ECS797P COURSEWORK ASSIGNMENT 2: FACE RECOGNITION USING EIGENFACES

Animesh Devendra Chourey – 210765551

Queen Mary University of London – March 2022

Complete the lab2.m file

The main concept of the *eigenface* is that each face can be represented as a weighted combination of a face template ("basis" faces).

Eigenface can be used in two ways:

- **Data Compression:** We can store and then reconstruct a face from a set of weights.
- **Face Recognition:** We can recognise a new picture of a familiar face. Eigenface is the name given to a set of eigenvectors when used in the computer vision problem of human face recognition.

Eigenface is a method that is extremely 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 face without the full information and thus reducing the computation and space complexity. We find (learn) a set of basis faces (from a set of real training faces) which best represent the differences between them. A statistical criterion is used for measuring this notion of "best representation the differences between the training faces". The technique used is called *Principal Component Analysis*. Each face can then be stored as a set of weights for those basis faces.

1. Read in the training and test images

Images are loaded.There are 200 images in training dataset and 70 images in the test set. Each image in both the dataset is represented in high-dimensional (23x28) space. Each image in the training set of size NxN has to represented by vector of size N^2x1. Therefore every image of size (23x28) will be represented by vector of (23x28,1) which is (644,1).

2. Construct the mean image and the covariance matrix of the training image set

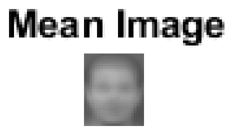
Firstly, the size of training set is calculated which gives us 200 value as there are 200 images in the training set. Then the means of all images is calculated which is of dimension (644,1). After the mean vector is calculated, the difference between the mean vector and the training images is calculated to obtain *CenteredVectors* variable. Through these *CenteredVectors* the covariance matrix is calculated. The *CovarianceMatrix* variable is of dimension (N^2xN^2) which is calculated by the formula $C = AA^T$ where C is *CovarianceMatrix* and A is *CenteredVectors*. Thus *CovarianceMatrix* is of (644x644) dimension.

3. Compute the Eigenface of the training set

Eigenface of the training set is calcualated by the eigenvalue decomposition. This operation is done by *svd()* which is singular value decomposition. From this, first 200 eigenvectors are used to represent the space. The eigenvalues are ranked from large to small and these eigenvalues are listed in the diagonal entries.

4. Display the mean image

The mean image is calculated by averaging the corresponding pixel of all the images. The corresponding code reshapes the mean vector to (28x23) matrix image.



5. Display the first 20 Eigenfaces

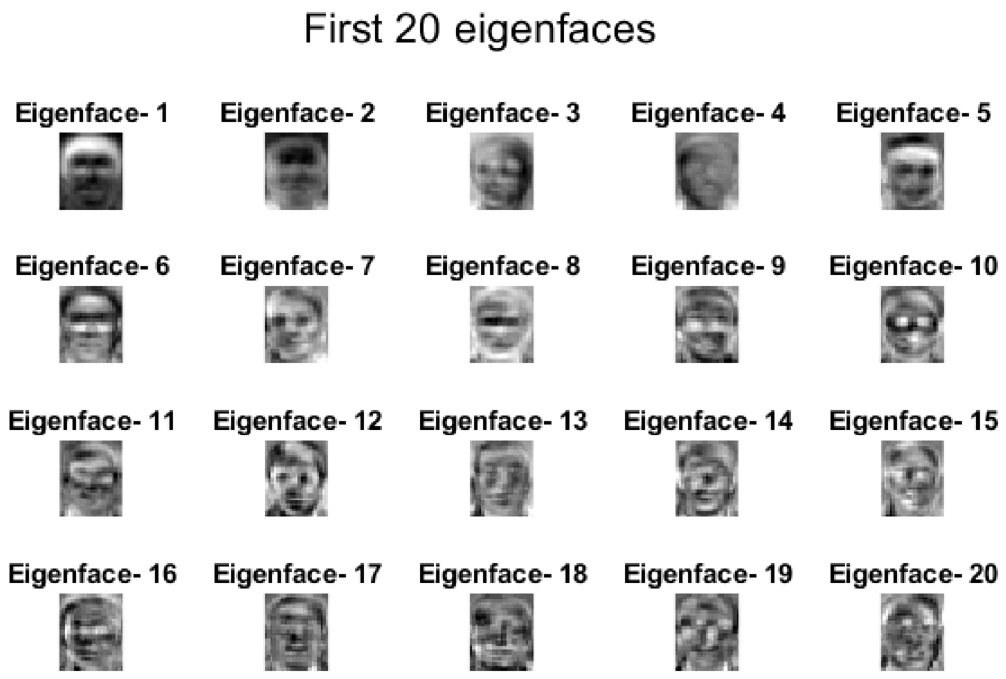
The first 20 eigenfaces are represented by the eigenvectors of the first 20 largest eigenvalues. As also previously mentioned these eigenvalues are ranked from large to small i.e. in descending order. The code to display the first 20 Eigenfaces is as follow:

```
figure;

for i = 1:20
    subplot(5,5,i);
    imshow(reshape(Space(i,:),[28,23]), []);
    title(['Eigenface- ',num2str(i)])
end

sgtitle('First 20 eigenfaces');
```

The first 20 Eigenfaces are as follow:



6. Project both training images and testing images onto the first 20 Eigenfaces.

With the function *projectImages()* given, we can project all the training and the test images onto the space. This function uses means and the eigenvectors calculated previously from the training dataset. The means is subtracted from both the training and the test data each and the resulting values are multiplied with the eigenvectors obtained from the training dataset. Here the threshold is set to 20. This tells us that the first 20 Eigenfaces will be used from the space.

7. Compute the distance from the project test images to the projected training images

The projected value of the training image is subtracted from the test image. Their difference is then squared. The resulting value is then added to the sum variable which gives us the distance between each test image to every other training image. After that, the code sorts the best training image associated test image which is done by the *sort()* function.

8. Display the top 6 best matched training images for each test image

Top 6 best matched training image for each test image is as follow:

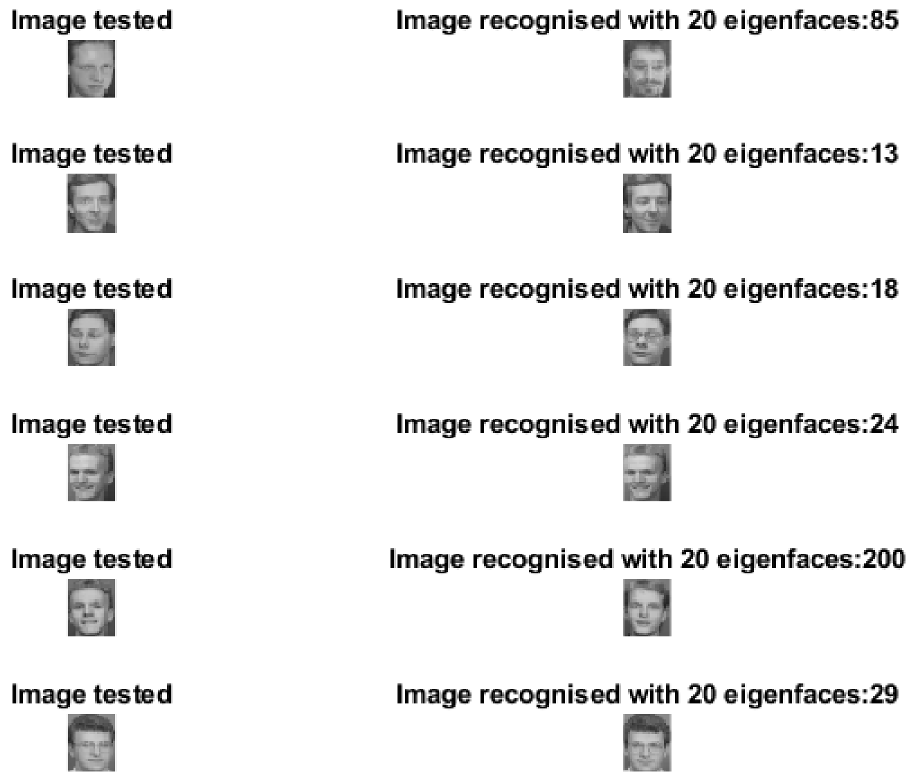


Image tested
Image recognised with 20 eigenfaces:85
Image tested
Image recognised with 20 eigenfaces:13
Image tested
Image recognised with 20 eigenfaces:18
Image tested
Image recognised with 20 eigenfaces:24
Image tested
Image recognised with 20 eigenfaces:200
Image tested
Image recognised with 20 eigenfaces:29

In the above figure, on the left side are the test images projections with 20 eigenfaces. On the right side are the best matched training images with 20 eigenfaces displayed. The figure shows the images that looks extremely similar to each other.

9. Compute the recognition rate using 20 Eigenfaces

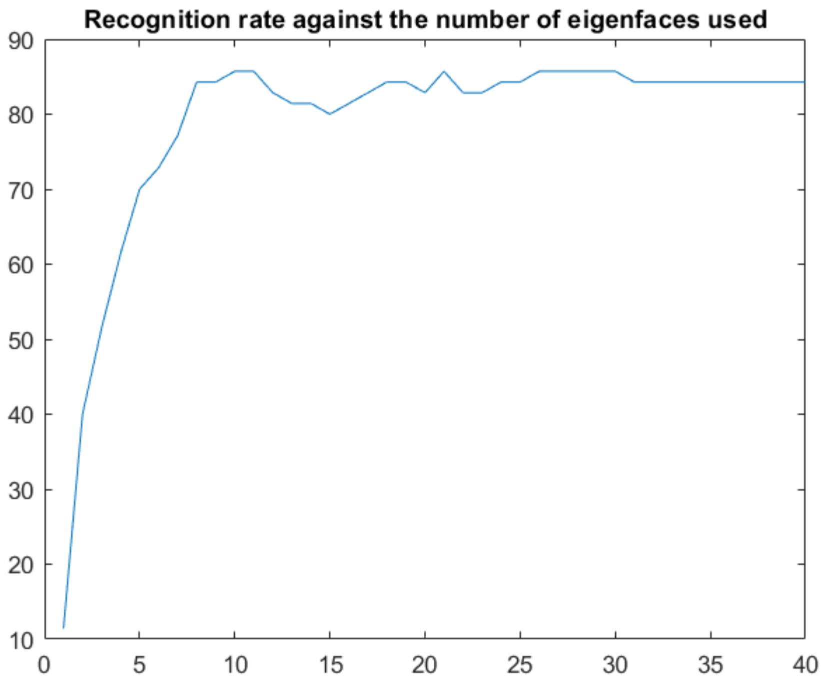
If the face is recognized correctly, the corresponding index is assigned 1, otherwise it is assigned 0. The recognition rate computed using the 20 eigenfaces gives us a mean of **82.85**.

```
% Initialising the rate for test set
rate = [];
```

```
for i = 1: length(Imagestest(:,1))
    % Compare the train indices with test identity. If equals then rate = 1
    if ceil(Indices(i,1)/5) == Identity(i)
        rate(i) = 1;
    else
        rate(i) = 0;
    end
end
```

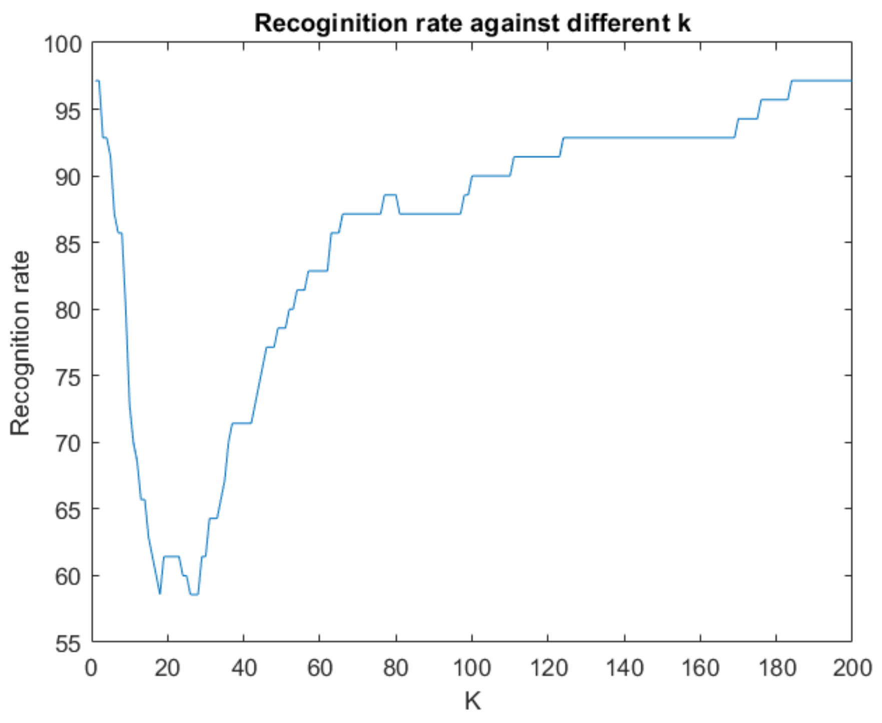
10. Investigate the effect of using different number of Eigenfaces for recognition

In this section, the recognition rate is calculated from 1 to 40 eigenfaces to get an estimate of accuracy. Here for 20 eigenfaces we are getting the same result as in the previous section. From the graph below we can see that the rate keeps on increasing until is reaches 8 eigenfaces and then from there on it keeps on fluctuating until around 20 eigenfaces. After that it almost stabilises neither improving nor damaging the performance. So, we can conclude that the optimum value for the number of eigenfaces can be obtained between 8 to 20.



11. Investigate the effect of K in K-Nearest Neighbour (KNN) classifier. Plot the average recognition rate against K

Here we can see that the recognition rate plunges down quickly as k increases in KNN. As k increases training images projections that are further away from the test image are also taken into account to perform the classification on the test images. This far away images can be considered as noise which results in incorrect recognition. However when the number of clusters are increased after a certain point each test image is assigned only to its corresponding training image which results in the recognition rate again increasing.



The code for the following part is displayed below:

```
TrainingLabels = []; % initialize the training labels
for i = 1:40
    TrainingLabels = horzcat(TrainingLabels, repmat(i,1,5));
end

% Overall Recoginition Rate for different k values
overall_recog_rate = [];


for k = 1:200 % iterating over k neighbours (i.e. from 1 to 200)
    %fit knn model to training data
    knn_model = fitcknn(Imagestrain, TrainingLabels, 'NumNeighbors', k,'BreakTies', 'nearest');
    knn_predict = predict(knn_model, Imagestest); % Prediction on the test data

    % Initalise recoginition rate for every k
    knn_recog_rate = [];

    for i = 1:length(Imagestest(:,1))
        % Compare the predictions with identity of test image and predicted
        if ceil(Indices(i,1)/5) == knn_predict(i)
            knn_recog_rate(i) = 1;
        else
            knn_recog_rate(i) = 0;
        end
    end

    overall_recog_rate(k) = ((sum(knn_recog_rate)/70)*100);
end
figure
plot(1:200, overall_recog_rate);
xlabel('K'); ylabel('Recognition rate')
title('Recoginition rate against different k');
```