HW9 WriteUp

yc704 Yuan-Yao Chang

I created three python file for the eigenfaces, test 1 and test 2. I will list the implementation method below:

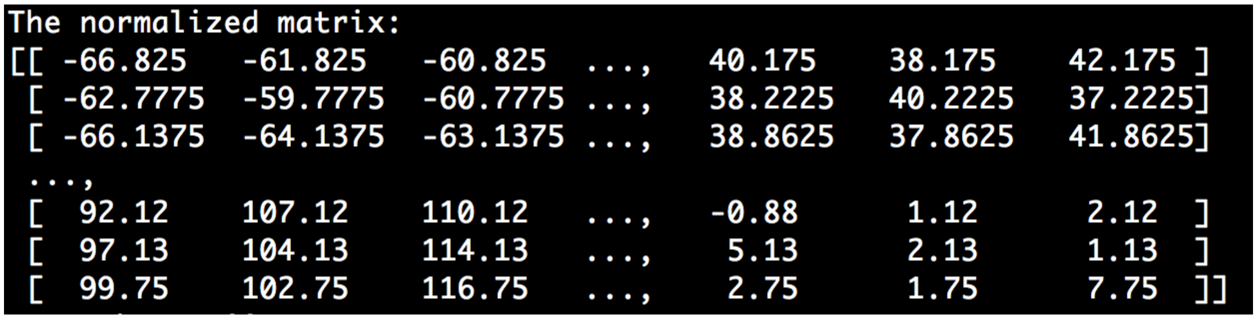
A. eigenfaces.py

For this program, simply run it and it will print out the method used for the matrix multiplication(Turk or Pentland), print out ATA(if normalized_ImageMatrix.shape[0]>normalized_ImageMatrix.shape[1]) or AAT(if normalized_ImageMatrix.shape[1]>normalized_ImageMatrix.shape[0]). Then it will print the normalized matrix, following the prompt to enter the k for the eigenvalue in order to generate the eigenfaces. After entering the k value, the program will print out the eigenfaces matrix; the program will also store a text file which contains the normalized matrix and the k eigenvector matrix, depends on the k user choose.

Steps:

1) Read the images through the entire database, by create a list that contains all the paths.
2) Transform each image face into a column vector by which that is 10304 by 400 matrix, where 10304 is a long, by using numpy(np.ravel) and PIL(used the "L" option to open) to flattened vectors of a total 400 images.
3) Then we use the face matrix to calculate the mean of each row, also generate an meanface image shown below.

4) For normalizing the face matrix, once again I used the facematrix generated above to normalize the matrix with the function of taking each element in the row and subtract the mean of each rows. The matrix is shown below:



```
The normalized matrix:
[[ -66.825   -61.825   -60.825  ...,   40.175    38.175    42.175 ]
 [ -62.7775  -59.7775  -60.7775 ...,   38.2225   40.2225   37.2225]
 [ -66.1375  -64.1375  -63.1375 ...,   38.8625   37.8625   41.8625]
 ...,
 [  92.12    107.12    110.12   ...,   -0.88      1.12      2.12  ]
 [  97.13    104.13    114.13   ...,    5.13      2.13      1.13  ]
 [  99.75    102.75    116.75   ...,    2.75      1.75      7.75  ]]
```

Following step is to reducing the dimension by producing covariance matrix which is 400 by 400 thru Turk and Pentland method. Show as equations:

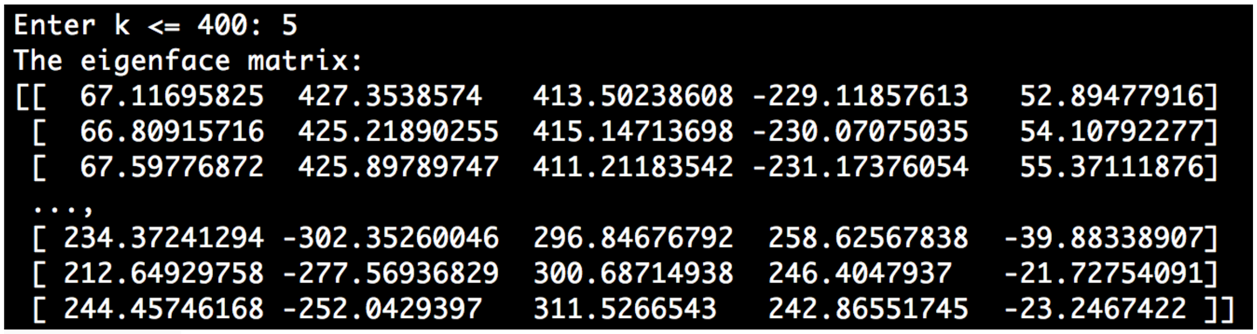
$$A^T A v_i = u_i v_i \text{ where } v_i \text{ is an eigenvector}$$

Then, multiplying the both sides by the normalized matrix A,

$$AA^T Av_i = Au_i v_i \rightarrow (AA^T)Av_i = u(Av_i) \rightarrow Av_i = u(Av_i)$$

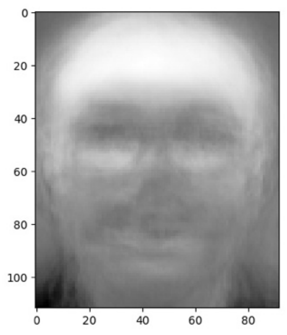
Proves that Turk and Pentland method can be used for reducing dimension of the matrix.

5) Let the user to input the k for the top 5 eigen faces.
6) Finding eigenvalues and eigenvectors by using numpy.linalg.eig of the matrix generate above.
7) sorting the eigenvalues using argsort to get index of the values in increasing order than flip the array in order to make the top value easier to access to find the top k eigenvectors.
8) Lastly, by multiplying the normalized matrix and the to generate the top k eigenfaces. Then output as a jpg.

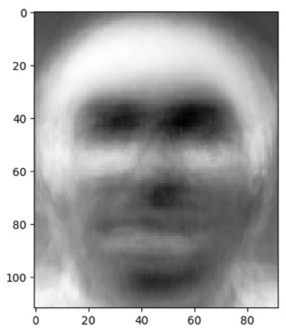


```
Enter k <= 400: 5
The eigenface matrix:
[[  67.11695825  427.3538574   413.50238608 -229.11857613   52.89477916]
 [  66.80915716  425.21890255  415.14713698 -230.07075035   54.10792277]
 [  67.59776872  425.89789747  411.21183542 -231.17376054   55.37111876]
 ...,
 [ 234.37241294 -302.35260046  296.84676792  258.62567838  -39.88338907]
 [ 212.64929758 -277.56936829  300.68714938  246.4047937   -21.72754091]
 [ 244.45746168 -252.0429397   311.5266543   242.86551745  -23.2467422 ]]
```

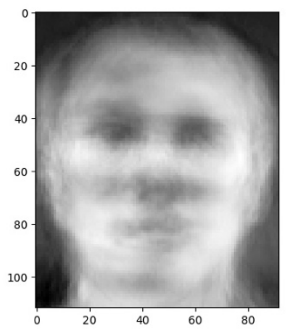
The eigenfaces:



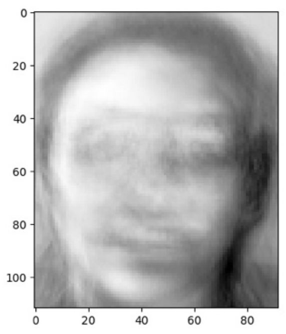
0



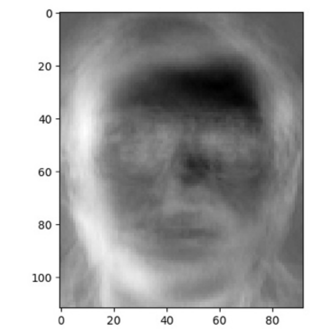
1



2



3


4

B. test1.py and test2.py

1) For the test 1 and 2 the process is the same, the difference and extra process is have to input an TEST_Image and convert into matrix, then normalizing the testimage_matrix.
2) After, also generate the eigenface matrix which is (10304, k) like above. By multiplying the eigenface matrix and the test image's normalized matrix, a projected testimage vector is generated(k,1). Then we do the same thing for the normalized matrix of the entire database; which we have all 400 of the projected image vectors(k,1). Lastly, We compare the Euclidean distance of every projected image vector to the projected testimage vector, and we find the minimum distance of all.
3) The smallest distance means that is closest to the testimage. For the test 1 it is 100% will be the same image which dis=0; but the test2 will not be 0, because the testimage has been pulled out(not in the database), and the image been replaced by some other image.