Report

[Group member]

Inyeob Kim, Michael Jones

[Introduction]

My job was to implement necessary functions for the project. As soon as each function was implemented, Michael did some testing to determine whether the function works correctly. We had several errors while working on this project, but we were able to fix them by working together from the start to the end.

[Section 1]

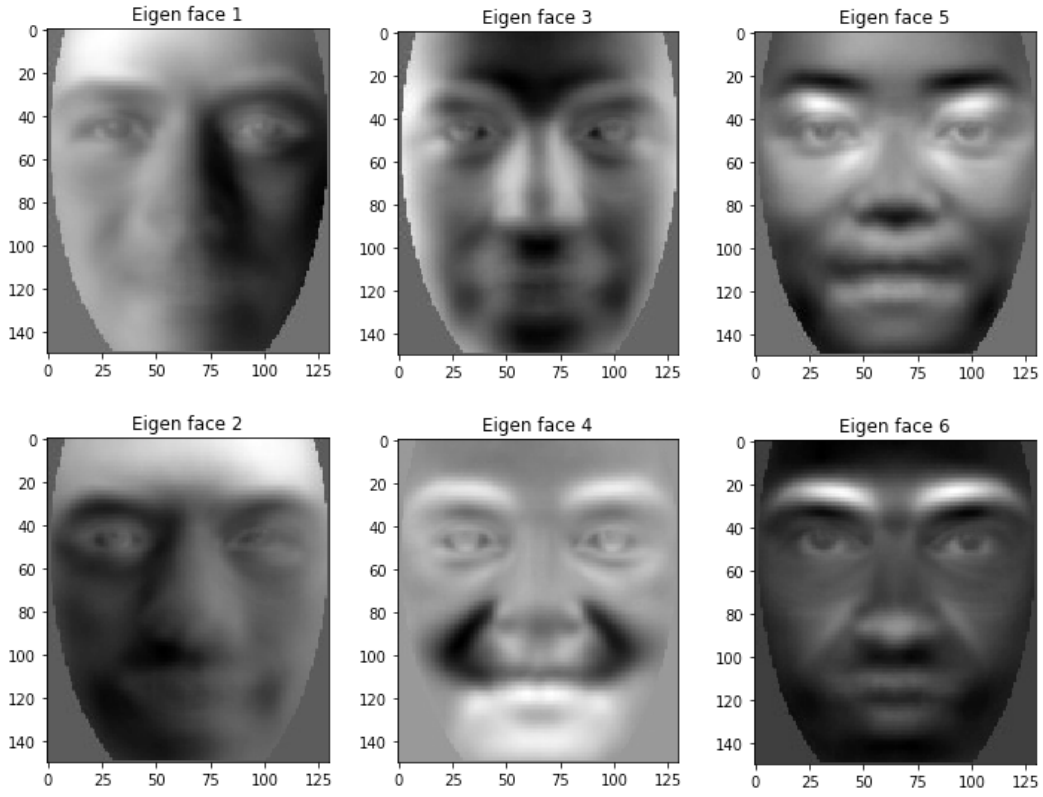
Principal Component Analysis is to find the largest K number of eigenvalues and eigenvectors that captures a certain percentage of variance. Here, the largest eigenvalue will be PC1, which has the largest variance coverage, and the second largest eigenvalue will be PC2, which has the second largest variance coverage, and so on. We implemented a function called get_best_eig_vecs(images, target_variance) in order to obtain the best K eigenvalues along with the corresponding eigenvectors that captures target variance. For instance, if we pass in training images and a target variance of 90% into the function, it yields K of 265. This means that eigenvalues from $\lambda_1, \lambda_2, \ldots \lambda_k$ captures at least 90% of the variance. Even though it loses 10% of the information of the original images, it has an advantage of reduced dimension. The function returns 265 largest eigenvalues and corresponding eigenvectors. Implementing PCA was relatively simple. Let's assume that a matrix of the training images is called A. Since, the

dimension of AA^T is massive compare to that of A^TA, we computed eigenvalues of A^TA, using the property of two matrices share the nonzero eigenvalues. We could obtain eigenvectors of A^TA by multiplying A by the eigenvectors of A^TA.

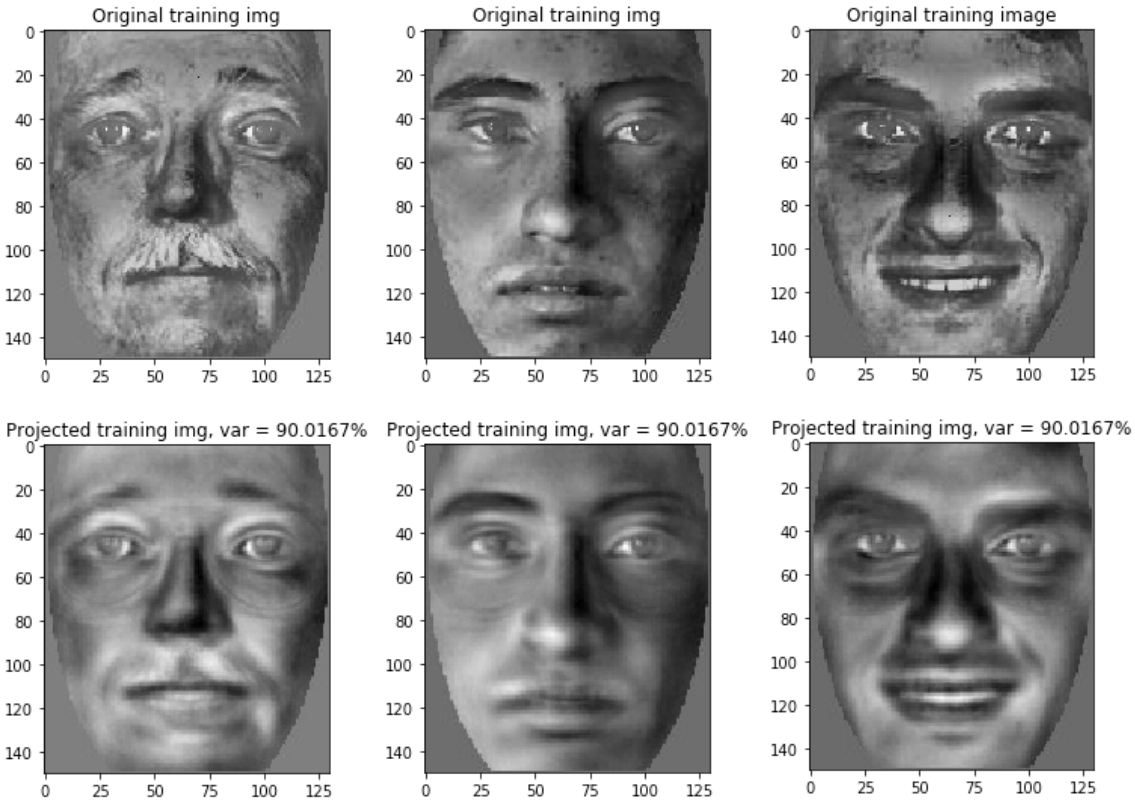
```
Variance capture percentage: 90.016705%
K =  265
K best eigenvectors:
 [[ 2.84924670e-05  2.89520239e-05  1.28034956e-04 ...  7.82715282e-04
   9.19499238e-04 -1.20206437e-03]
 [-1.75881945e-04 -1.99725306e-04 -1.89706620e-04 ...  3.04504476e-04
   3.70370960e-04  3.63911197e-04]
 [-1.03921376e-04 -9.43555275e-05  2.87133738e-05 ... -3.83187455e-05
  -5.95345075e-04  7.38290150e-05]
 ...
 [-3.07412973e-06 -2.29153526e-06  4.13601008e-05 ...  6.26102209e-05
  -5.45910630e-06 -1.27595864e-04]
 [-3.07412973e-06 -2.29153526e-06  4.13601008e-05 ...  6.26102209e-05
  -5.45910630e-06 -1.27595864e-04]
 [-3.07412973e-06 -2.29153526e-06  4.13601008e-05 ...  6.26102209e-05
  -5.45910630e-06 -1.27595864e-04]]
```

[Section 2]

Here are some of the eigenfaces that we obtained (90.016705% variance).

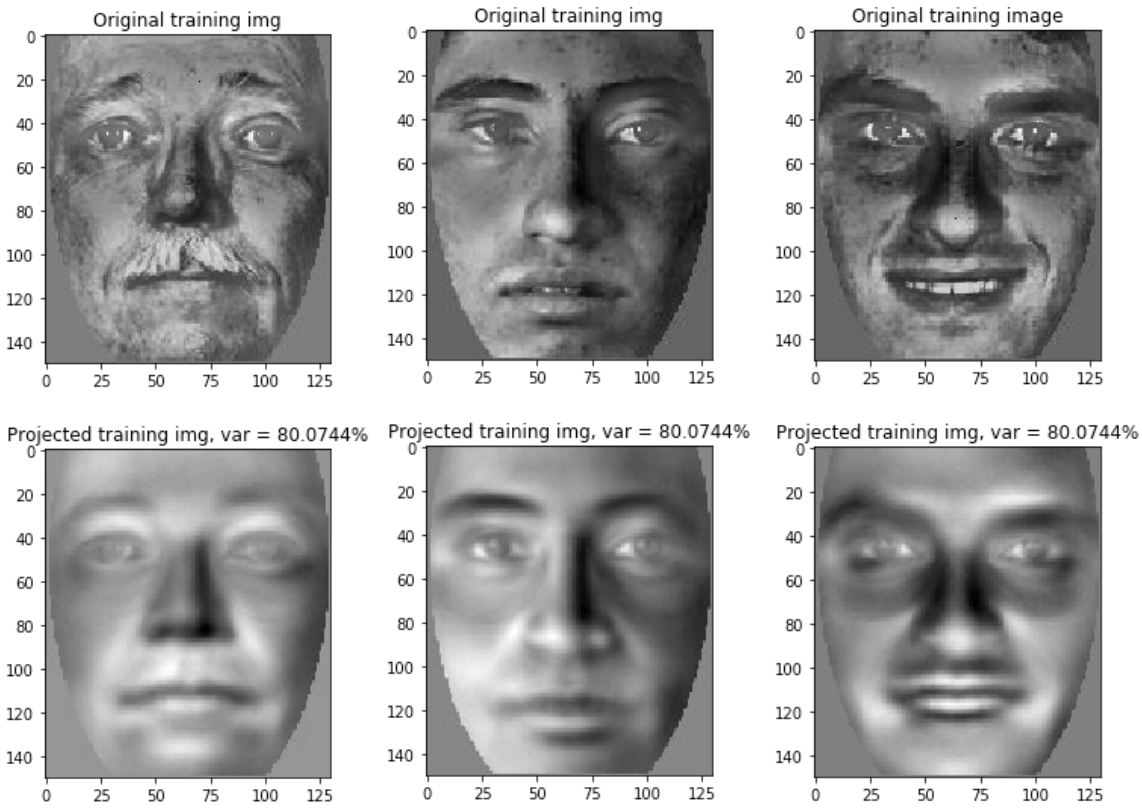


Images above are the eigenfaces of each eigenvector. All of the training images can be represented as a linear combination of these eigenvectors. Let's compare several projected images with the original.



There is a slight difference between the original image and the projected image in each pair. This is because we kept 90.0167% of the variance when choosing the best K number of eigenvectors. However, two images in each pair are very similar to each other. We want to see what happens if we choose a lower variance percentage. Let's pass in 80 instead of 90 this time into our get_best_eig_vecs().

The variance capture percentage is 80.0744%. This resulted in getting 76 eigenvalues and corresponding eigenvectors.



Since we chose to lose more information, it is obvious that the two images in each pair is not as similar as the prior examples.

[Section 3]

For face recognition, we implemented a function called find_closest_img(). It takes 3 arguments: a test image, a centered training image matrix, and the best eigen vectors. Both the test image and the centered training images were projected onto the same eigenspace. Then the Euclidean distances between the test image and each training images were computed in order to obtain the minimum value and its corresponding index. We were able to find the most similar image that had the minimum distance value. We tried several experiments with different variance percentages to see how accurately the program recognizes the correct match.

Variance	Accuracy(ratio)
95%	37/44
80%	35/44
60%	26/44
45%	13/44

The table above shows the accuracy of recognizing correct faces of different variance. It is clear that the smaller variance you choose, it is less likely for the program to find the best matching faces of test images. This shows you that in order to maintain a good accuracy, a reasonable percentage of variance should be kept.

[Conclusion]

The first thing I learned in this project is that I got more used to handling image data in python. Even though I struggled at first, since this was my first-time coding in python, but eventually it became much comfortable at the end. This will be a great help for me to have a faster progress in the next project. Additionally, the interesting fact about PCA is that it can help reducing dimensions of data while keeping as much variation as possible. These chosen K number of eigenvectors basically become a new orthonormal basis for the images, and those images can be represented as a linear combination of these basis vectors. Also, it was interesting that you can get eigenvectors of AA^T by multiplying A by eigenvectors of A^TA, which reduces a significant amount of work. During testing face recognition, it was very interesting that even if there are two images of a same person, of course with different looks, it successfully found its match with pretty high accuracy.