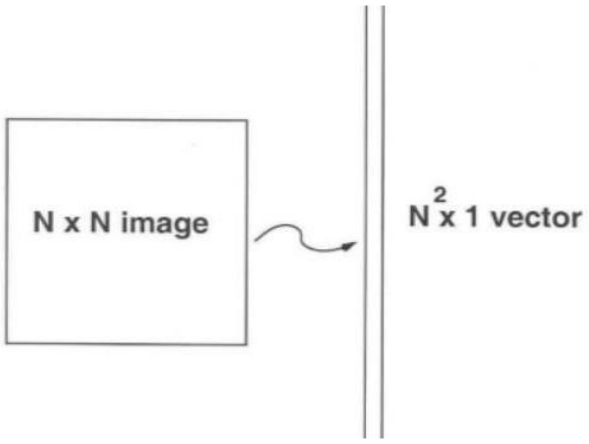
$$\Omega^T = [\omega_1, \omega_2, \ldots\ldots, \omega_M]$$

Training the Algorithm:

- Let's Consider a set of *m* images of dimension *N*N* (training images):



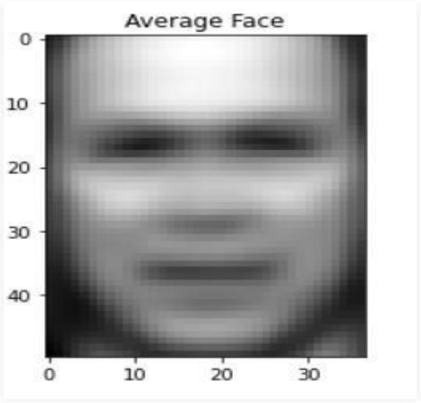
- We first convert these images into vectors of size N^2.



- Now we calculate the average of all these face vectors and subtract it from each vector.

$$\psi = \frac{1}{m} \sum_{i=1}^{m} x_i$$

$$a_i = x_i - \psi$$



- Now we take all face vectors so that we get a matrix of size of $N^2 * M$.
- Now, we find Covariance matrix by multiplying A with A^T. A has dimensions $N^2 * M$, thus A^T has dimensions $M * N^2$. When we multiplied this gives us matrix of $N^2 * N^2$, which gives us N^2 eigenvectors of N^2 size which is not computationally efficient to calculate. So we calculate our covariance matrix by multiplying A^T and A. This gives us $M * M$ matrix which has M *(assuming* $M << N^2$*)* eigenvectors of size M.
- In this step we calculate eigen values and eigen vectors of above covariance matrix using the formula below.

$$A^T A \nu_i = \lambda_i \nu_i$$

$$A A^T A \nu_i = \lambda_i A \nu_i$$

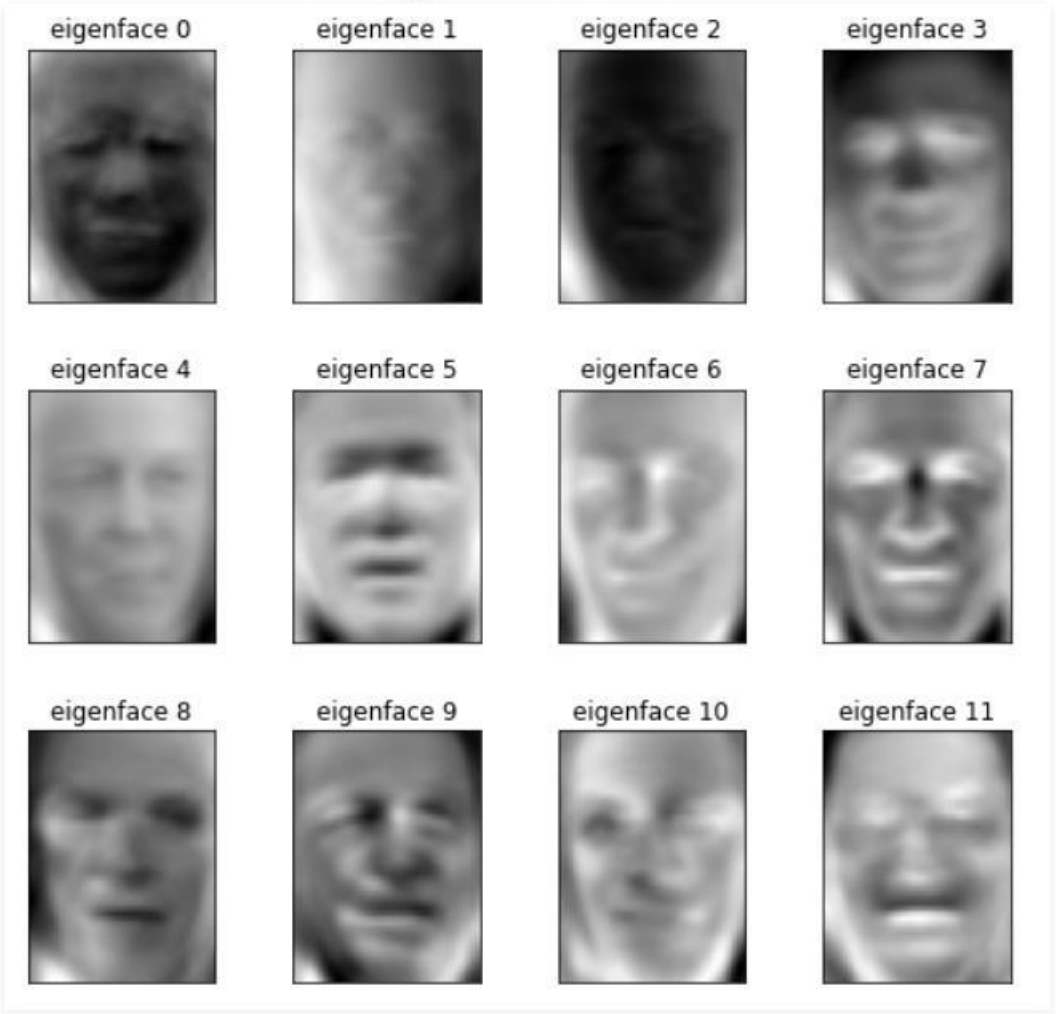
$$C' u_i = \lambda_i u_i$$

Where, $C' = AA^T$ and $u_i = A\nu_i$

- Now we calculate Eigenvector and Eigenvalues of this reduced covariance matrix and map them into the C` by using the formula.
- Now we select the K eigenvectors of C` corresponding to the K largest eigen values (where K < M). These eigen vectors have size N^2.
- In this step we used the eigenvectors that we got in previous step. We take the normalized training faces (face – average face) Xi and represent each face vectors in the linear of combination of the best **K** eigenvectors (as shown in the diagram below).

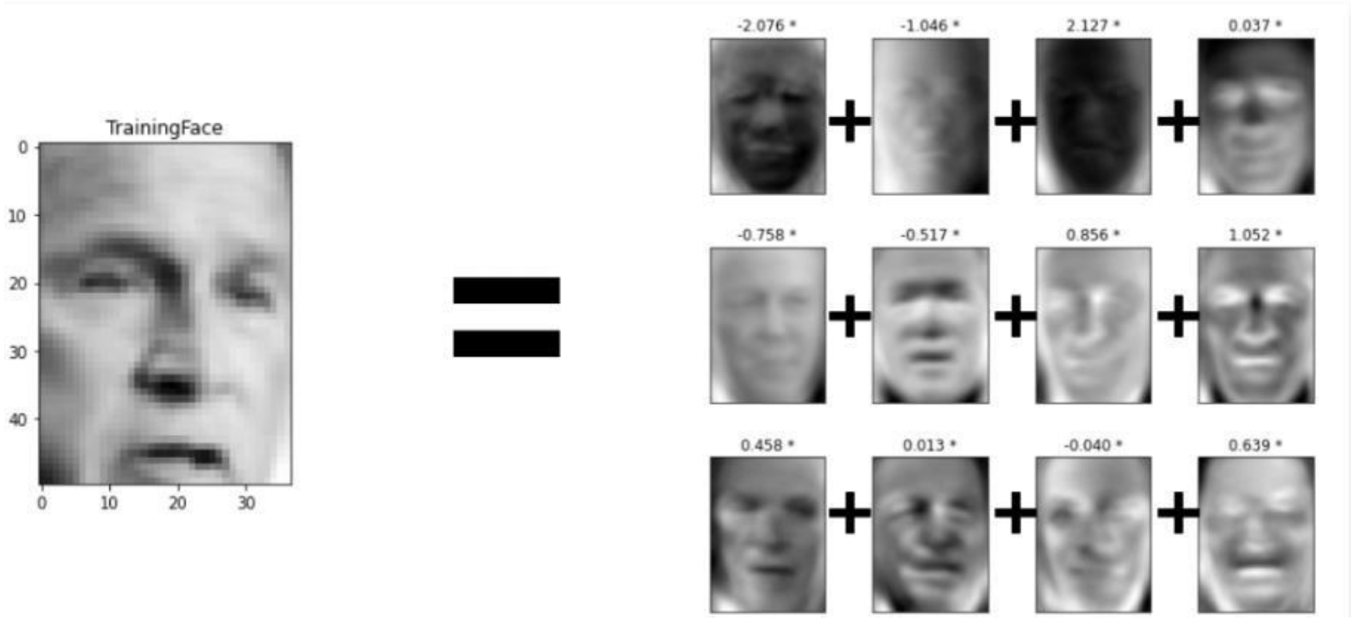
$$\mathrm{x}_i - \psi = \sum_{j=1}^{K} w_j u_j$$

These u_j are called **EigenFaces**.

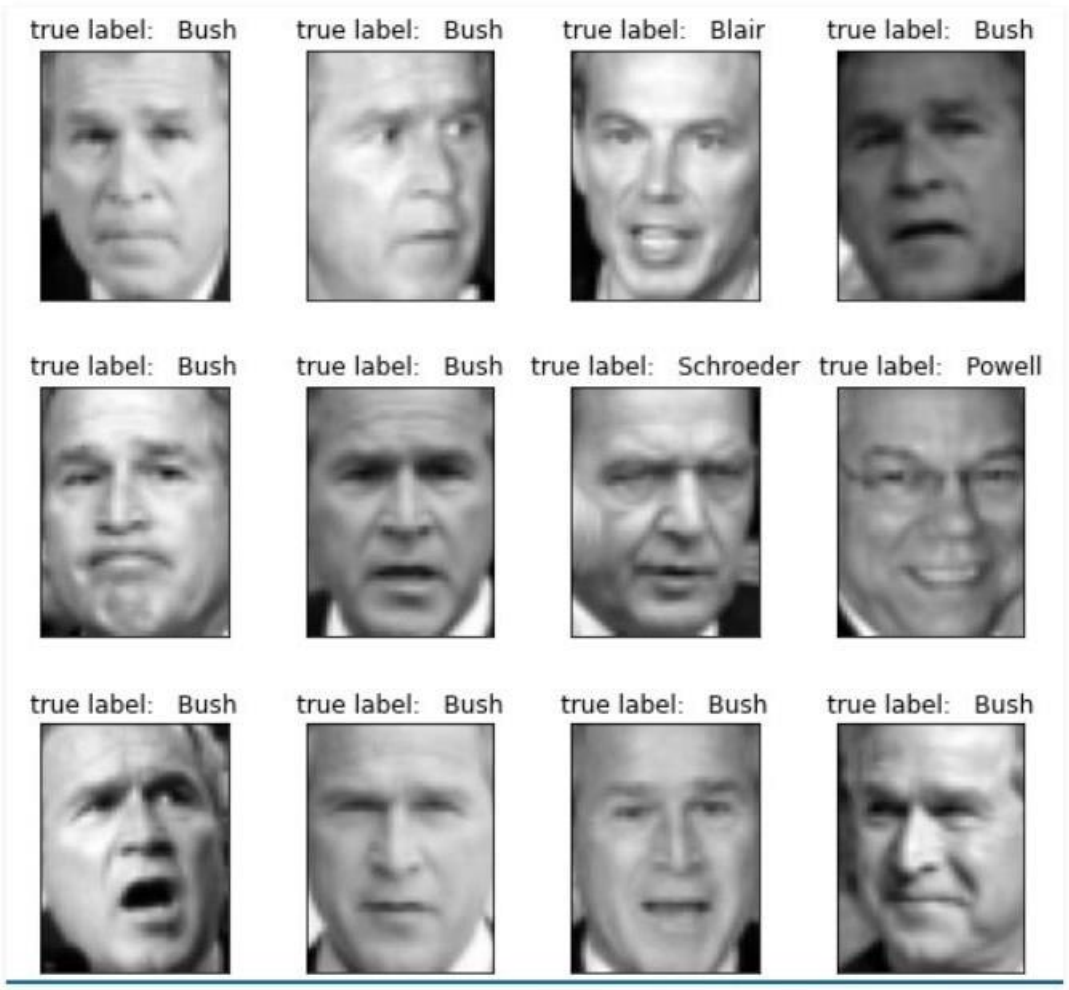


- In this step, we take the coefficient of eigenfaces and represent the training faces in the form of a vector of those coefficients.

$$x_i = \begin{bmatrix} w_1^i \\ w_2^i \\ w_3^i \\ . \\ . \\ w_k^i \end{bmatrix}$$



Testing the Algorithm:



- Given an unknown face *y*, we need to first pre-process the face to make it centred in the image and have the same dimensions as the training face

- Now, we subtract the face from the average face ψ.

$$\phi = y - \psi$$

- Now, we project the normalized vector into eigenspace to obtain the linear combination of eigenfaces.

$$\phi = \sum_{i=1}^{k} w_i u_i$$

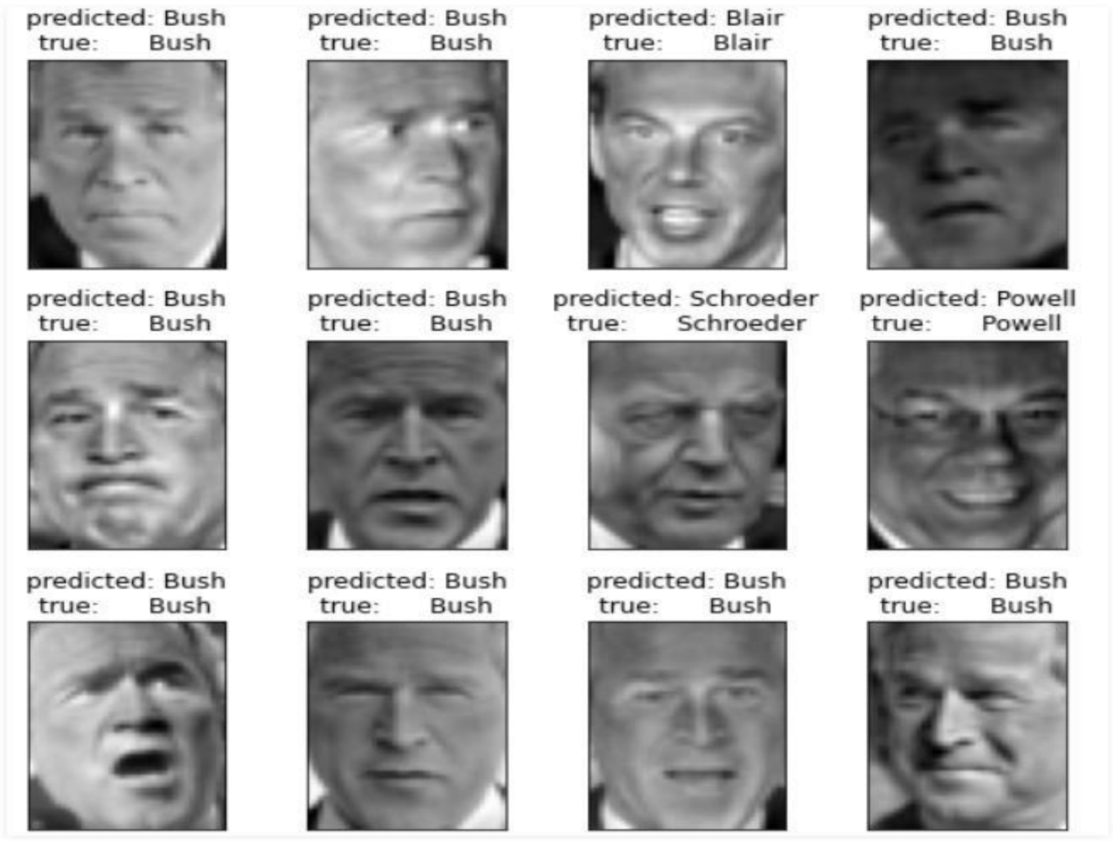
- From the above projection, we generate the vector of the coefficient such that:

$$\Omega = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ . \\ . \\ w_k \end{bmatrix}$$

- We take the vector generated in the above step and subtract it from the training image to get the minimum distance between the training vectors and testing vectors

$$e_r = min_l \|\Omega - \Omega_l\|$$

- If this e_r is below tolerance level T_r, then it is recognised with l face from training image else the face is not matched from any faces in training set.



These represent the “generic” faces of our dataset. Intuitively, these are vectors that represent directions in “face space” and are what our neural network uses to help with classification. Now that we’ve discussed the eigenfaces approach, you can build applications that use this face recognition algorithm.

The essence of eigenfaces is an unsupervised dimensionality reduction algorithm called **Principal Components Analysis (PCA)** that we use to reduce the dimensionality of images into something smaller. Now that we have a smaller representation of our faces, we apply a classifier that takes the reduced-dimension input and produces a class label. For our classifier, we used a single-layer neural network.

Result

Code: Importing libraries:

```python
# Import matplotlib library
import matplotlib.pyplot as plt

# Import scikit-learn library
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

import numpy as np
```

Code: Import the LFW_people dataset using sklearn's fetch_lfw_people function API:

```python
# this command will download the LFW_people's dataset to hard disk.
lfw_people = fetch_lfw_people(min_faces_per_person = 70, resize = 0.4)

# introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape

# Instead of providing 2D data, X has data already in the form  of a vector that
# is required in this approach.
X = lfw_people.data
n_features = X.shape[1]

# the label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

# Print Details about dataset
print("Number of Data Samples: % d" % n_samples)
print("Size of a data sample: % d" % n_features)
print("Number of Class Labels: % d" % n_classes)
```

Output:

```
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976012
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976009
Downloading LFW metadata: https://ndownloader.figshare.com/files/5976006
Downloading LFW data (~200MB): https://ndownloader.figshare.com/files/5976015
Number of Data Samples: 1288
Size of a data sample: 1850
Number of Class Labels: 7
```

Code: Data Exploration:

```
# Function to plot images in 3 * 4
def plot_gallery(images, titles, h, w, n_row = 3, n_col = 4):
    plt.figure(figsize =(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom = 0, left =.01, right =.99, top =.90, hspace =.35)
    for i in range(n_row * n_col):
        plt.subplot(n_row, n_col, i + 1)
        plt.imshow(images[i].reshape((h, w)), cmap = plt.cm.gray)
        plt.title(titles[i], size = 12)
        plt.xticks(())
        plt.yticks(())

# Generate true labels above the images
def true_title(Y, target_names, i):
    true_name = target_names[Y[i]].rsplit(' ', 1)[-1]
    return 'true label:   % s' % (true_name)

true_titles = [true_title(y, target_names, i)
                     for i in range(y.shape[0])]
plot_gallery(X, true_titles, h, w)
```

Code: Training and Test splitting:

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size = 0.25, random_state = 42)
print("size of training Data is % d and Testing Data is % d" %(
        y_train.shape[0], y_test.shape[0]))
```

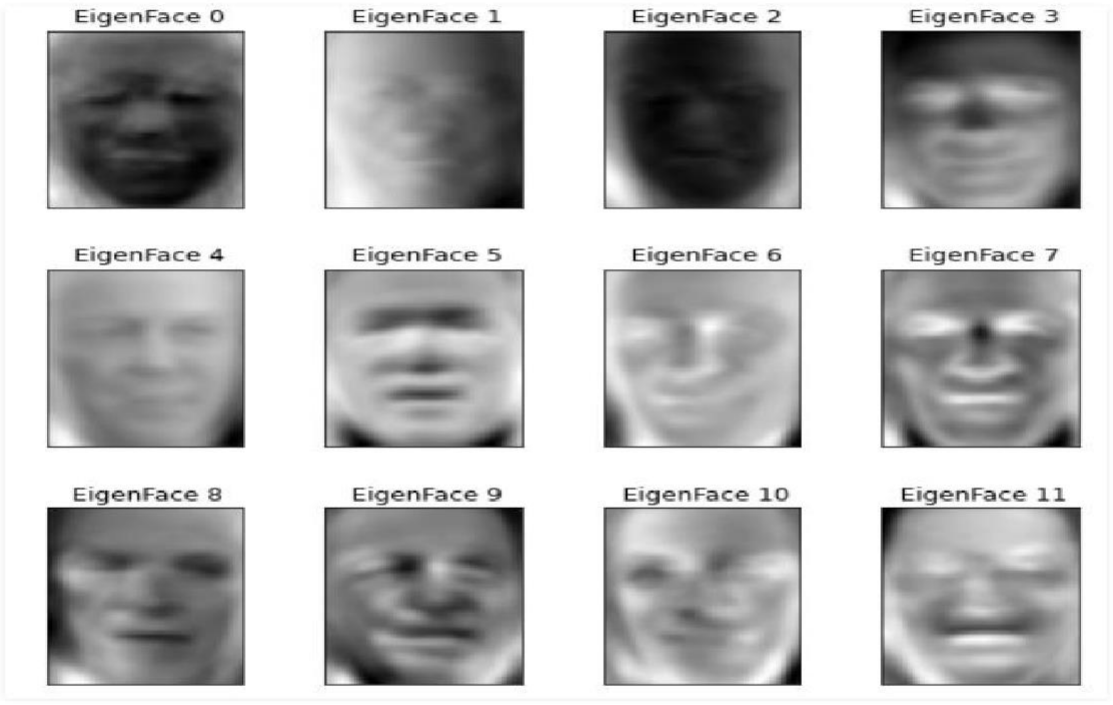
```
size of training Data is 966 and Testing Data is 322
```

Code: Implementing PCA:

```
n_components = 150

t0 = time()
pca = PCA(n_components = n_components, svd_solver ='randomized',
          whiten = True).fit(X_train)
print("done in % 0.3fs" % (time() - t0))

eigenfaces = pca.components_.reshape((n_components, h, w))

print("Projecting the input data on the eigenfaces orthonormal basis")
t0 = time()
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)
print("done in % 0.3fs" % (time() - t0))
```

```
Extracting the top 150 eigenfaces from 966 faces
done in 0.209s
Projecting the input data on the eigenfaces orthonormal basis
done in 0.023s
```

Eigenfaces generated by this PCA algorithm:



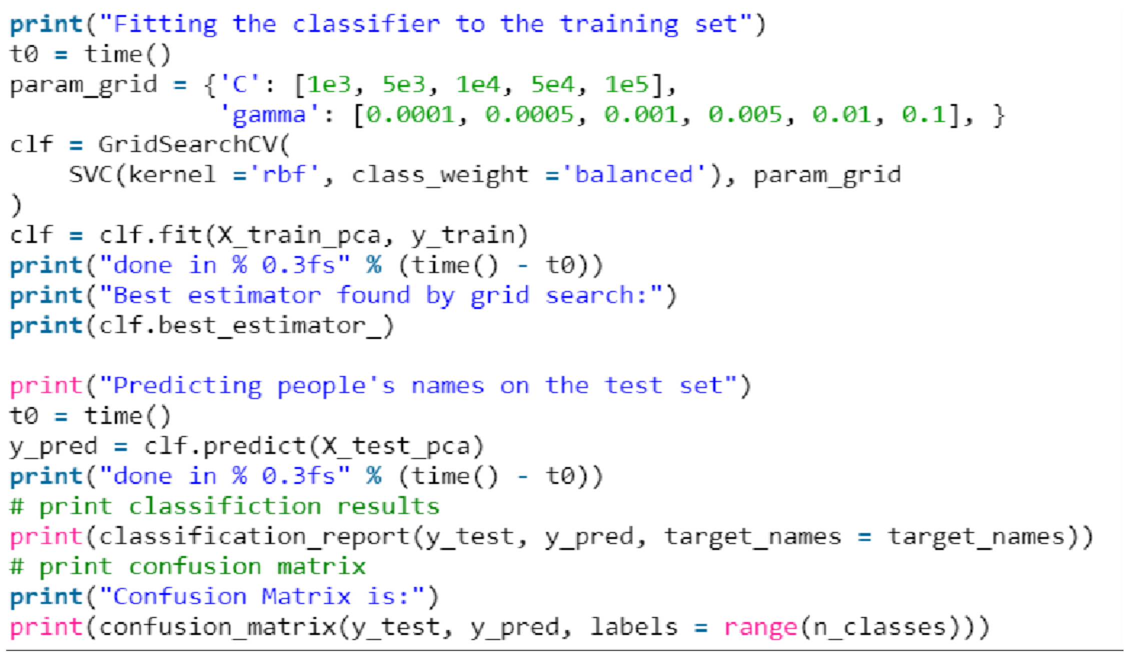
Code: Explore the coefficient generated by the above algorithm:

```
print("Sample Data point after applying PCA\n", X_train_pca[0])
print("-----------------------------------------------------------")
print("Dimesnsions of training set = % s and Test Set = % s"%(
        X_train.shape, X_test.shape))
```

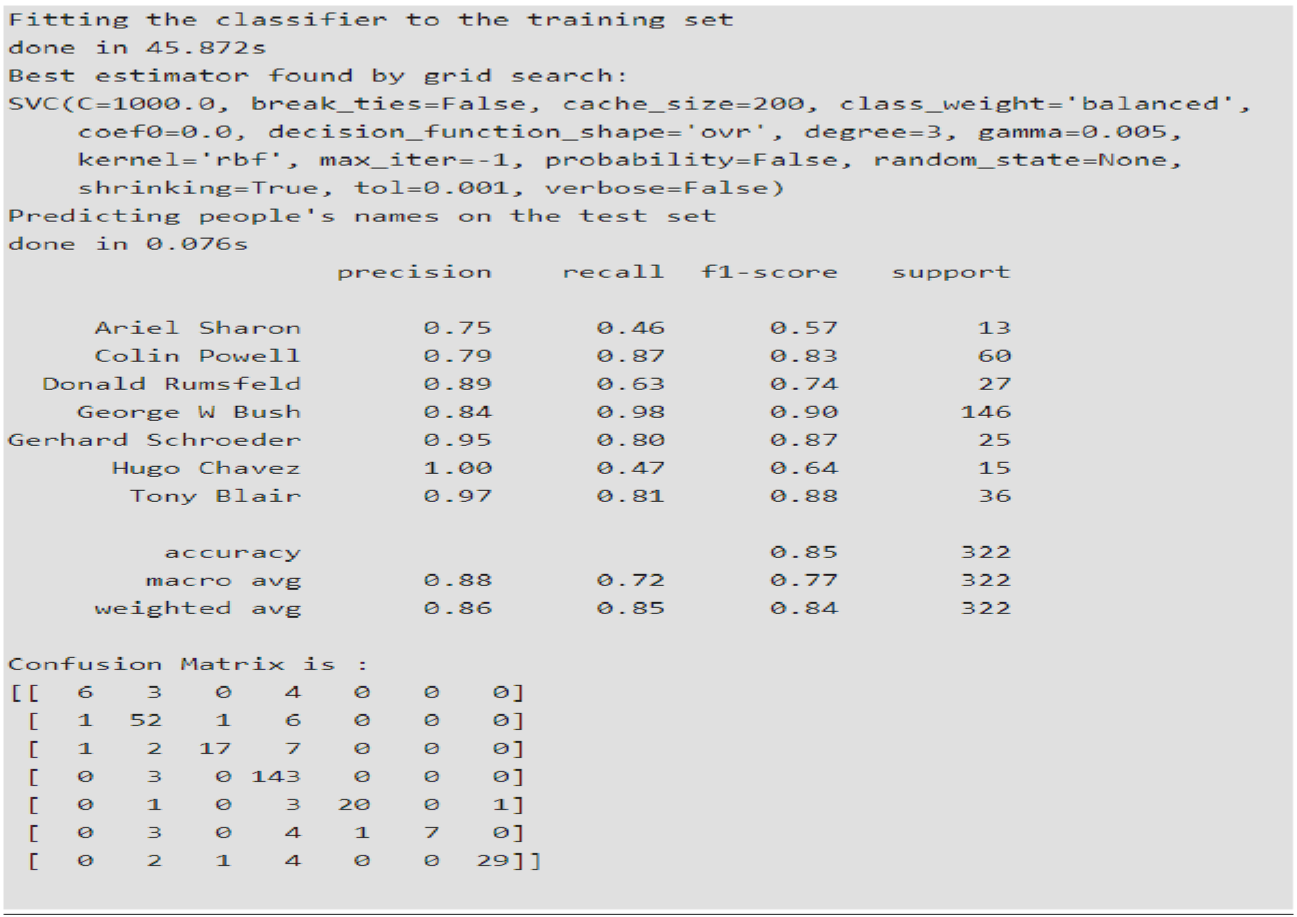
Output:

```
Sample Data point after applying PCA
[-2.0756025  -1.0457923   2.126936    0.03682641 -0.7575693  -0.51736575
  0.8555038   1.0519465   0.45772424  0.01348036 -0.03962574  0.63872665
  0.4816719   2.337867    1.7784412   0.13310494 -2.271292   -4.4569106
  2.0977738  -1.1379385   0.1884598  -0.33499134  1.1254574  -0.32403082
  0.14094219  1.0769527   0.7588098  -0.09976506  3.1199582   0.8837879
 -0.893391    1.1595601   1.430711    1.685587    1.3434631  -1.2590996
 -0.639135   -2.336333   -0.01364169 -1.463893   -0.46878636 -1.0548446
 -1.3329269   1.1364135   2.2223723  -1.801526   -0.3064784  -1.0281631
  4.7735424   3.4598463   1.9261417  -1.3513585  -0.2590924   2.010101
 -1.056406    0.36097565  1.1712595   0.75685936  0.90112156  0.59933555
 -0.46541685  2.0979452   1.3457304   1.9343662   5.068155   -0.70603204
  0.6064072  -0.89698195 -0.21625179 -2.1058862  -1.6839983  -0.19965973
 -1.7508434  -3.0504303   2.051207    0.39461815  0.12691127  1.2121526
 -0.79466134 -1.3895757  -2.0269105  -2.791953    1.4810398   0.1946961
  0.26118103 -0.1208623   1.1642501   0.80152154  1.2733462   0.09606536
 -0.98096275  0.31221238  1.0365396   0.8510516   0.5742255  -0.49945745
 -1.3462409  -1.036648   -0.4910289   1.0547347   1.2205439  -1.3073852
 -1.1884091   1.8626214   0.6881952   1.8356183  -1.6419449   0.57973146
  1.3768481  -1.8154184   2.0562973  -0.14337398  1.3765801  -1.4830858
 -0.0109648   2.245713    1.6913172   0.73172116  1.0212364  -0.09626482
  0.38742945 -1.8325268   0.8476424  -0.33258602 -0.96296996  0.57641584
 -1.1661777  -0.4716097   0.5479076   0.16398667  0.2818301  -0.83848953
 -1.1516216  -1.0798892  -0.58455086 -0.40767965 -0.67279476 -0.9364346
  0.62396616  0.9837545   0.1692572   0.90677387 -0.12059807  0.6222619
 -0.32074842 -1.5255395   1.3164424   0.42598936  1.2535237   0.11011053]
-----------------------------------------------------
Dimesnsions of training set (966, 1850) and Test Set (322, 1850)
```

Code: Applying Grid Search Algorithm:

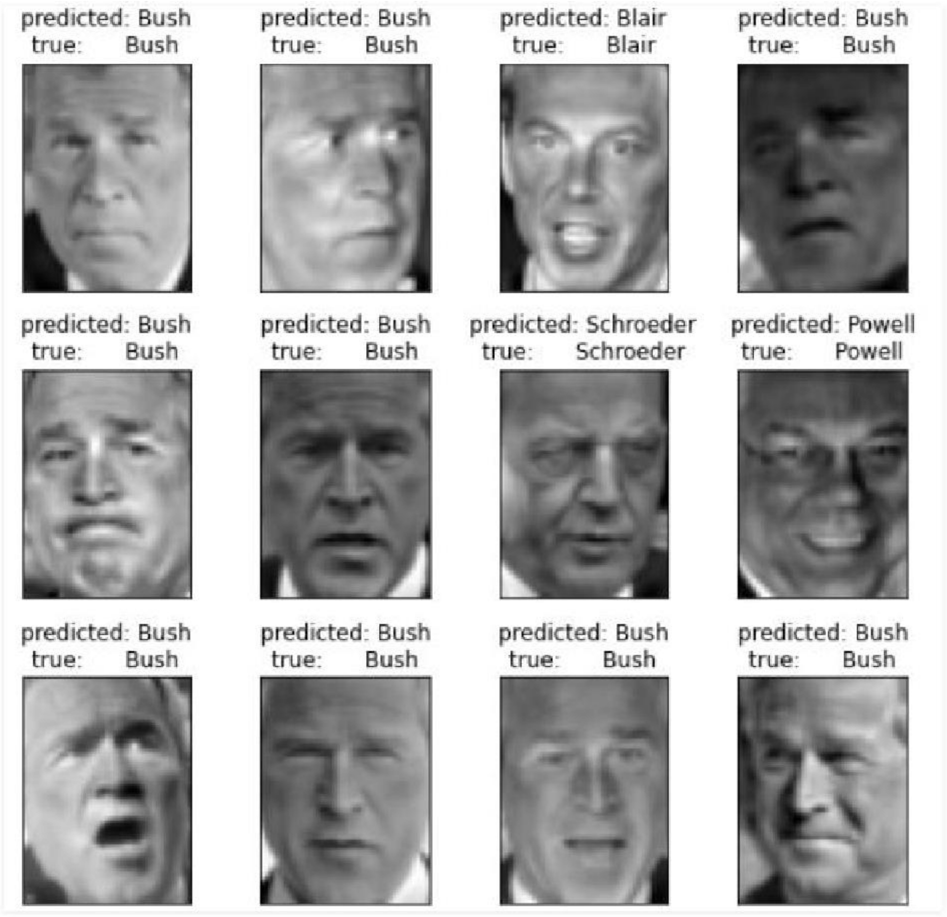


```python
print("Fitting the classifier to the training set")
t0 = time()
param_grid = {'C': [1e3, 5e3, 1e4, 5e4, 1e5],
              'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1], }
clf = GridSearchCV(
    SVC(kernel ='rbf', class_weight ='balanced'), param_grid
)
clf = clf.fit(X_train_pca, y_train)
print("done in % 0.3fs" % (time() - t0))
print("Best estimator found by grid search:")
print(clf.best_estimator_)

print("Predicting people's names on the test set")
t0 = time()
y_pred = clf.predict(X_test_pca)
print("done in % 0.3fs" % (time() - t0))
# print classifiction results
print(classification_report(y_test, y_pred, target_names = target_names))
# print confusion matrix
print("Confusion Matrix is:")
print(confusion_matrix(y_test, y_pred, labels = range(n_classes)))
```



```
Fitting the classifier to the training set
done in 45.872s
Best estimator found by grid search:
SVC(C=1000.0, break_ties=False, cache_size=200, class_weight='balanced',
    coef0=0.0, decision_function_shape='ovr', degree=3, gamma=0.005,
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
Predicting people's names on the test set
done in 0.076s
                   precision    recall  f1-score   support

     Ariel Sharon       0.75      0.46      0.57        13
     Colin Powell       0.79      0.87      0.83        60
  Donald Rumsfeld       0.89      0.63      0.74        27
    George W Bush       0.84      0.98      0.90       146
Gerhard Schroeder       0.95      0.80      0.87        25
      Hugo Chavez       1.00      0.47      0.64        15
       Tony Blair       0.97      0.81      0.88        36

         accuracy                           0.85       322
        macro avg       0.88      0.72      0.77       322
     weighted avg       0.86      0.85      0.84       322

Confusion Matrix is :
[[  6   3   0   4   0   0   0]
 [  1  52   1   6   0   0   0]
 [  1   2  17   7   0   0   0]
 [  0   3   0 143   0   0   0]
 [  0   1   0   3  20   0   1]
 [  0   3   0   4   1   7   0]
 [  0   2   1   4   0   0  29]]
```

So, our accuracy is *0.85* and Results of our prediction are:



Conclusion

Thus, the face recognition system based on PCA has been implemented. It accurately identifies input face images of an individual which differ from the set of images of that person already stored in the database thus serving as an effective method of recognizing new face images. The base code for training face images using Back Propagation Neural Network has also been completed. Hence when a new image is fed into the system for recognition the main features are extracted and computed to find the distance between the input image and the stored images.

Thus, some variations in the new face image to be recognized can be tolerated. When the new image of a person differs from the images of that person stored in the database, the system will be able to recognize the new face and identify who the person is. Recognition accuracy and better discriminatory power Computational cost because smaller images (main features) require less processing to train the PCA. Because of the use of dominant features and hence can be used as an effective means of authentication.

Bibliography

1. https://ieeexplore.ieee.org/document/8280652
2. https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/
3. https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect14.pdf
4. https://setosa.io/ev/principal-component-analysis/
5. http://pace.ac.in/documents/ece/FACE%20RECOGNITION%20SYSTEM%20WITH%20FACE%20DETECTION.pdf
6. https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/