Assignment 2: EigenFaces for Face Recognition

Ruchit Chudasama (20110172)

The Google Colab link of this assignment is given below.
https://colab.research.google.com/drive/1nnfLbu2uAbtEeCwU6sJKCA4hmZO1CuSC?usp=sharing

We used the approach mentioned in the research paper by Matthew Turk and Alex Pentland for face recognition using Eigenfaces. The algorithm was implemented using introductory matrix algebra and NumPy.

Dataset:

The dataset used was the AT&T dataset, where 10 different images of 40 distinct subjects were given. The 10 images of the same person were taken under varying circumstances, like different lighting, facial expressions and accessories.

We took two images of each of the 40 people for testing. The model was trained on 8 images of the first 20 people. We did this to calculate the accuracy for people inside and outside our training set.

On running the code on google colab, train_small.zip, test1.zip and test2.zip will appear in the contents folder. These signify the following:

1. dataset: training dataset contains 8 images of 20 different people.
2. test1.zip: testing dataset contains 2 images of all people from the training dataset.
3. test2.zip: testing dataset, consisting of 2 images of 20 new people whose images weren't present in the training dataset.

Training Methodology:

1. We start by grayscaling the images and converting them into matrices of shape 112x92.
2. Each matrix was flattened and converted into a matrix of shape 10304x1. (Here 10304 comes from 112*92). All these vectors were stacked row-wise into a single matrix (image_1D in our code).
3. Now we normalize each row of this matrix by subtracting the row-wise mean from each element of the corresponding row. This new matrix will be called A^T, and the transpose of this matrix will be called A.
4. Next we calculate the covariance matrix by doing $A^T x A$.
5. Next we calculate the eigenvalues and eigenvectors of the covariance matrix using the linalg.eig of NumPy. The following equations were used:

$$A^T A\nu_i = \lambda_i \nu_i$$
$$AA^T A\nu_i = \lambda_i A\nu_i$$
$$C' u_i = \lambda_i u_i$$

   where
$$C' = AA^T$$

   and
$$u_i = A\nu_i \qquad \text{—---------1}$$

   C' and C have the same eigenvalues and their eigenvectors are related by equation 1.
   So, using equation 1, we get the Eigenvector and Eigenvalues of this reduced covariance matrix and map them into the C'.
6. Next step is dimensionality reduction. We choose a number of K and K eigenvectors corresponding to the K largest eigenvalues.
7. We calculated the normalized training faces (face-average face) and represented each normalized face as a linear combination of the eigenvectors obtained in step 6.

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

These w vectors were calculated using the `np.linalg.lstsq` function of NumPy.

8. After calculating the weights (w vectors), we stacked those vectors and represented the training faces as follows:

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

Testing:

1. We start by gray scaling and resizing the test image to fit our algorithm.
2. Next, we normalize the test image by subtracting the mean face from our unknown face.
3. This normalized vector is projected into the eigenspace to obtain the linear combination of the eigenfaces.

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

4. We stack the w vectors obtained as follows:

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

5. We take this vector and subtract it from the training images to get the minimum distance between the training vectors and testing vectors.

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

6. If this e_r comes out to be lower than the set threshold, then we find which face it is most similar to in the training images, else we report that the test image does not match with any image in the training set.

Assumptions:

1. We took a comparatively smaller dataset, because bigger datasets were taking larger runtimes.
2. The threshold value for classifying the test dataset is taken as the average value of the minimum and maximum values of the errors in the true test dataset.
3. The error rate that we have defined is according to the number of incorrect classifications that were made by the algorithm.

Accuracy Calculation:

We tested our model on people both inside and outside our training set and calculated true_positive, true_nagative, false_positive and false_negative.
If our test image is within our training dataset, we will take positive, else negative.
Then the metric we have defined to measure the error rate is given as follows:
incorrect_classifications = false_positive + false_negative
And, error_rate = $\frac{incorrect\_classifications}{test\_size} * 100$

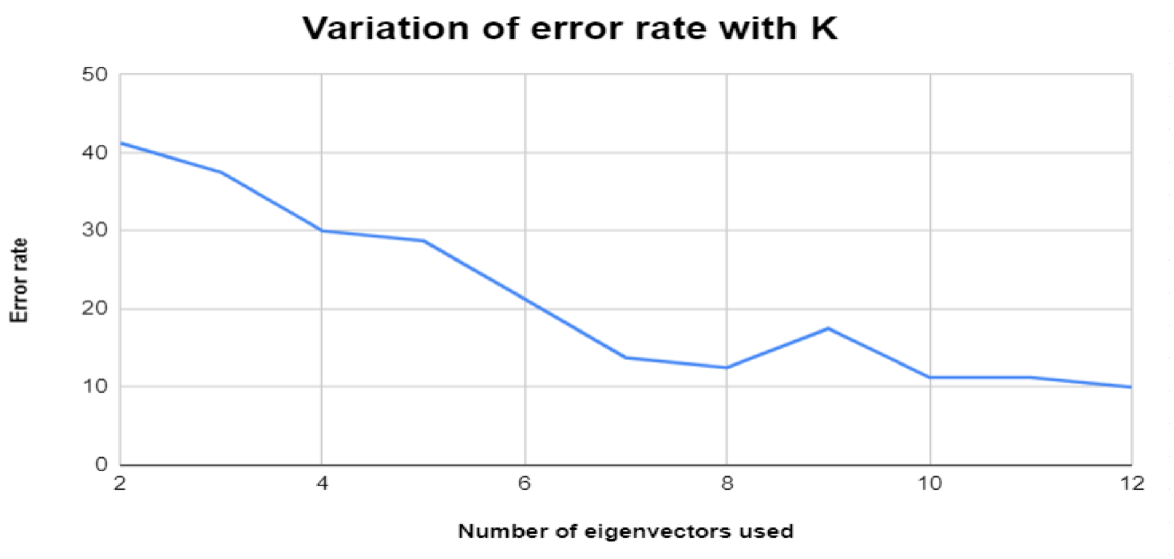
Advantages:

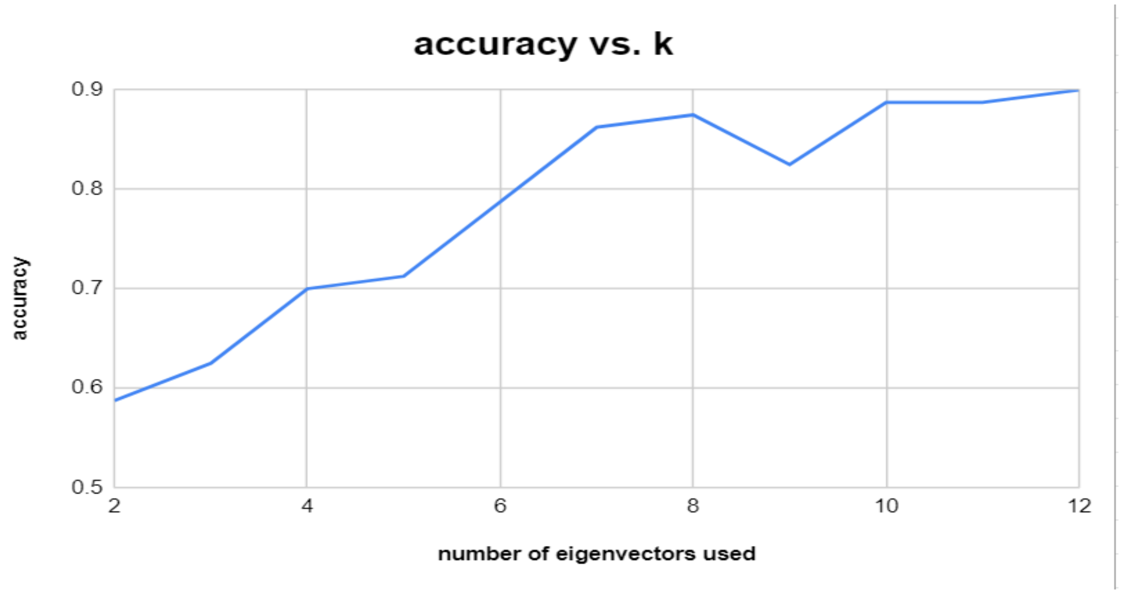
1. Simple to implement and less expensive in terms of computing.
2. No prior knowledge of the image such as face features is necessary. Only image_ID is required.

Limitations:

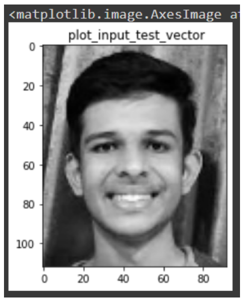
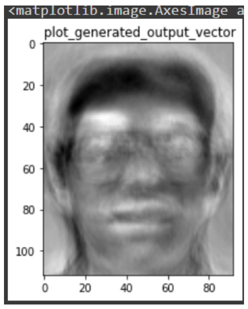
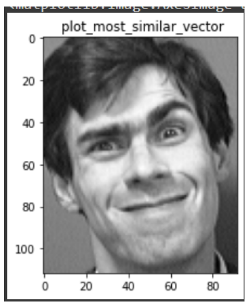
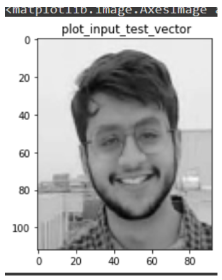
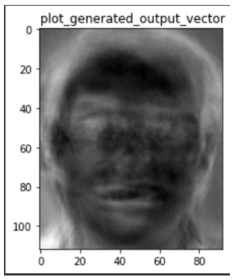
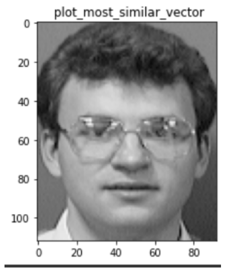
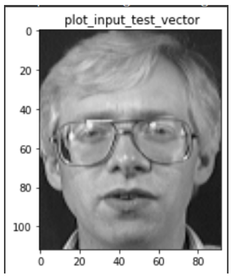
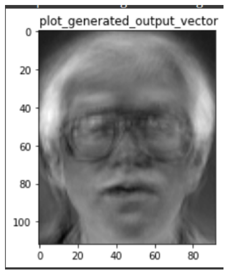
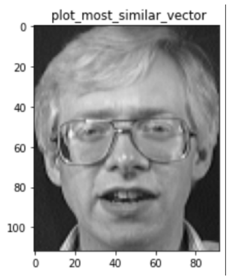
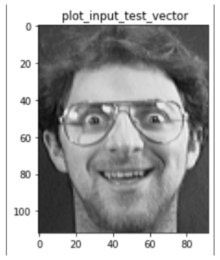
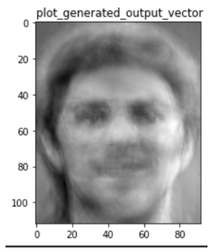
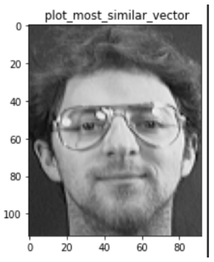
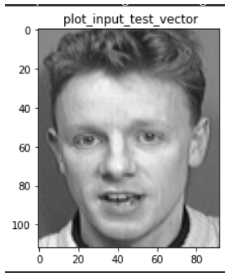
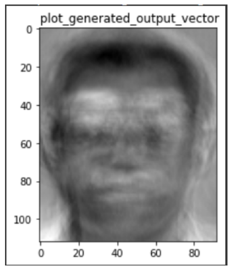
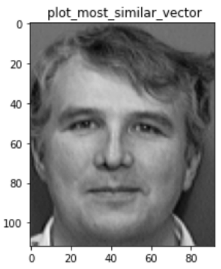
1. The algorithm is sensitive to the amount of light and shadow in an image as well as the proportion of the facial area in the test image.
2. This algorithm must have a front view of the face in order to function effectively.
3. The larger datasets take more time for computation.

Results:





The algorithm is tested on a few images and the results are tabulated as follows. Image 1,2 and 5 are of the people outside the training dataset and hence the generated output predicts the person in the dataset that most closely resembles them. Image 3 and 4 are of the people from the training dataset (but not exactly the same image from the training data). Hence we see that the algorithm correctly detects the corresponding person in the image. Note that the generated output is an image that is a linear combination of the k eigenfaces.

	Input Image	Generated Output	Most similar Image
1		 <matplotlib.image.AxesImage a plot_generated_output_vector	
2			
3			
4			
5			

References:

1. https://www.geeksforgeeks.org/ml-face-recognition-using-eigenfaces-pca-algorithm/
2. https://ieeexplore.ieee.org/document/139758
3. https://ieeexplore.ieee.org/abstract/document/6793549
4. I did this assignment along with Aryan Gupta (2011026) and Aditi Dey (20110007), so our codes and reports will be similar.