ES 331 - Probability and Random Processes
Assignment 1 - Report

Sai Praneeth Maddi
Roll No: 16110145

Task: To develop a face recognition system using eigenfaces in python.

Dataset: I have used an image dataset containing 38 different people with their images took in different illumination conditions, poses. Total number of images used for the algorithm is 2452 of 38 different persons. Image dataset was downloaded from this link.(However this image dataset has been submitted along with code file)

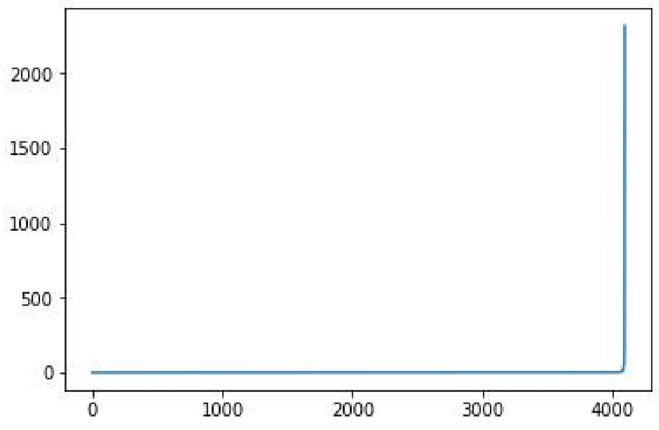
Sample image set

Algorithm:

1) Initially the images and their respective names(labels) have been imported using the libraries os,cv2. Images are in ".pgm" format(portable gray map). While reading the images, every image has been resized to 64X64 dimension to decrease the computation required.(This is also done to ensure that every random vector has same number of components)
2) Images have been flattened to one dimensional arrays. This has been done to represent each image as a random vector containing number components equal to number of pixels in the image. Finally, a matrix will be created containing rows as images (which are flattened) and columns as components of images. Shape of the matrix is 2452X4096 which says that there are 2452 distinct images which are 64X64 in shape(After flattening, 64X64 image becomes 1X4096 vector). This operation is done using a function in numpy, that is numpy.ndarray.flatten.
3) Now, for getting the matrix of random vectors as columns, transpose of the obtained data matrix has been taken. This matrix has columns as random vectors(flattened images) and rows as components of all images.
4) For constructing the covariance matrix of the random matrix, the random matrix has to be normalized. To do that, column wise mean of the random matrix has been subtracted(column wise) from the random matrix and then every element of random matrix has been divided by the standard deviation of that random matrix. These

operations are done with the help of functions in numpy library like numpy.std, numpy.mean etc.

5) After normalizing the random matrix, it is multiplied by its transpose and then the product is divided by number of samples(i.e, number of random vectors). Thus we obtain covariance matrix of the random matrix. These operations are done with the help of functions in numpy library like numpy.matmul.
6) Next, the covariance matrix has to be decomposed into its eigenvectors and diagonal matrix of eigenvalues. But for projection of the data matrix onto lower dimensions, we don't need the decomposition but we need its top eigenvectors. A functional tool in numpy library ,i.e, numpy.linalg.eigh, has been used to find the eigenvalues and eigenvectors of covariance matrix. That function returns the eigenvalues in ascending order and eigenvectors as columns corresponding to its eigenvalues. Dimension of eigen matrix is 4096X4096.
7) As we can choose number of components, I took top 256 components which contains almost all the data. This we can say by looking at the eigenvalue distribution,



Eigenvalue distribution

top 100 eigenvalues contain all the data and all other are very small compared to them. A submatrix of random matrix containing last 256 columns of all rows has been taken.

8) For projecting our data onto lower dimensions, our data matrix is multiplied by the eigen submatrix(which obtained in the above step). Dimension of data matrix is 2452X4096, dimension of eigen submatrix is 4096X256, so the projected data has dimension of 2452X256. After doing principal component analysis, we can say that without major loss in data we can reduce computational requirements by decreasing components.

9) Thus we got projected data of less dimensions, now this projected data is used to train a machine learning model and test that model. For that, we have to split our projected image data and label data into two groups, one for training and other for testing the trained model. This splitting is done using function in scikit learn library, that is sklearn.model_selection.train_test_split.

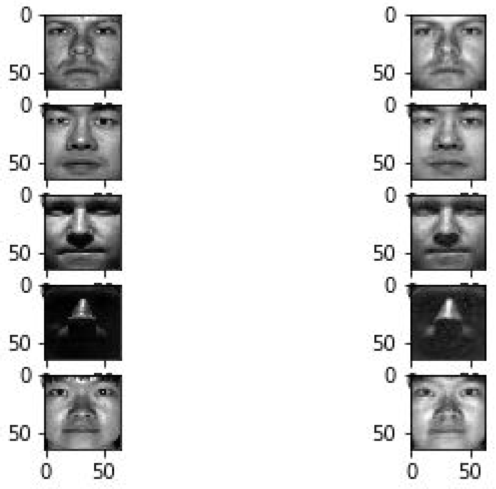
10) The machine learning algorithm I used is MLPClassifier(Multi Layer Perceptron Classifier). This model is imported using scikit learn library(sklearn.neural_network.MLPClassifier()). I have trained the model using the train group(which was obtained in the above step) and predicted the labels for projected test image

data. Then compared the predicted labels and actual test labels which has an accuracy of 74% to 75%.

11) Steps 10, 11 are repeated 100 times and average accuracy was taken. This is because to reduce uncertainty, as the accuracy changes every time we train and test.

Observations:

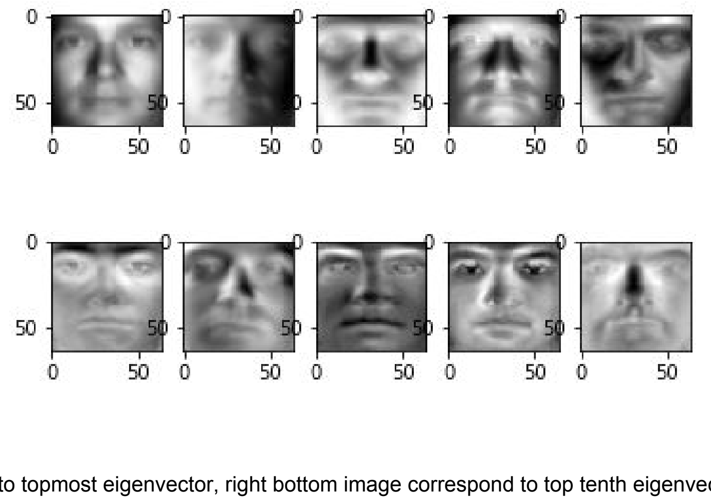
When images are projected onto lower dimensions and again reprojected into higher dimensions, they look like



Left images are original images, right images are reprojected images.

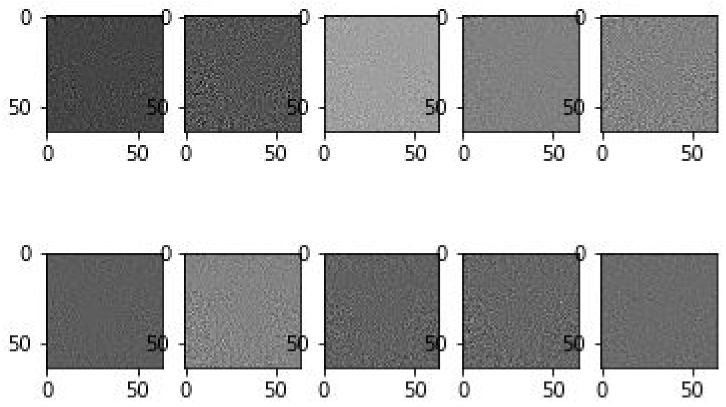
Reprojected images are almost same but we can see some data loss in the above picture. They are almost same because we took top 256 eigenvectors which contain almost all the data.

In the below picture, top ten eigenfaces are attached



Left top image correspond to topmost eigenvector, right bottom image correspond to top tenth eigenvector(Other images follow the pattern)

In the below picture, least ten eigenfaces are attached



Left top image corresponds to least eigenvector, right bottom image corresponds tenth eigenvector from last(All others follow the same pattern)

Conclusion:
From the visualisation of top ten and least ten eigenfaces, we can say that most of the features that contribute to a face are stored in eigenvectors with large eigenvalues and eigenvectors with least eigenvalues has noise. So, we can conclude that PCA(Principal Component Analysis) is a technique which reduces computational requirement without much data loss.

References:

- http://didawiki.cli.di.unipi.it/lib/exe/fetch.php/mcl/1992_turk_eigenfaces_for_recognition.pdf
- https://ieeexplore.ieee.org/document/139758
- Class notes
- http://vision.ucsd.edu/~iskwak/ExtYaleDatabase/ExtYaleB.html
- http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html