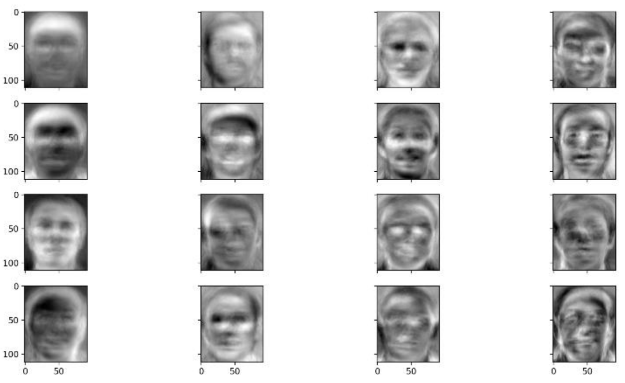
Applied Linear Algebra
COURSE CODE: MAT161
CSE'27
School of Engineering

PROJECT REPORT

EIGENFACES FOR RECOGNITION

Inspired by Matthew Turk and Alex Pentland's research paper



(A collection of processed images from the later mentioned database)

NAME: Arnab Mandal
ROLL NO: 2310110428
SNU ID: am483

CONTENTS

INTRODUCTION:

The aim of this project is to utilize applied linear algebra, via the principal component analysis (PCA) technique, as a method of facial recognition. This technique was proposed initially by Sirovich and Kirby in 1987, and since then it has been popularized as a reliable and consistent method of facial recognition. It represents facial features as a linear combination of a small set of basis images called eigenfaces, and utilizes these in a small procedure to make identification an efficient process.

We initialize this process by

1. Utilizing a training set, which is an initial collection of face photos, preferably multiple of the same person, from difference angles and orientations.

2. We then proceed to calculate the eigenfaces, only keeping the M images that correspond to the greatest eigenvalues. The face space is defined by these M pictures, and are revised, and updated as new faces are encountered by constantly appending to the space.

3. We then project each known person's face images onto the previously created face space, to determine the distribution that corresponds to each one in M-Dimensional weight space.

This is then utilized in the following steps, to identify and distinguish fresh face images:

1. Scanning and projecting the input image, aka "query image" onto each of the M eigenfaces to determine a set of weights based on the input image and the eigenfaces.

2. Image is classified as a face image, by examining its proximity to the face space, previously created and modified.

3. If it is a face, we categorize the weight pattern, and search in the previously created distribution to determine whether the query is a known or an unknown person.

EIGENFACE RECOGNITION:

As previously mentioned, principal component analysis, PCA is applied to identify the most important facial features, that distinguish one face from the other.
We begin by creating a matrix, X, where each row represents a different face image, with columns representing the pixel intensities of that image.
We then compute the mean vector, $\boldsymbol{m\vec{u}}$, of all the face images, and subtract it from each image vector to find a mean of sorts, so as to centre the data.
The centred matrix is denoted by, Y, and as explained initially, each row represents a "cantered face vector".
We proceed to compute the covariance matrix, C, of the centred data, Y, which is given by,

$$C = \frac{YT^{Y}}{n-1}$$

where n=number of face images.
The covariance matrix has information on the intensity of each pixel varies with respect to the others across the set of face images.
We then compute the eigenvectors and eigenvalues of C as,

$$V_i = \lambda_i v_i$$

We then sort the eigenvectors in descending order based on their corresponding eigenvalues and select the first k eigenvectors to form the eigenface basis set,

$$V = [v_1, v_2, \ldots., v_k]$$

The corresponding eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$ tell us how much of the variance in the data is explained by each of the k principal components. To represent a new face image, y, in terms of the eigenface basis set V_k. This projection is given by the formula,

$$a = (V)^T (y - m\vec{u})$$

where a is the vector of coefficients, that represent the image in terms of the eigenfaces. Each coefficient a_i tells us how much the image contributes to the i-th eigenface.

Finally, we can perform facial recognition by comparing the coefficient vector of a new face image, with those already present in the set we create with known faces. The image with the closest match, in terms of the Euclidean distance, between their respective coefficient vectors, is considered the match.
Summarizing this we get,
Considering a set **S** of *m* images:
1. Create a matrix **M** where each row is an individual image (i.e. concatenate the rows of each image into a single row).
2. Compute the average face, $\mathbf{M_{avg}}$, and subtract that face from each row in M, i.e.

$$\mathbf{M = M - M_{avg}}$$

3. Calculate the eigenvectors of the covariance matrix for each image, where we define the matrix as,

$$C = \frac{\sum \mathrm{Ai}*\mathrm{Ai}'}{m}$$

$$= AA^T$$

where $A' = A^T$ is the transpose of matrix A.

However, as the matrix C is a N^2*N^2, it is not feasible to calculate the requisite eigenvectors and eigenvalues manually, and this needs to be done computationally. Instead, we can resolve the eigenvectors of a M*M matrix, N, and then solve for dimensional eigenvectors,

$$P= AA^T$$

and then find N's M eigenvectors. This is used as a training set, and the face pictures are combined linearly using these vectors to create the eigenfaces,

$$u_p=\sum vA$$

Here, as M<<N, the training set of face images can now be handled with ease, as the number of necessary iterations has been drastically reduced, and now the eigenvectors can be ordered on the basis of how well they capture the variation among the photos thanks to the associated eigenvalues.

CLASSIFYING A FACIAL IMAGE USING EIGENFACES:

1. Preprocessing: Let us consider the test face image as y. We normalise the size and orientation of the image, grayscale it, and define it as y_{pre}.

2. Training: Let's consider "x_1, x_2, …., x_n" so as to represent the collection of training face photos. Keep in mind these have already been normalised, after which they are represented as vectors in a matrix X, where each row denotes a pre-processed training face vector. We proceed with calculations

$$\boldsymbol{m\vec{u}} = \frac{\sum \boldsymbol{x\_i}}{\boldsymbol{n}}$$

where the vector is the pre-processed training face vectors' mean, and I can range from 1 to n.

We then obtain the training set of face vectors Y by

$$Y= [y_1 - m\vec{u},\ y_2 - m\vec{u}, \ldots., y_n - m\vec{u}]$$

We proceed to apply PCA, to determine the eigenface basis set V_k as mentioned previously.

3. Representation: To acquire the relevant coefficient vector a, we project the pre-processed test face image, onto the eigenface basis set by,

$$a = (Y - m\vec{u})\ V_kT$$

4. Classification: We compute the Euclidean distance between the test face image's coefficient vector a and the coefficient vectors of the training set of face pictures in order to classify it. Let's write "a_1, a_2, …., a_n" for the coefficient vectors of the training set of face photos. The following formula provides the Euclidean distance between each training face image and the test face image: For i = 1 to n, $d_i = ||a - a_i||$ The best match in the training set is represented by the index i with the shortest distance d_i. The class label of the test image is then changed to match the class label of the matched training image.

In conclusion, we first preprocess the image and compute the eigenface basis set from a series of training face images in order to categorise a face image using Eigenfaces. Using the eigenface basis set to represent the pre-processed test face image as a coefficient vector, we can then identify the test picture's class label by calculating the Euclidean distance between its coefficient vector and the coefficient vectors in the training set of face images.

LOCATING AND DETECTING FACES:

The analysis in the parts before assumes that the face picture is centred and the same size as the eigenfaces and training images. So, in order to perform the recognition, we need means to identify a face in a scene, namely via motion detection, and face space image editing. When projected into the face space, images of faces don't alter significantly, but projections of nonface images look very different.
This fundamental concept is utilised to identify faces in a scene: calculate the distance ϵ between the local sub-image and face space at each position in the image.

$$\begin{aligned}\epsilon^2 &= \|\mathbf{\Phi} - \mathbf{\Phi}_f\|^2 \\ &= (\mathbf{\Phi} - \mathbf{\Phi}_f)^T(\mathbf{\Phi} - \mathbf{\Phi}_f) \\ &= \mathbf{\Phi}^T\mathbf{\Phi} - \mathbf{\Phi}^T\mathbf{\Phi}_f + \mathbf{\Phi}_f^T(\mathbf{\Phi} - \mathbf{\Phi}_f) \\ &= \mathbf{\Phi}^T\mathbf{\Phi} - \mathbf{\Phi}_f^T\mathbf{\Phi}_f\end{aligned}$$

since ϕf is perpendicular to (ϕ-ϕf). Because ϕf is a linear combination of the eigenfaces (ϕf = sum(ωiui)) and the eigenfaces are orthonormal vectors,

$$\mathbf{\Phi}_f^T\mathbf{\Phi}_f = \sum_{i=1}^{L} \omega_i^2$$

And,

$$\epsilon^2(x, y) = \mathbf{\Phi}^T(x, y)\,\mathbf{\Phi}(x, y) - \sum_{i=1}^{L} \omega_i^2(x, y) \qquad (1)$$

where ϕ(x,y) is a vector function of image location and ϵ(x,y) and ωi(x,y) are scalar functions of image location. A correlation with the L eigenfaces is used in practise to derive the second term of Eq. (1):

$$\begin{aligned}\textstyle\sum_{i=1}^{L} \omega_i^2(x, y) &= \textstyle\sum_{i=1}^{L} \mathbf{\Phi}^T(x, y)\mathbf{u}_i \\ &= \textstyle\sum_{i=1}^{L} [\mathbf{\Gamma}(x, y) - \mathbf{\Psi}]^T\mathbf{u}_i \\ &= \textstyle\sum_{i=1}^{L} [\mathbf{\Gamma}^T(x, y)\mathbf{u}_i - \mathbf{\Psi}^T\mathbf{u}_i] \\ &= \textstyle\sum_{i=1}^{L} [I(x, y) \otimes \mathbf{u}_i - \mathbf{\Psi}^T\mathbf{u}_i]\end{aligned}$$

So that

$$\boldsymbol{\epsilon}^2(x, y) = \boldsymbol{\Gamma}^T(x, y)\boldsymbol{\Gamma}(x, y) - 2\boldsymbol{\Gamma}(x, y) \otimes \boldsymbol{\Psi} + \boldsymbol{\Psi}^T\boldsymbol{\Psi} + \sum_{i=1}^{L} [\boldsymbol{\Gamma}(x, y) \otimes \mathbf{u}_i - \boldsymbol{\Psi} \otimes \mathbf{u}_i]$$

The variables $\Psi^T\Psi$ and $\Psi \otimes u_i$ may be computed beforehand because the eigenfaces ui and the average face Ψ are fixed.
Thus, only L + 1 correlations across the input image and the computation of the first term Γ T (x, y)Γ(x, y) are needed to create the face map. This is calculated by squaring the input image I(x, y) and adding the squared values of the local sub images at each image point.

RECOGNISING NEW FACES:

We can learn, detect new information, and recognise new faces via the existing information stored in the face space. For example, images that are sufficiently close to face space but do not match the recognised faces directly, are classified as unknown, and are hypothesised and stored as a group of unknown pattern vectors in the existing space.
The distance from each image to the mean vector, must be less than a situation specific limit, or threshold, so as to declare whether images are even comparable. If they are, they are declared to pass the similarity test, and thus, are added to the database of recognised face, and might even be added to the training set, to provide a recalculation of existing values, to improve the accuracy rate.

ISSUES:

Eigenvalues while being successful via this method of PCA and follow up, have some bugs, that effect the accuracy of the recognition, because of which better methods are being investigated into, to resolve issues such as

1. Eliminating the background:

Eigenfaces, are sensitive to pixels even in the background, which makes it necessary to remove background of the image, in order to increase accuracy.

2. Scale and Orientation variation:

In order to ensure highest accuracy, the images must be at the same scale, and of similar orientations, which is not possible in real life situations, and thus further preprocessing such as multi-scale representation, or localised PCA is done.

3. Face space singularity:

Eigenfaces are based on a single distribution of face images in the face space, maybe even a more of a one-dimensional approach, unlike the world we’re examining which exists in 3-D, and thus cannot capture the full range of variation, which may necessitate multiple face spaces or distributions of subspaces at the same time.

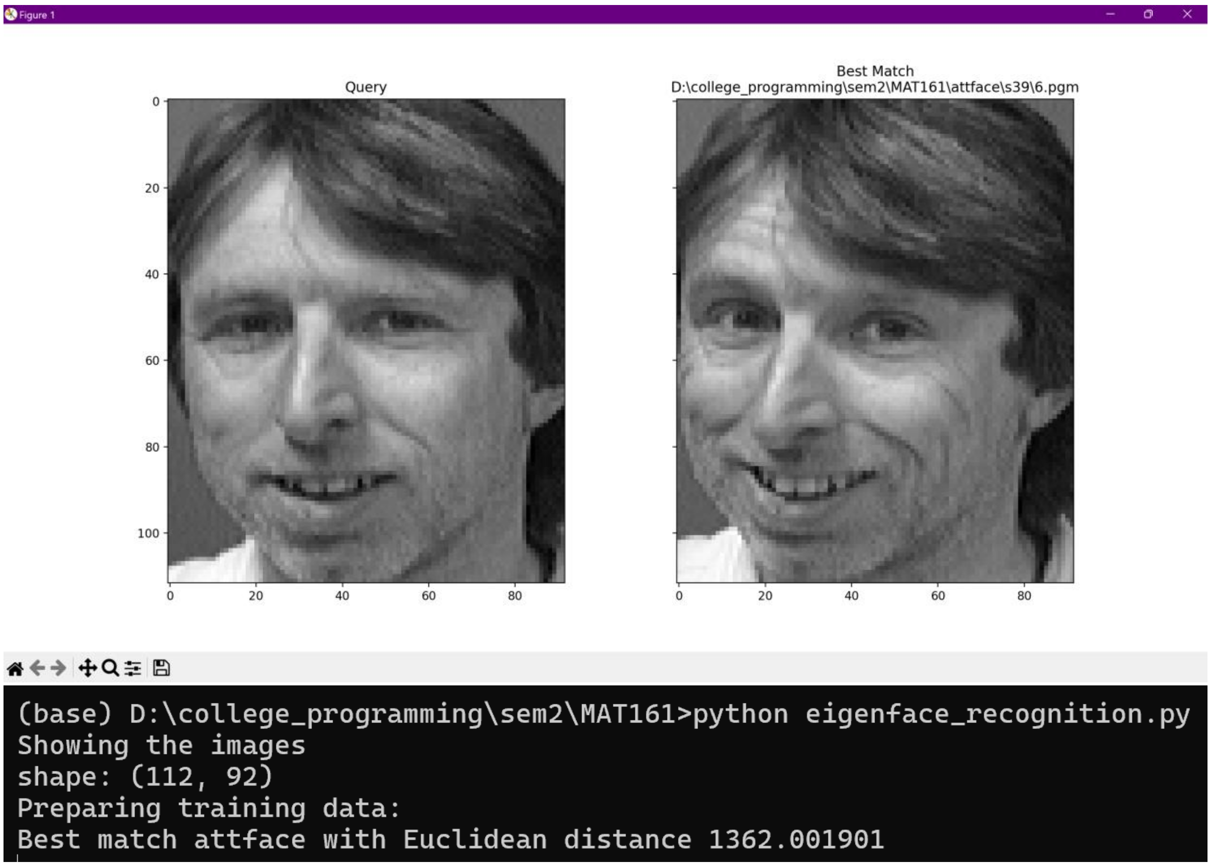
4. Capture point:

Eigenfaces do not compensate for changes in viewing or capturing angle, and to handle such multiple viewpoints, techniques such as 3-D face modelling and multi-view representation is to be implemented.

IMPLEMENTATION THROUGH PROGRAMMING

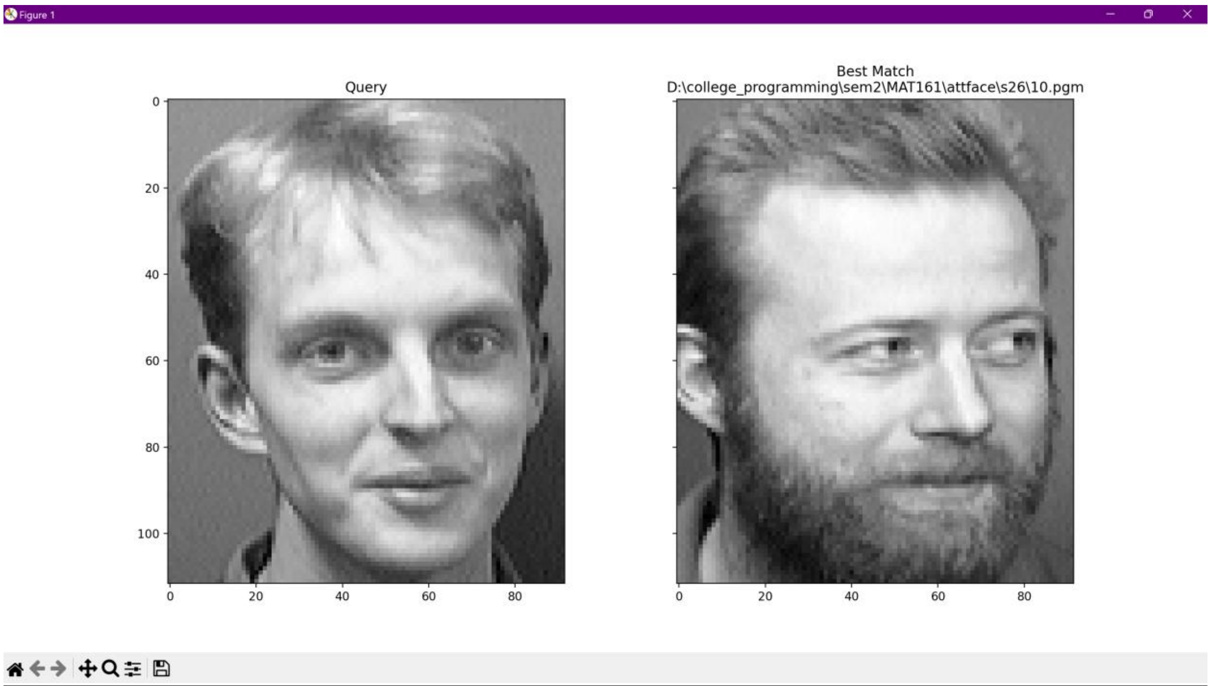
I have chosen to implement the basis via a programming language of my choice, aka python. A face space, and relevant information is created via the help of a ORL Database of faces, a zip file collection of 10 pictures of 40 people, creating a total of 400 pictures.
The output and relevant code is being shown, and the relevant steps to run the code on a local system will be attached as following, with all necessary files in a public github repository.
We are utilising 39 people from the given database as our training set, with one image excluded from the 39th person, and thus testing on 389/400 images.
We will be testing the created algorithm on 2 images.

The first image to be tested, will be of that of a person included in the training set, but from a unique position, or in a unique orientation. This follows, that while a perfect match will not be found, the person should be same, and there should be visible similarities, but however will have a non-zero Euclidean distance.
The output of the algorithm confirms the same as shown below.



```
(base) D:\college_programming\sem2\MAT161>python eigenface_recognition.py
Showing the images
shape: (112, 92)
Preparing training data:
Best match attface with Euclidean distance 1362.001901
```

The second image we work with, will be a completely unique person foreign to the training set. Thus, there'll be much greater differences, a larger Euclidean distance, but there should be still bare visible similarities.
The output of the code confirms the same.



```
Best match attface with Euclidean distance 2300.173588
```

The complete code, with all necessary files and instructions to run the program on a local machine can be found in this GitHub repository.

CONCLUSION:

Through this report, we analysed the use of eigenfaces for facial recognition, and exploring the linear algebra utilised in the process itself. We begun by explaining the created of eigenfaces by using principal component analysis, and utilising it for feature extraction and classification. We then delved into the mathematical basis for using eigenfaces for facial recognition, and then resolved issues with using eigenvalues for the same.
It must be noted that while however eigenfaces are indeed a powerful technique for facial recognition, they need a lot of pre and post processing steps, to increase accuracy.
Regardless in our current age, there are more accurate and robust recognition systems such as deep-learning based systems which while requiring more computational resources, offer much higher accuracy.

REFERENCES:

1. Wikipedia for Eigenfaces
2. Eigenfaces for recognition, by Matthew Turk and Alex Pentland