CSE 691: Image and Video Processing

Spring 2020 Assignment 5

Eigenface

Lichen Liang

03/22/2020

Objectives

- Understand Principle Component Analysis and Eigenface.
- Understand each step of the algorithm and what does it do.
- Know how the variables affect the output.
- Apply PCA.

Method

To run the code, run ***main.m*** and make sure all function files are in same folder.

1. Out of the 165 faces, I randomly picked 145 training images and the rest 20 images are used for testing. First, read all the training images. These images are reshaped to 50% of their original size. The reason will be explained in Results Discussion section. Then reshape every image into a column vector. Each vector is then placed in a matrix X, with number of rows equal to the size of the image, and number of columns equal to the number of images. For example, a 10x10 image will be reshaped into a 100x1 column vector and there are 145 such images, so X matrix will have the size 100x145. In this case, the size of the image is 77x58 = 4466 rows.

   The same method is applied to reading test images and non-face images.

2. Find the average face by using the average function. It calculates the mean for each row. The result will be a column vector. The function returns this vector as well as the reshaped version of this vector, so we can visualize the average face.

3. There are three ways to calculate the principle component. The first method (function *PCAA())* get the new X by subtracting the average face from it. Then calculate the covariance matrix as C=XX'. Find the eigenvalues and eigenvectors of the covariance matrix. The eigenvalues are in a diagonal matrix. Make it into a column matrix and sort the column in descending order. At the same time, update the order of eigenvectors as well. Finally, normalize the eigenvectors. These eigenvectors are the principal component or we call eigenfaces.

4. The second method (*function PCASVD())* uses singular value decomposition. If M is a matrix, then represent M by using three matrices: unitary matrix U, diagonal matrix S, and eigenvectors V such that M = USV'. We need S and V in our case.

   Similar to part 1, subtract the average face from X. the covariance matrix C is calculated from $C = \frac{X^T}{\sqrt{number\ of\ images-1}}$. Then use the matlab function *svd()* that automatically calculates the three matrices U, S, and V. Form S into a column vector, sort the eigenvalues and their corresponding eigenvectors, and normalize the eigenvectors.

5. The third method (*function PCA())* is similar to part 1, but the covariance matrix is calculated by C=X'X. Similarly, calculate the eigenvalues and eigenvectors. However, the eigenvectors V in this part has a size of 145x145, whereas eigenvectors from part 1 has size of 4466x4466. So we need to multiply X by V to get it into 4466x145 before normalizing.

6. Although all three methods calculates the principle component, their execution speeds varies by a large factor. I will discuss about this in the Result Discussion. For you information, I used the function in part 3 *PCA()* for the upcoming steps.

   Now we have to defined a number K, where K is the number of eigenfaces that we use to reconstruct or recognize faces.

7. The next step is to calculate the weight from K eigenfaces. The function *getweight()* gets the first K eigenvectors from our PCA results. Note that in this part, I have already created a matrix Xtest where each column is a test image from test folder and this matrix already has its own average subtracted (Refer to section 1). Using the first K eigenvectors, apply to Xtest by multiplication. This product will be the weight.

8. To recover or reconstruct the faces, multiply the weight and the K eigenvectors. Then add back the average face we subtracted earlier. The output is dependent on the size of K. This will be discussed later.

9. To recognize the test images from the original training images, we first need to find the new weight. This weight uses original matrix X rather than test images matrix Xtest. Then call the function *recognize()*. First, subtract the average face of the test faces from the test faces. Multiply by the weight we got from the original images to get a new weight for the test images. We now have two weights matrix: new weight with size 40x20 and weight from original matrix 40x145, where 40 is our K.

   For each image(column) in the new weight, calculate the Euclidian norm with each in the old weight. Then find the minimum of the norm we calculated and save that index to a new array. The image at this index is calculated to be the match. For example, use first image(first column) from the new weight, calculate the norm with each column in the old weight, and store this norm in a result array. This result array will have 145 entries. Find the minimum and save its index to a index keeper array. Assume this index of result array is 5, then the first image(also the 1st index in index keeper) will have a value of 5. This means the first image of test images is matched to 5th image in training images.

   With the non-face images, the steps are the same.

10. In the end, there is a part to calculate the Frobenius Norm. First, get the absolute value of the difference of original(for test and non-face images) and their reconstructed matrix. Then calculate the Frobenius Norm and plot it.

Results and Discussion

Speed

The three different ways to calculate the principle component varies a lot. If I did not resize the images to 50%, the first method roughly takes 30 minutes because it has to calculate eigenvalues and eigen vectors of a roughly 18000x18000 matrix. Currently with the image sizes reduced, the first method takes about 24 seconds, the second method takes about 1.4 seconds, and the third takes about 0.04 seconds. Keep in mind that this is only for 145 images. If we have thousands of imaged, then first and second method would take hours to complete, which can be hardly implemented.

I used the third method to calculate the principal components throughout this assignment.

Reconstruction and effect of K

I have tested multiple values for K: 1, 5, 10, 20, 40, 100 to see the effects. Also, the original test images to be compared to in figure 1.

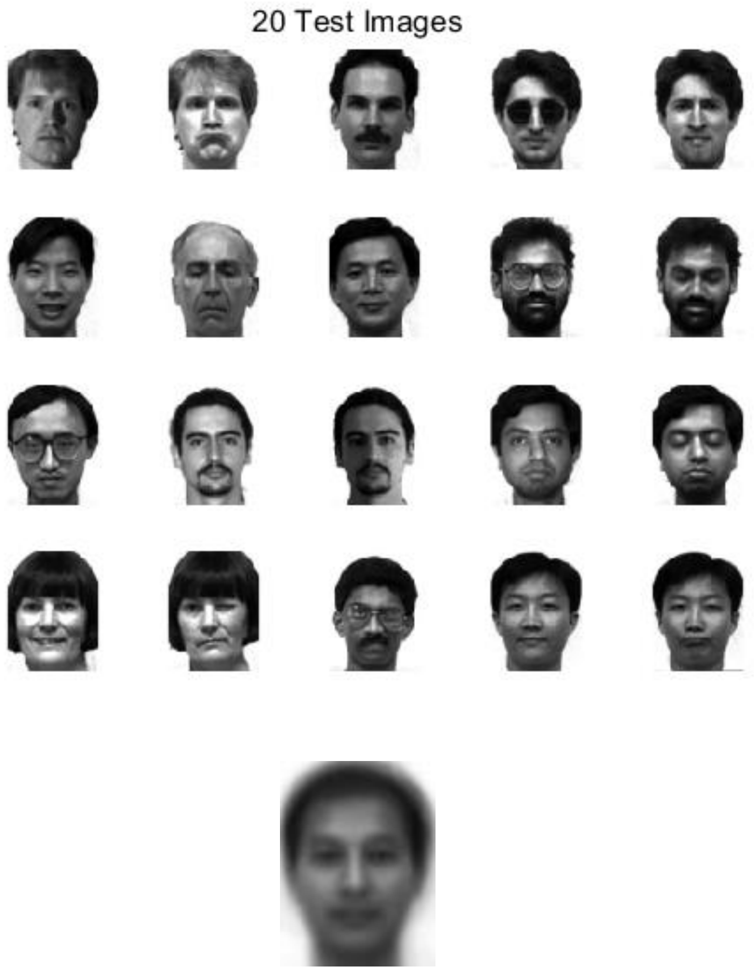

Figure 1. Original 20 Test Images and its average face

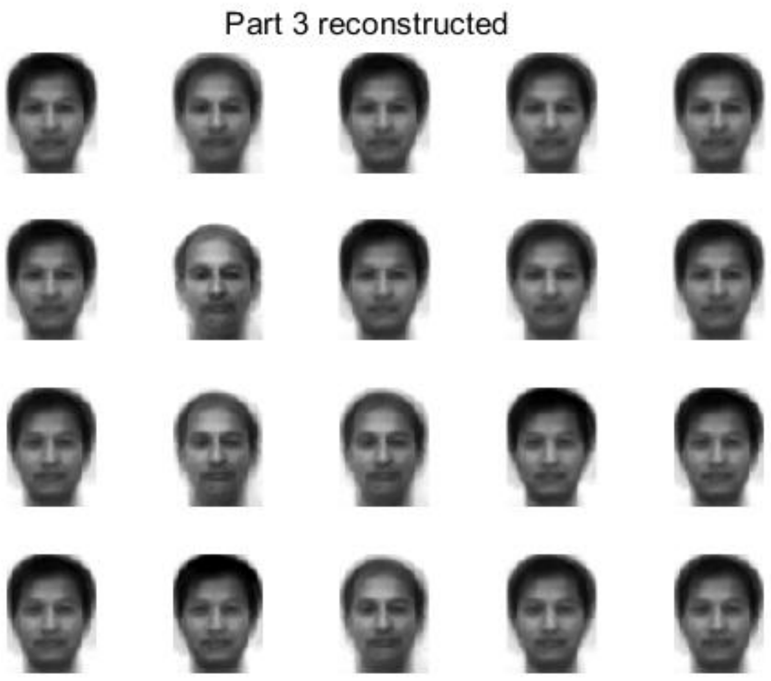

Figure 2. 3rd method reconstructed image with K = 1

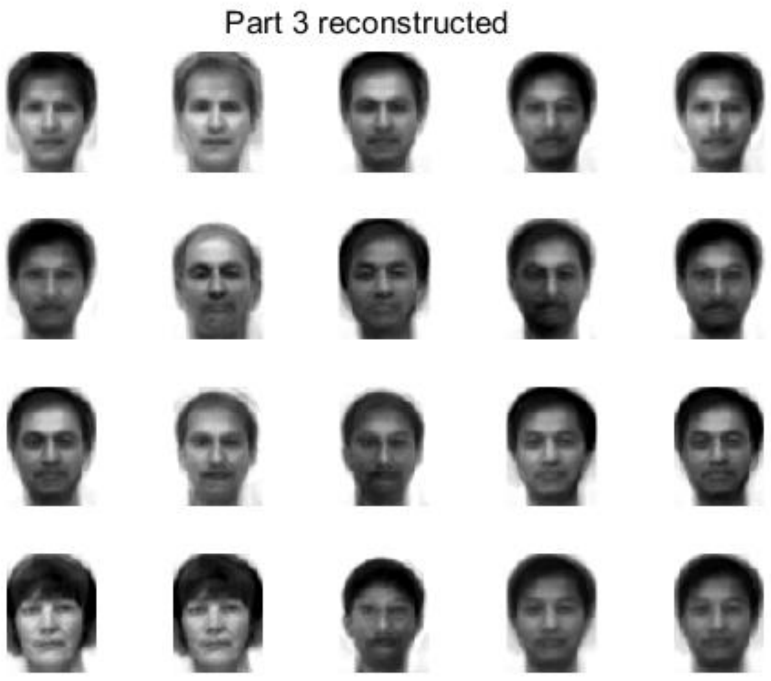

Figure 3. 3rd method reconstructed image with K = 5

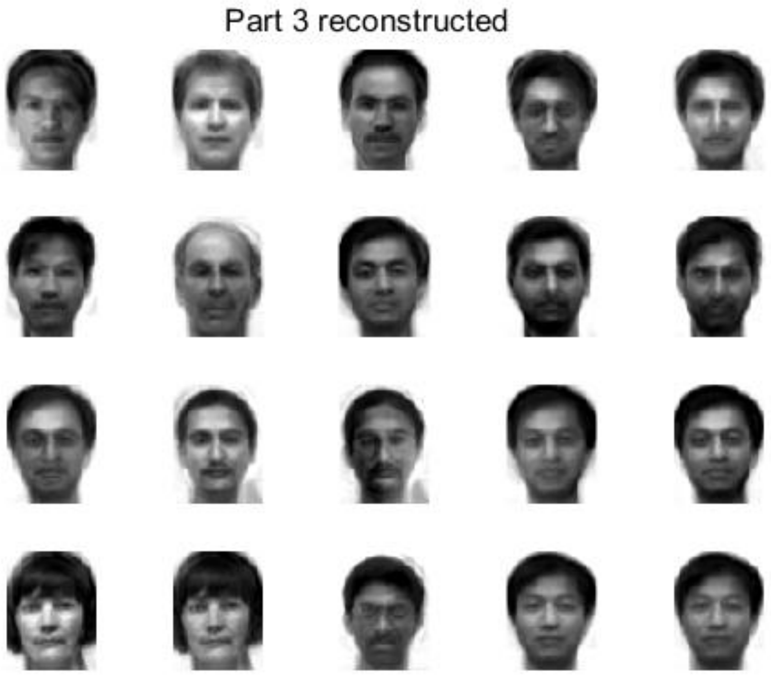

Figure 4. 3rd method reconstructed image with K = 10

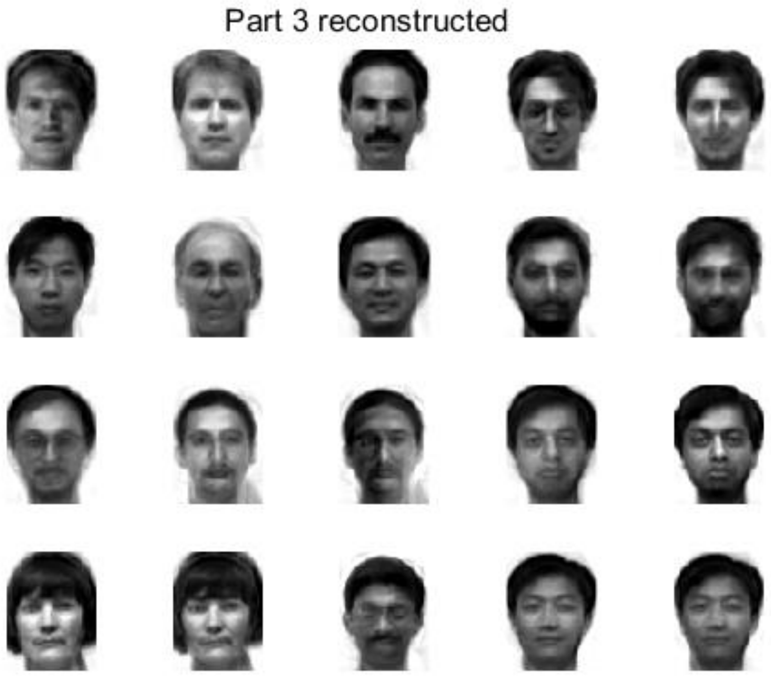

Figure 5. 3rd method reconstructed image with K = 20

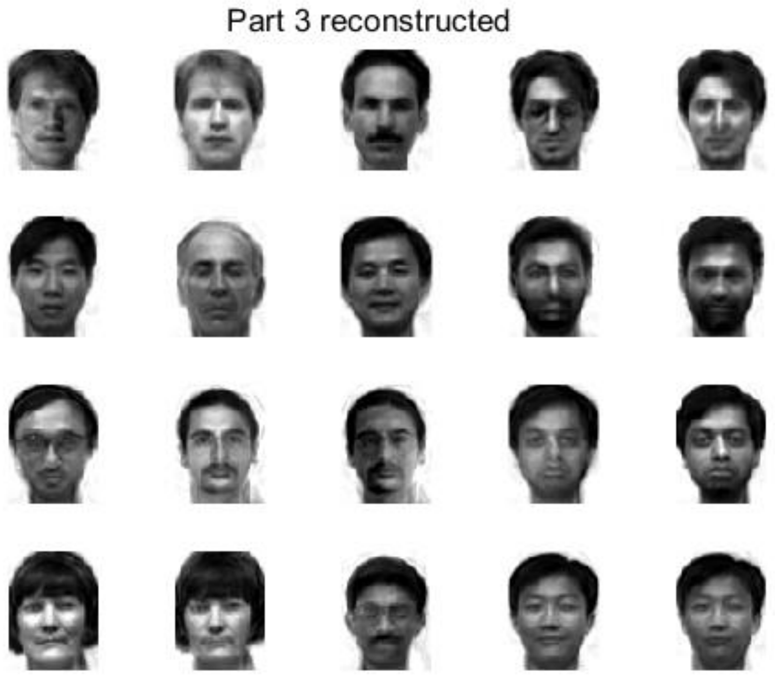

Figure 6. 3rd method reconstructed image with K = 40

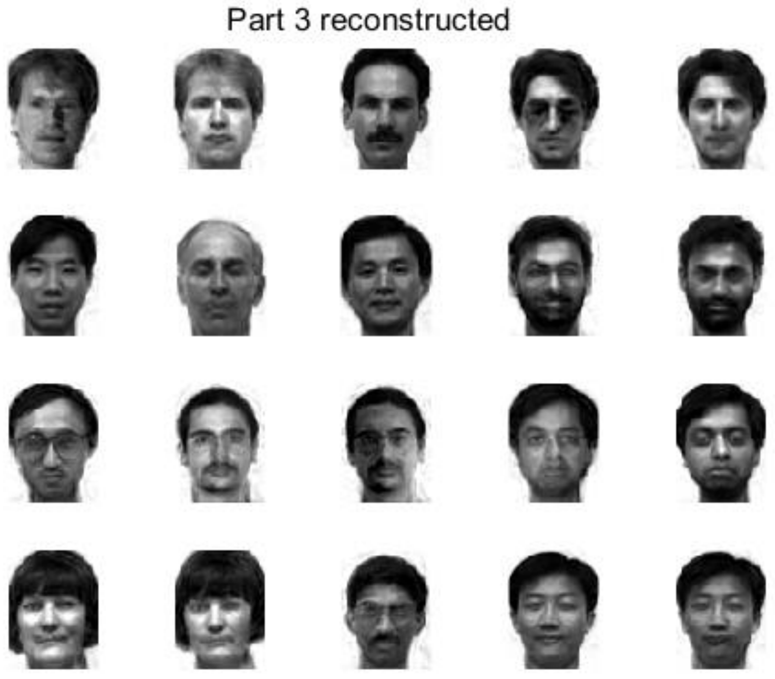

Figure 7. 3rd method reconstructed image with K = 100

Starting from figure 2 to figure 7, compare with figure 1. We can see that when K is small, it is very difficult to differentiate the images between different people. When K=1, almost everyone has the average face; and when K=5, there are minor differences but still hard to tell. Starting from K=10, we can start to identify individuals by looking closely and at K=20, we can pretty much identify each individual at a glance. However, some details such as emotion is still unclear. Looking at K=40 and K=100, we can see that some emotions or dark glasses are being formed. However, it does not fully reconstruct the image. Overall, to be able to visualize and differentiate the faces, we need K to be at least equal to 10.

Next, I will show the results from first and second method to see if there are any difference between the methods we use.

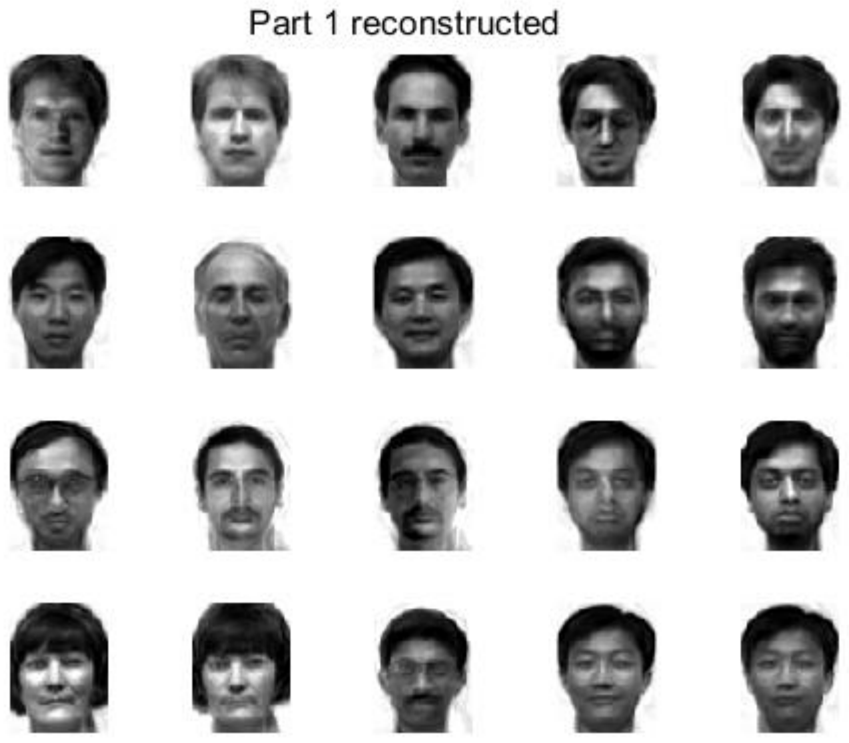

Figure 8. 1st method reconstructed image with K = 40

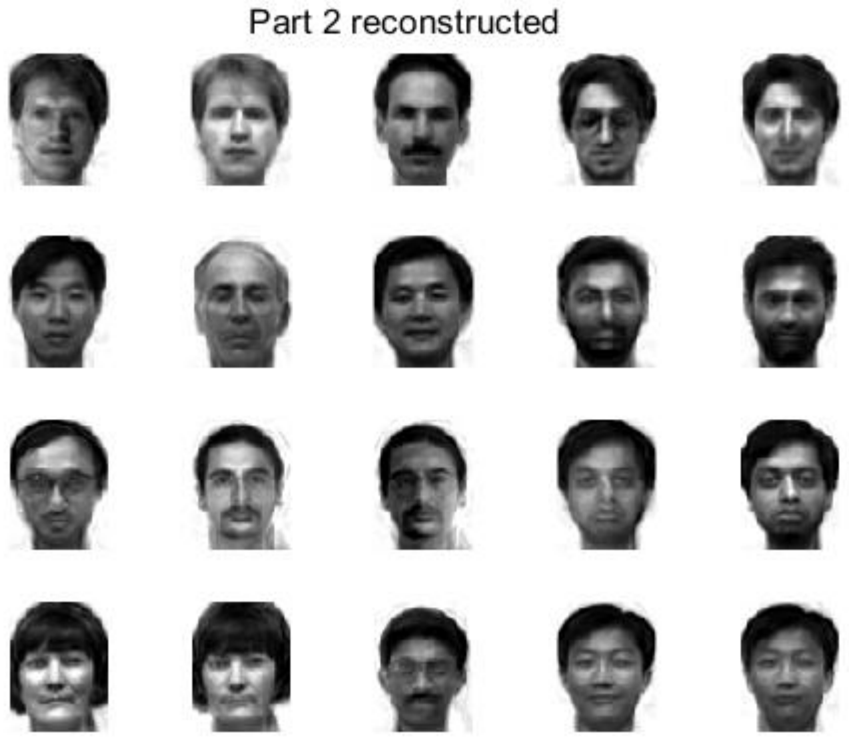

Figure 9. 2nd method reconstructed image with K = 40

Comparing figures 6, 8, and 9, which all uses K=40, we can see that all three methods have the same output. In other words, we can use any method to get the principle components, and choosing the one that takes the least amount of time of execution would be wise.

Recognition

I have tested for two K values 13 and 40 to see how they differ in the recognition result.

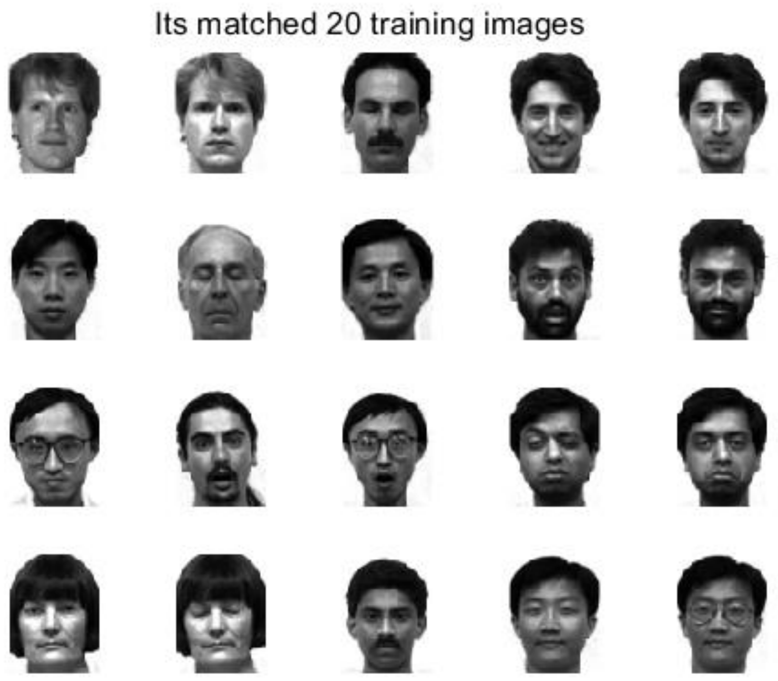

Figure 10. Recognition result when K=13

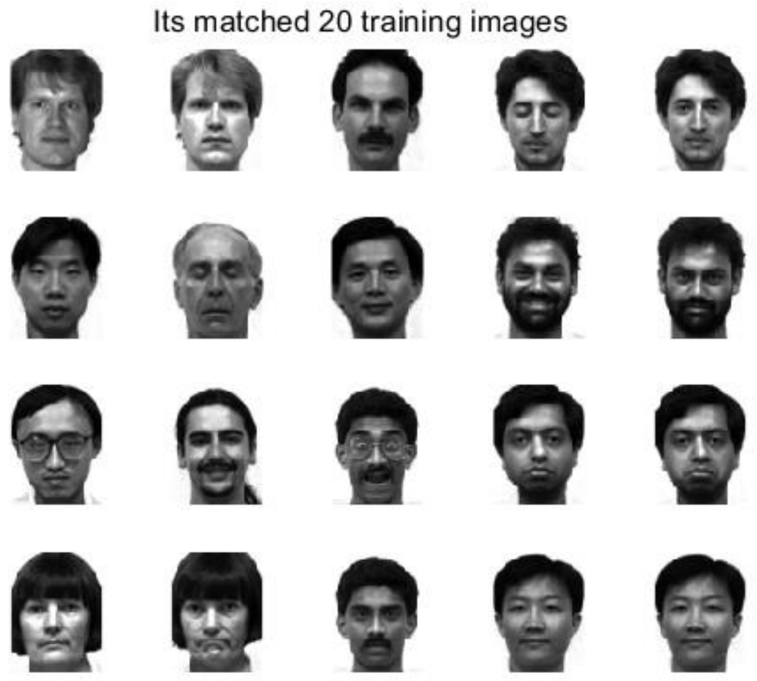

Figure 11. Recognition result when K=40

Comparing the figures 10 and 11 with the original test images in figure 1, we can see that 19/20 faces are recognized correctly. The only wrong detection happened in both figures at (3,3) where none of them matched to the original image. Some possibility that might cause this failure is because of the size of the image or similarity in the grayscale values.

Reconstruction using Non-face images
I reconstructed the faces using non-face images with K values 40 and 140.

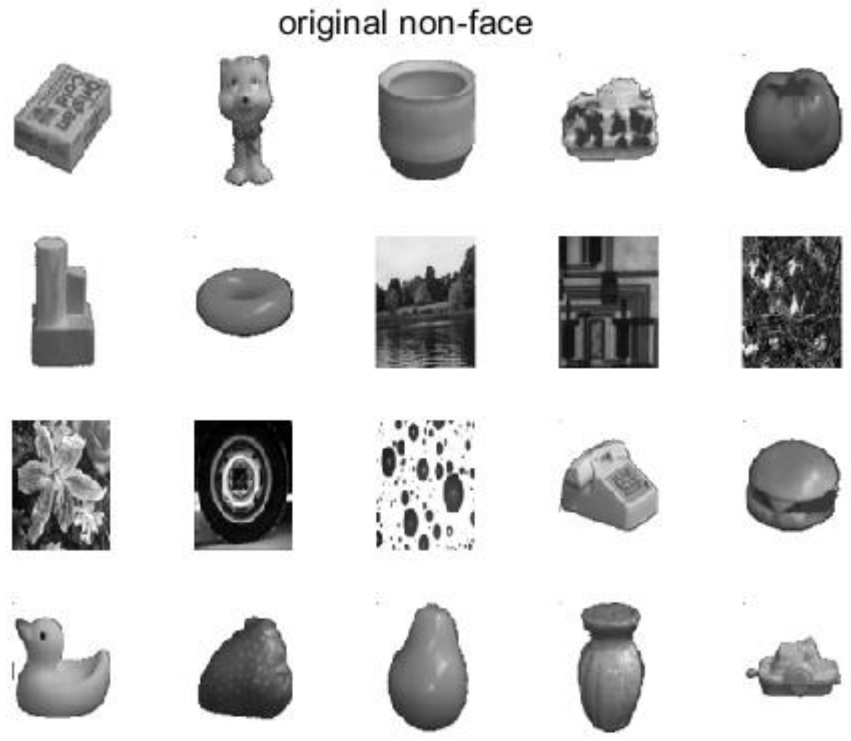

Figure 12. Original 20 Non-face Images

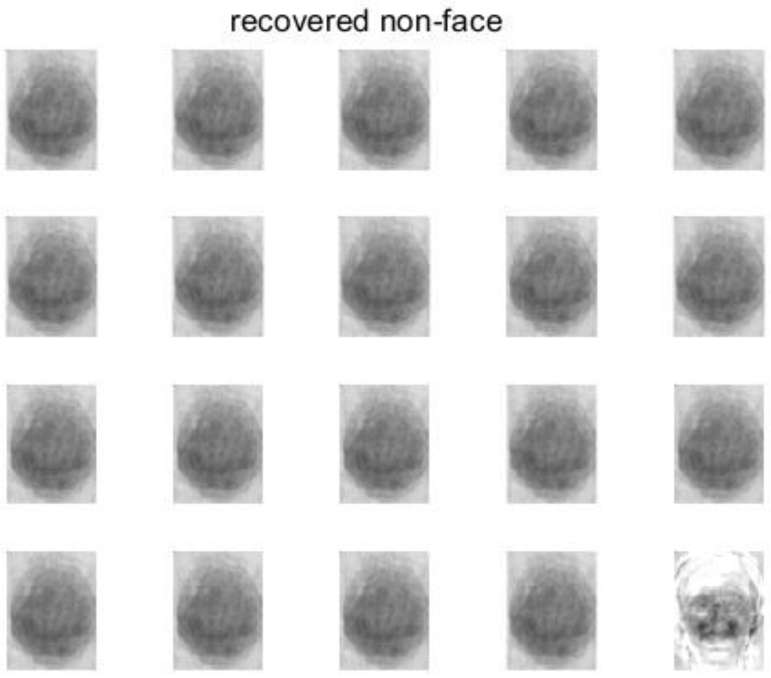

Figure 13. 3rd method reconstructed image with K = 40 using non-face images

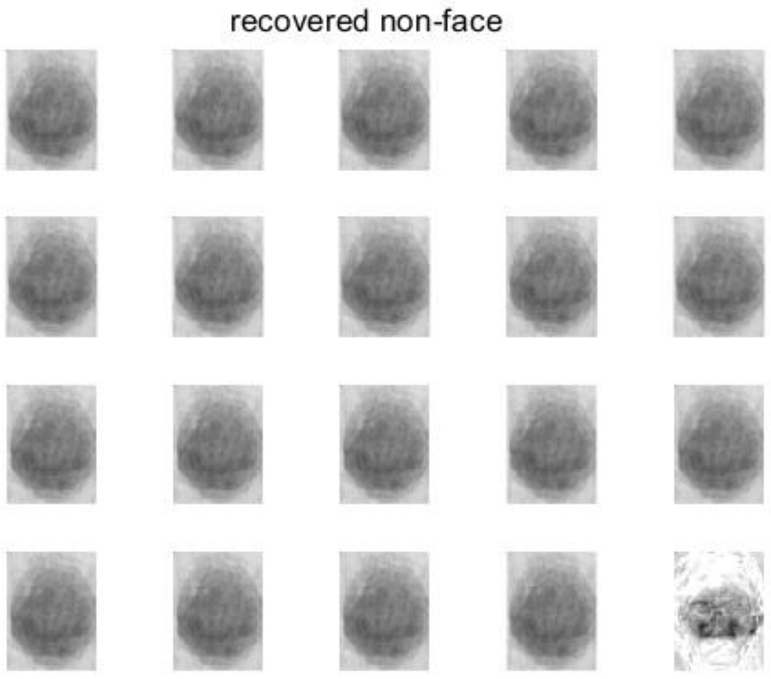

Figure 14. 3rd method reconstructed image with K = 140 using non-face images

The results are expected and clear. Comparing figures 13 and 14 to figure 12 with original non-face images, we cannot reconstruct a face out of a non-face images despite the number of eigenfaces we use.

Frobenius Norm

In this part we calculate the difference between original and reconstructed image, and take the Frobenius norm of the difference image. For non-face original and reconstructed image, refer to figures 12 and 13. For test face original and reconstructed image, refer to figures 1 and 6.

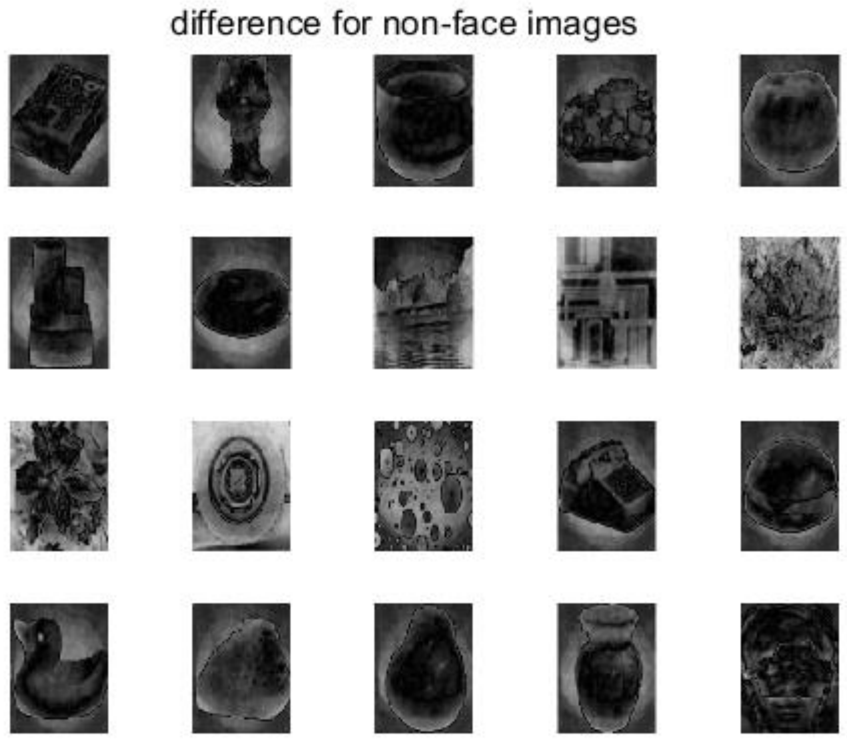

Figure 15. Difference image for non-face images

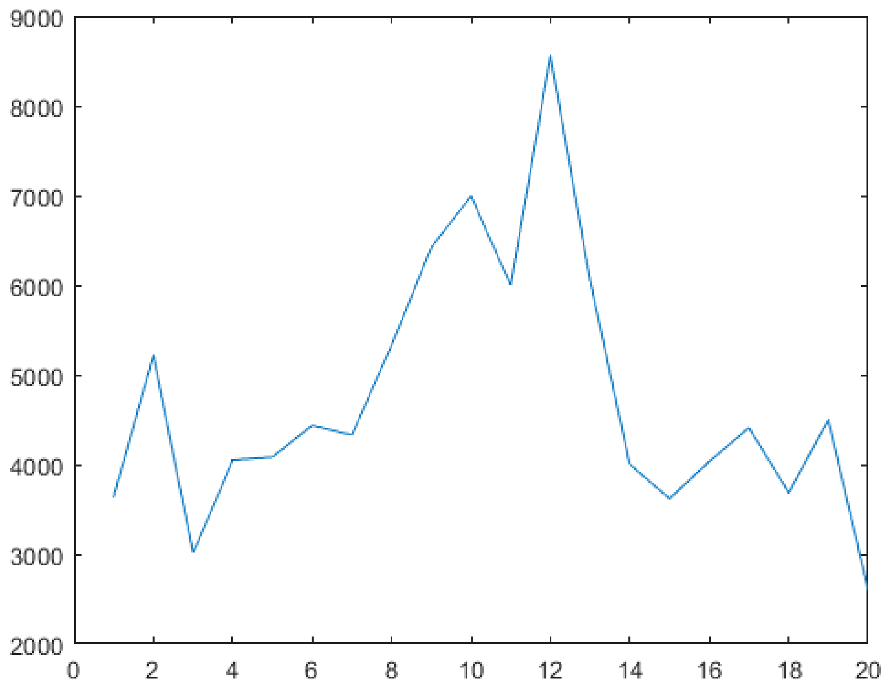

Figure 16. Frobenius Norm for Non-face difference(fig 14)

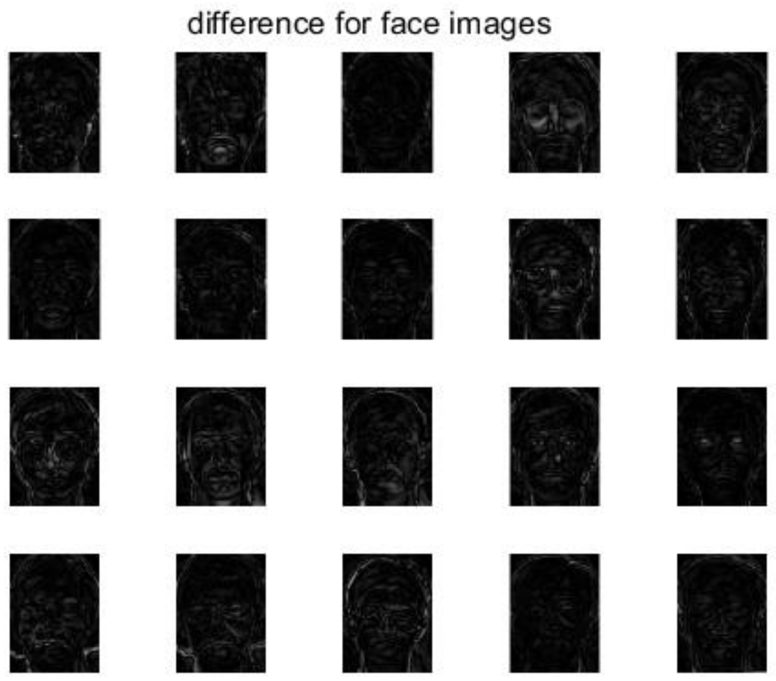

Figure 17. Difference image for face images.

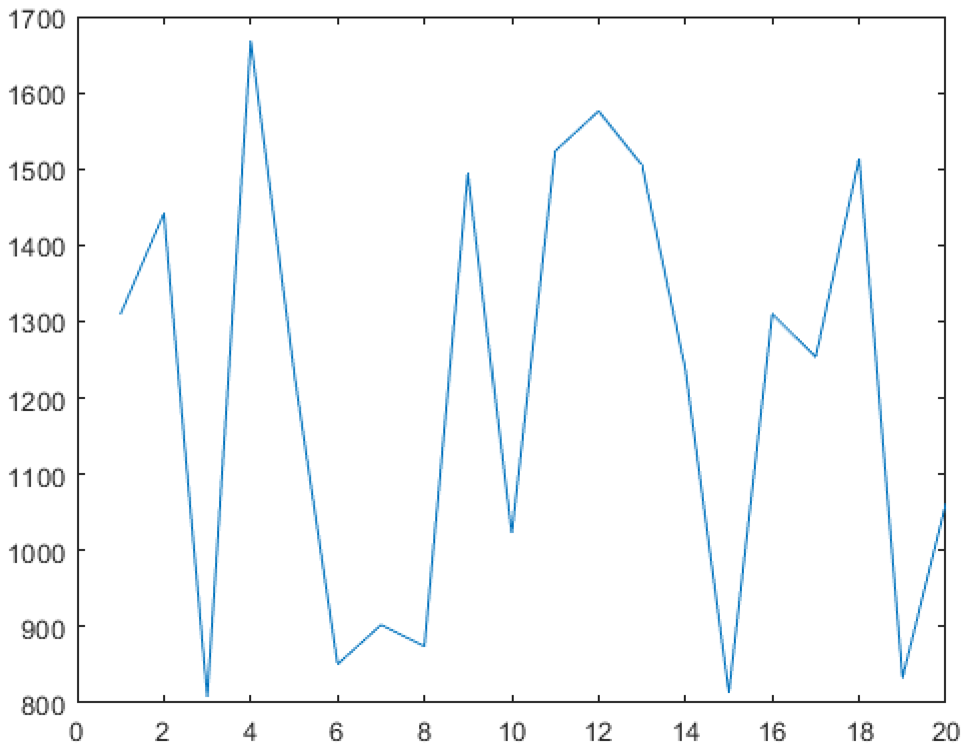

Figure 18. Frobenius norm for face images difference(fig 16)

From our observation from above, we can see that the difference image for face images is very small. This means the reconstruction is successful. The only difference is at the edges of the face, which demonstrates the difference in emotions.
The difference images for non-face images is very visible. This means that we cannot reconstruct a face from a non-face image. Comparing the Frobenius norm, the x axis represent 20 images and y axis represent the norm values. We can see that the norm for non-face images is much higher than the norm of face images. The maximum norm for face images is ~1670 and the minimum norm for non-face images is ~2700.

Conclusion

In this assignment we computed the principal components of faces as eigenfaces and projected onto different test data. Few major observations we can conclude:

- Using X'X is fastest way to calculate eigenfaces
- The test image must have a similar image in the training data.
- The higher the K value, the better the reconstruction.
- The higher the K value, the better the recognition.

- The effect of wearing glasses is very minor even if we exclude images with glasses.
- The difference in the non-face with its reconstructed is very high compared to face with its reconstructed. Therefore, the Frobenius norm is also much higher.