Project 1 Report: Eigenfaces for face recognition

CMSC 426 Spring 2021

Hongyu Tu 115323568

Logistics and bookkeeping

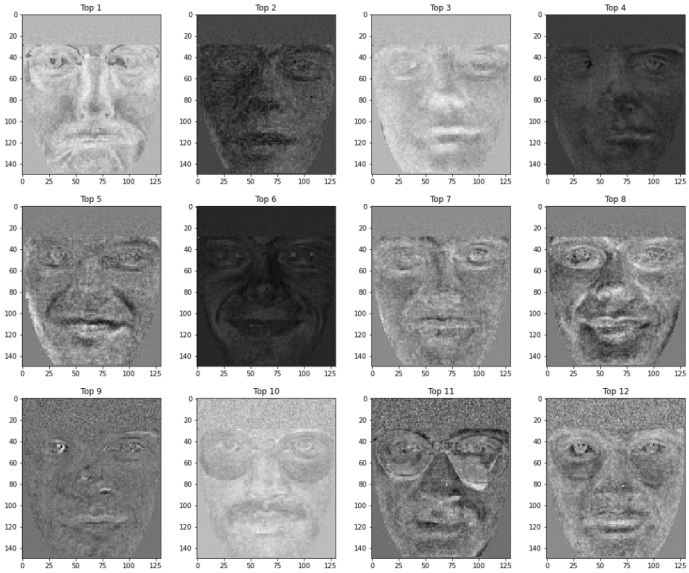
For project 1, I decided not to work in a group. All the codes and this report are done by myself.

Section 1: Implement PCA to represent faces onto a lower-dimensional space

For this part, I followed the instructions in the case study steps 1 to 7. On step 6 I got stuck for a little while since we are using A directly to find the eigenvectors of the covariance matrix. The covariance matrix can be A^TA or AA^T depending on how A was built (row by row vs. column by column). I end up solving that problem by looking at the dimensions and picked the way there are only 3772 singular values (total number of pictures). My A has each column representing an image, with 1 by 150 x 130. And after I did SVD on A, u gives me the first 3772 eigenvectors. I picked the quality to be 0.9 to start with. Since the first singular values carry out more weights, I used a for loop to find the k value that yields the proper portion of quality that gets to be kept. With quality set to 0.9, my k was found to be 2511. (More on picking optimal k in section 4)

Section 2: Display some of the top K eigenvectors also called the eigenfaces

From the previous part, I was able to find the top 2511 eigenvectors that corresponding to the largest 2511 eigenvalues. Each eigenvector has shape 1 by 19500. I picked the first 10 to display after reshaping the matrix into 150 by 130.



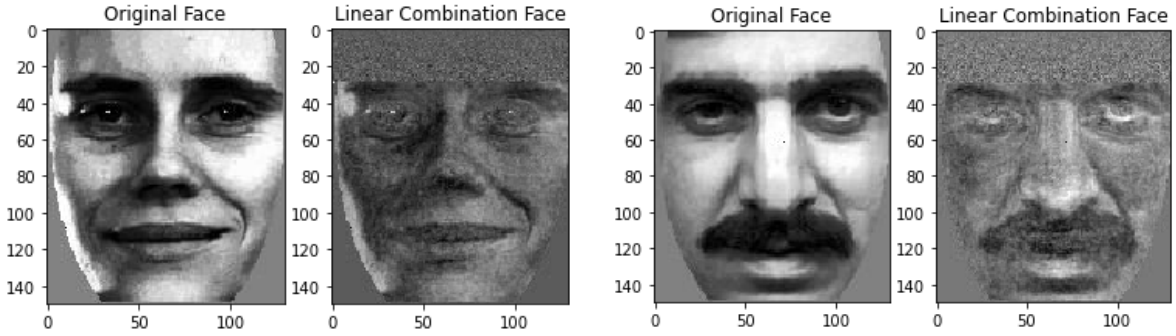
From the images I had above, I think the eigenfaces demonstrate the important characteristics that all the faces have.

Section 3: Represent face as a linear combination of the K eigenvectors

For this section, I followed the instructions in the case study again. From the formula:

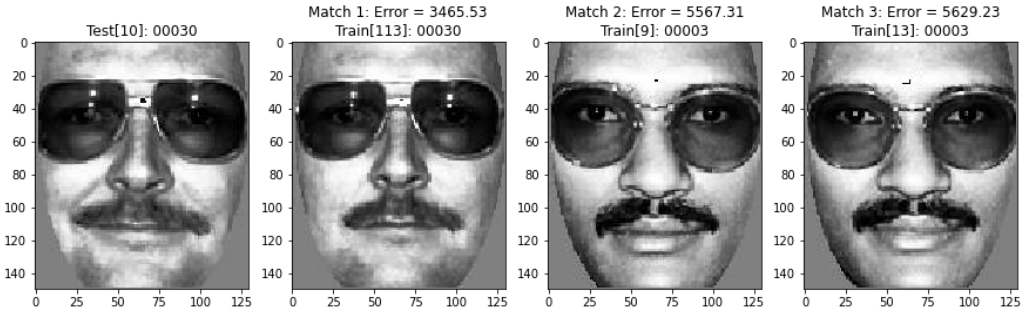
$$\hat{\Phi}_i - mean = \sum_{j=1}^{K} w_j u_j, \quad (w_j = u_j^T \Phi_i)$$

The weight of each eigenvector(eigenface) is calculated by the transpose of the eigenvector multiplied by the row in the mean of the image vector, which I believe is how we do the PCA and project our original vector to the lower dimension. With the weight, the original face can be recreated by summing up all weighted eigenfaces. From the faces, I'd say the recreation from combination worked pretty well. **The lower-dimensional representation of a face looks similar to the original face. One thing that I noticed is that the forehead part of all faces is sort of blurred out, filled with meaningless noise, which is the only dissimilar part comparing to the original image.** I suspect it might be that basically all faces have similar-looking foreheads and therefore not many characteristics are there and thus yield back an averaged-out forehead.



Section 4: Perform face recognition in the lower dimensional space

For this last part, the goal is to find the training image that is closest to our test image, after we do the PCA, and when each face was converted into a vector, the eigenvector that I found earlier can be used to project all the face onto the same lower-dimensional space. Therefore, I used a for-loop to store all images (both train and test) in the form of **Ω**, consist of $w_i = u_i^T \Phi$. And after that, I iterate through the test images again to find the distances of each test face to all the train images. To find our corresponding face, simply sorting the list will give us the best matches.



Here are the different k values that I have tried out:

Quality	# of e-vectors	Matched at 1	Matched at 2	Matched at 3	Matched within first 3
0.3	133	25/44 (57%)	2/44 (4.6%)	2/44 (4.6%)	29/44 (66%)
0.5	480	29/44 (66%)	1/44 (2.3%)	1/44 (2.3%)	31/44 (71%)
0.7	1197	31/44 (71%)	0	1/44 (2.3%)	32/44 (73%)
0.9	2511	31/44 (71%)	1/44 (2.3%)	0	32/44 (73%)
0.98	3427	31/44 (71%)	1/44 (2.3%)	0	32/44 (73%)

To find the optimal k, I did the test above by trying out different values. I end up picking the quality to be 0.7, and that results in a total of 1197 eigenvectors to be kept. By picking 0.7 and k = 1197, I was able to compress the original size by 68%, with minimized loss in terms of accuracy. **That's how I picked the lower dimensional basis vectors.** With over 3000 pictures and most of the pictures belongs to a different person, I think 70% accuracy is a fairly acceptable result, especially given the only thing we have done is PCA. One thing that was mention in the case study is that Mahalanobis distance works better, but I did the Euclidean distance, the same as what the study has done. That's one part I could have potentially done better.

Overall, I think I learned how powerful and simple PCA can be. Although I have learned PCA before by doing some dimension reduction for NLP and mostly using OpenCV or more complicated libraries like PyTorch. I have never thought about PCA and has such powerful usage in computer vision. I also looked at the face detection part in the case study, it's pretty impressive work. Besides that, I was a little confused by the singular value decomposition on A can be done instead of the covariance matrix to save time and space. Trying it myself helped me figure it out. It's a nice trick and I'll remember to do that next time when there's a similar problem.