BDA492

Assignment-1

Face Recognition using Eigen-Faces

Introduction:

Face recognition is a method where we scan people's faces, recognize features and then match them with the person in records, something similar to police scanner. It's a type of classification where inputs are images and outputs are people's names. Eigen faces is a technique which makes this task easier and we'll discuss it in the further sections.

What exactly is Eigen Faces:

Eigenfaces is a method that is useful for face recognition and detection by determining the variance of faces in a collection of face images and use those variances to encode and decode a face in a machine learning way without the full information reducing computation and space complexity.

It's being used since the 1990's and is still relevant. Matthew Turk and Alex Pentland were one of the first people to use it for facial classification.

We've learnt that independent vectors create spaces in abstract ways. Imagine that all the spaces are formed by vectors and so are the subspaces inside these spaces. Some of these vectors contribute more to some of the things than the others and others may just be redundant.

At the heart of Eigen faces is an unsupervised dimensionality reduction technique called **principal component analysis (PCA)**.

We'll be discussing the theory and code of the paper predominantly in this report.

Basic concepts and their definitions:

1.) Eigenvectors and Eigenvalues:

Let's have a matrix **A** (square matrix). We know that almost all vectors change direction when they're multiplied by **A**. However, there are certain vectors **x** which have the exact same direction as **Ax**. These are called Eigenvectors. **Ax** and **x** just differ by their magnitudes.

Let's have another familiar equality, **Ax = λx**. **Lambda** is what makes these two equal and is called the Eigenvalue. This will tell us how our eigenvector changed after the multiplication.

2.) Principal Component Analysis (PCA):

The main goal of PCA is dimensionality reduction. It has many applications in visualization, feature extraction, data compression, etc.

The idea behind it is to linearly project original data onto a lower dimensional subspace offering the principal components (eigenvectors) maximum variance of the projected data and/or minimum distortion error from the projection. Eventually, both lead to the same result which is the best reconstruction formula. As a side note, this subspace is called principal subspace.

Research Paper:

Theory

Eigen Faces, in simpler terms, is nothing but a set of mathematical equations to narrow down three-dimensional pictures into a few vectors each of which represent different features or variation.

The set of equations given below describe how eigenfaces are formed given a set of actual MxM images.

We map the MxM matrix into a lower dimension space as it's hard to perform recognition on higher dimension space with each and every single detail.

Firstly, we flatten the MxM matrix into M^2 x 1 matrix.

We then calculate the average face using a simple equation: mean = 1/M(ΣT) where T is the set of all image vectors.

Calculating, Φ = T – mean, gives us the vector by which each face differs from the average vector. And we stack all these together to form MxMxI matrix where I is the number of images. A = MxMxI matrix.

1.)

$$\lambda_k = \frac{1}{M}\sum_{n=1}^{M}(\mathbf{u}_k^T\Phi_n)^2 \qquad (1)$$

We know that lambda is usually used to represent the scaled factor of the vector. We're just finding out the eigen values using the above equation and then computing U_k such that it is maximum which gives us the best distribution of the images.

2.)

$$\mathbf{u}_l^T\mathbf{u}_k = \delta_{lk} = \begin{cases} 1, & \text{if } l = k \\ 0, & \text{otherwise} \end{cases} \qquad (2)$$

This equation just gives information on how the best features from the images are taken into consideration. U_k being the best eigenvectors and Ul being all the eigen vectors of the given image and only when they are equal it's added to our image space.

3.)

$$C = \frac{1}{M}\sum_{n=1}^{M}\Phi_n\Phi_n^T \qquad (3)$$

$$= AA^T$$

We compute the covariance matrix C of the MxMxI matrix to find out how the input values are varying from the mean with respect to each other.

4.)

$$A^T A\mathbf{v}_i = \mu_i \mathbf{v}_i \qquad (4)$$

We must actually compute the eigenvalues and eigenvectors of the matrix AAT but assuming it's very huge since each image is typically of size 256x256 it's not practical at all.

Hence, we proceed to calculate for ATA and here vi is the eigen vector and μi being the eigenvalue. But how do we calculate for the actual AAT matrix? Simply by multiplying a few terms here and there just like we do in every other math equation.

5.)

$$AA^T A\mathbf{v}_i = \mu_i A\mathbf{v}_i \qquad (5)$$

We can simply reduce this equation as follows. Thus,

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

$$\Rightarrow (A v_i^* = \mu_i^* A v_i^*$$

$$\Rightarrow (u_i^* = \mu_i^* u_i^*$$

where $u_i^* = A v_i^*$

Thus, AAT and ATA have the same eigenvalues and their eigenvectors and their eigenvectors are related as follows: ui = Avi.

AAT can have up to MxM eigenvalues and eigenvectors whereas ATA can have only up to I eigenvectors and eigenvalues. Hence, I eigenvalues of ATA correspond to the I largest eigenvalues of AAT. We then go back to how equation 1 and equation 2 play a role in keeping only the best K eigenvectors of the calculated I eigenvectors.

6.)

$$\mathbf{u}_l = \sum_{k=1}^{M} \mathbf{v}_{lk} \mathbf{\Phi}_k, \qquad l = 1, \ldots, M \qquad (6)$$

We compute eigen face vector by simply multiplying the respective eigen vectors to their corresponding image as shown in the above equation. V_{lk} becomes zero when the best feature is not taken into consideration. Therefore, only best features take place in the eigenface vector ul.

After all this the number of points in the face plane are greatly reduced without compromising the quality.

Implementation:

We start by importing a few basic libraries and not any advanced ones as we're trying to get the results manually.

There are three important functions in the code which are as follows:

1.) **plot_portraits**: This function helps us visualize all the images that we need to see how our eigenfaces are coming along.

2.) **pca (X, n_pc)**: X is our vector of images and n_pc is the number of principal components.

Principal components are defined on an orthonormal basis that can extract the maximum variance in the original data. In our case, its shape is (1000, 4096) since we needed to transform the images into vectors for PCA. Then, we find the mean and subtract it from our data to center it around the origin. After that, we need to perform Singular Value Decomposition on the centered data to find those principal components called eigenfaces.

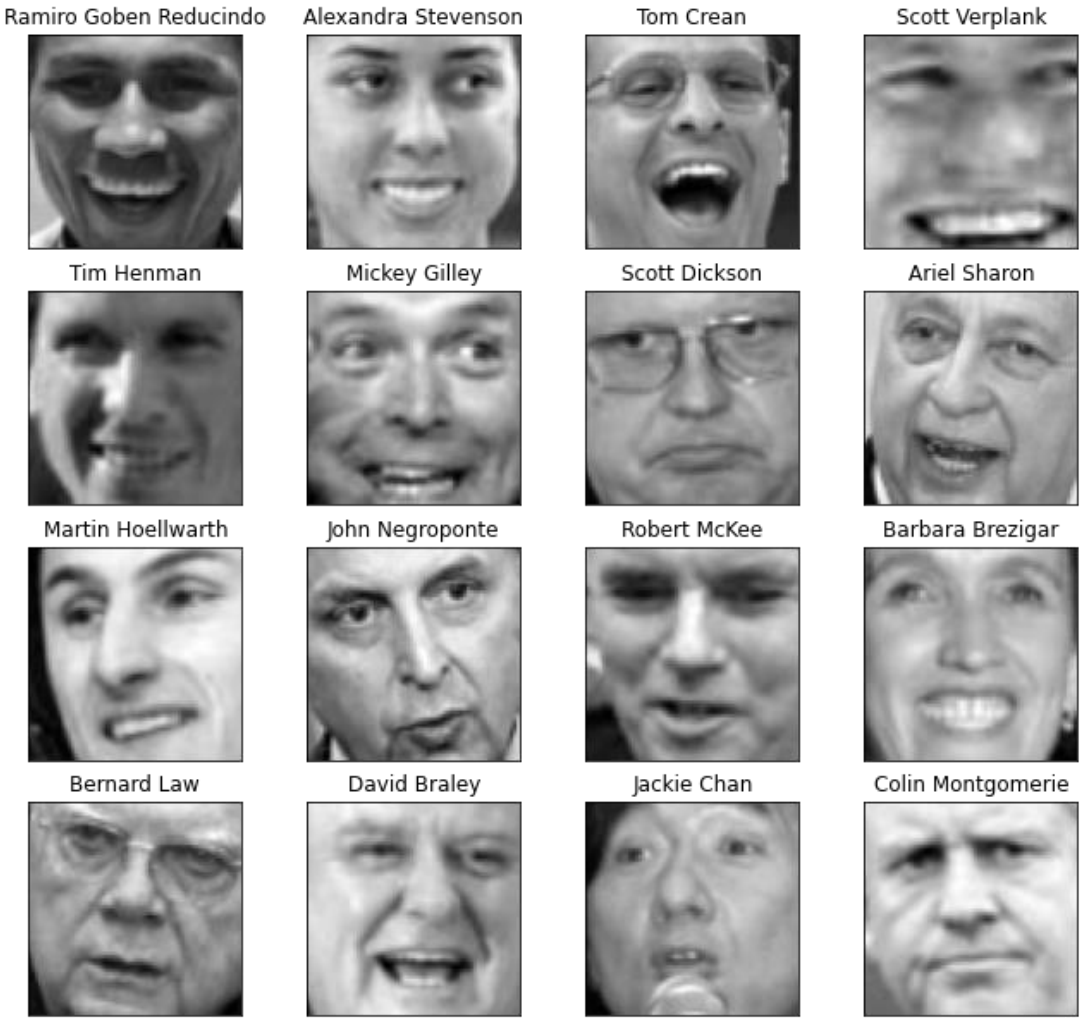
$$X = USV^T$$

3.) **reconstruction**: For reconstruction, we'll use the generated eigenfaces from pca function. Each face is a weighted combination of those eigenfaces as you might say. To find weights, we should dot multiply our centered data and eigenfaces. Then, we should weight eigenfaces which gives us the centered face and add it up to the mean face again. This gives our reconstructed face.

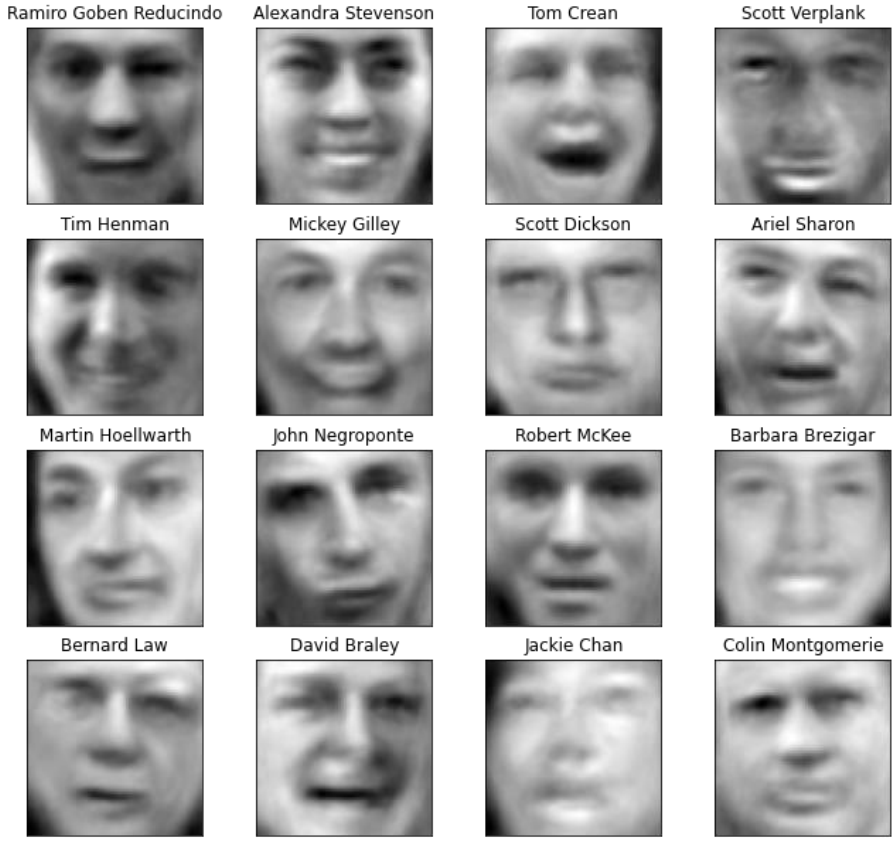
The results of reconstruction vary depending on the number of principal components we define. As the number of principal components grow, the faces start to look like the original ones.

Result Comparison:

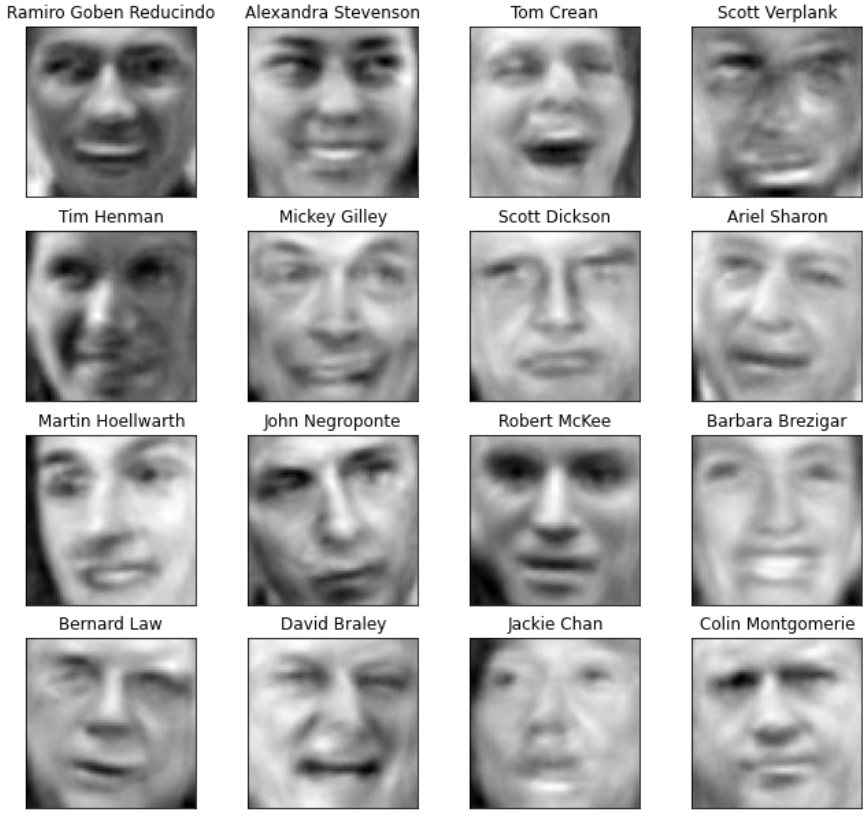
<u>The original 16 of the 1000 images</u>:



When principal components =50:



When principal components =100:



We can clearly see how the images come close to the original ones as we increase the number of principal components.