ITE4052 Computer Vision
Final Term Project - PCA
2016026080 한다진

 # Q1 – Calculate Eigenfaces

Eigenfaces are 2-dimentional version of eigenvectors. With these eigenfaces, we can express image A as a point on specific coordinate system. In this coordinate system, each eigenfaces are the coordinate axes. In this project, the Goal is to transform the coordinate system from with higher number of coordinate axes to smaller number of coordinate axes. Of course, there are information loss during coordinate system conversion. So the Key Idea is to loss small information. So We Calculate the eigenvectors, then use eigenvectors with higher eigenvalue as a new coordinate axes. Ths decrease the loss of information during conversion. The original Image is 28x28 so it is 784 dimension point. The converted image is expressed with 300 dimension point.

First normalize x_train dataset. “phi = x_train – np.mean(x_train)”
Second, Create Covariance C with np.cov function.
Then I can calculate eigenvalues and eigenvectors.
So <Q1 Top 40 Eigenfaces> is the visualization of 40 eigenvectors with high eigen value.

 # Q2 – Image Approximation

With eigenvector and eigenvalues, we can express image with 300 dimensional point. Now calculate K with two condition.

First, “Smaller K, more meaningful transformation”
Second “Bigger K, more accurate expression”

So I use eigenvalue to which eigenvector is better for coordinate axes. I Select K eigenvectors with highest eigenvalue. The sum of eigenvalues of selected eigenvectors should be bigger than the 85% of total eigenvalue sum. So K is calculated as 45. Then Calculate the MSE. When I increase K, MSE decreses.

 # Q3 – Fast approach

In Q1, I make C with 300 x 784 matrix(A). 300 is the number of images in dataset. 784 is the number of pixels in single image. ATA is 784 x 784 matrix. This time, I make C with AAT. So the shape is 300x300. This is faster than the Q1 algorithm. Often the number of dataset is smaller than the total pixel in single image because nowadays many study use the high resolution image as a dataset. So Using AAT as a C matrix and then calculate eigenvalue and eigenvectors in 300x300 dimension. Then Select K eigenvectors, so selected eigenvectos are Kx300 dimension. Now convert Kx300 dimension to Kx784 dimension using 300x784 dataset matrix. Then I compare the time, Q3 algorithm is 4 times faster than Q1 algorithm.

Other explanation is commented on my pca.ipynb jupyter file.

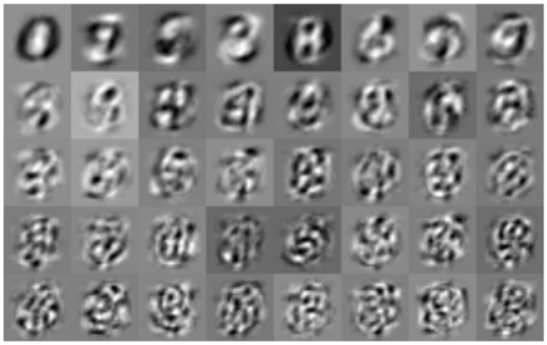
Results

```
1 (0.073908588995154l5+0j)
2 (0.06852356306909377+0j)
3 (0.064002194325848370j)
4 (0.06385102221445979+0j)
5 (0.062345657155014490j)
6 (0.061083266820642510j)
7 (0.050439008079551340j)
8 (0.049655391781908295+0j)
9 (0.048701959703484024+0j)
10 (0.048588307331533890j)
11 (0.046530122487157020j)
12 (0.042121648030088770j)
13 (0.042041826353035320j)
14 (0.039076867028318030j)
15 (0.039042381507703616+0j)
16 (0.038547916192862190j)
17 (0.036480189142895476+0j)
18 (0.036123071867007285+0j)
19 (0.035844325152573870j)
20 (0.034746449248293330j)
21 (0.034704096383500510j)
22 (0.034210433277678580j)
23 (0.033974927446240775+0j)
24 (0.033213502743365944+0j)
25 (0.033207825034119920j)
26 (0.033203074169509760j)
27 (0.033158683151442840j)
28 (0.033085024336807864+0j)
29 (0.033083636642784060j)
30 (0.032709134786096430j)
31 (0.032463581356808900j)
32 (0.031976339040626770j)
33 (0.031885100796602170j)
34 (0.031193955651974450j)
35 (0.030503220053249475+0j)
36 (0.030502500741696782+0j)
37 (0.030502379148337862+0j)
38 (0.03027020079252060j)
39 (0.029689636925812130j)
40 (0.029388613488592220j)
41 (0.029106011634214890j)
42 (0.029054486657108917+0j)
43 (0.028975459475363334+0j)
44 (0.028937141178430322+0j)
45 (0.028911366317372223+0j)
```

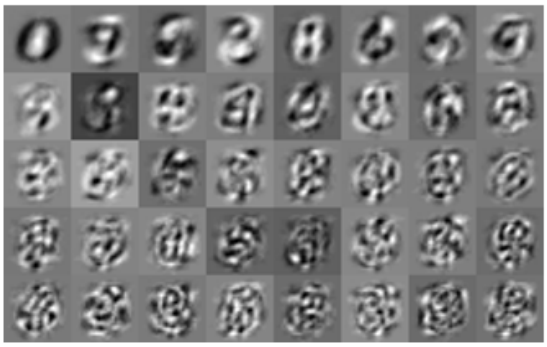
<Q1 MSE>

```
1 0.07363495994591424
2 0.06882006325568106
3 0.06401792979507115
4 0.064013272918952
5 0.06359953532480139
6 0.06255306259692563
7 0.051479054967131316
8 0.04914419051098041
9 0.0489827010541285
10 0.04892439752259409
11 0.04709908456518811
12 0.04275725929977415
13 0.042789200018245224
14 0.040148234482766680
15 0.040121294487915650
16 0.039620864653097165
17 0.038033663097374396
18 0.03749195426377777
19 0.036881222115609305
20 0.03620561044150005
21 0.03627256893063353
22 0.03573425838601146
23 0.03566323609193089
24 0.03484225468251479
25 0.03492112921716302
26 0.03492824409576505
27 0.0348807006021145
28 0.0348696092195741
29 0.034954415447032414
30 0.03468412652396107
31 0.03446749421193844
32 0.034013486917932996
33 0.03394947776565637
34 0.033244675145592685
35 0.03258358893453086
36 0.03263274860792406
37 0.03265080291651801
38 0.03260378653414392
39 0.03198902023319009
40 0.031804111541968216
41 0.0315678307099898
42 0.031542686560984586
43 0.031464251295905576
44 0.03152509838016462
45 0.031527725712152004
46 0.0314228499114197
```

<Q3 MSE>



< Q1 Top 40 Eigenfaces>



<Q3 Top 40 Eigenfaces>

```
### step5: choose K
ratio = 0
preserved_info_ratio = []
for eigenvalue in eigenvalues:
    ratio += eigenvalue
    preserved_info_ratio.append(ratio)
preserved_info_ratio /= np.sum(eigenvalues)

K = np.sum(preserved_info_ratio < 0.85)
print("K : ",K)
```

<Q2 choose K>

```
Shape of eigen vectors =  (784, 784)
K :  45
0.2568349838256836
Shape of eigen vectors =  (300, 784)
K :  46
0.062207937240600586
```

< Q4 Time comparison>

<Q2 approximation result>