수치해석 Project#1

EigenFace를 이용한 Face Recognition

Python(opencv) 사용

컴퓨터소프트웨어학부 2015005187 최철훈

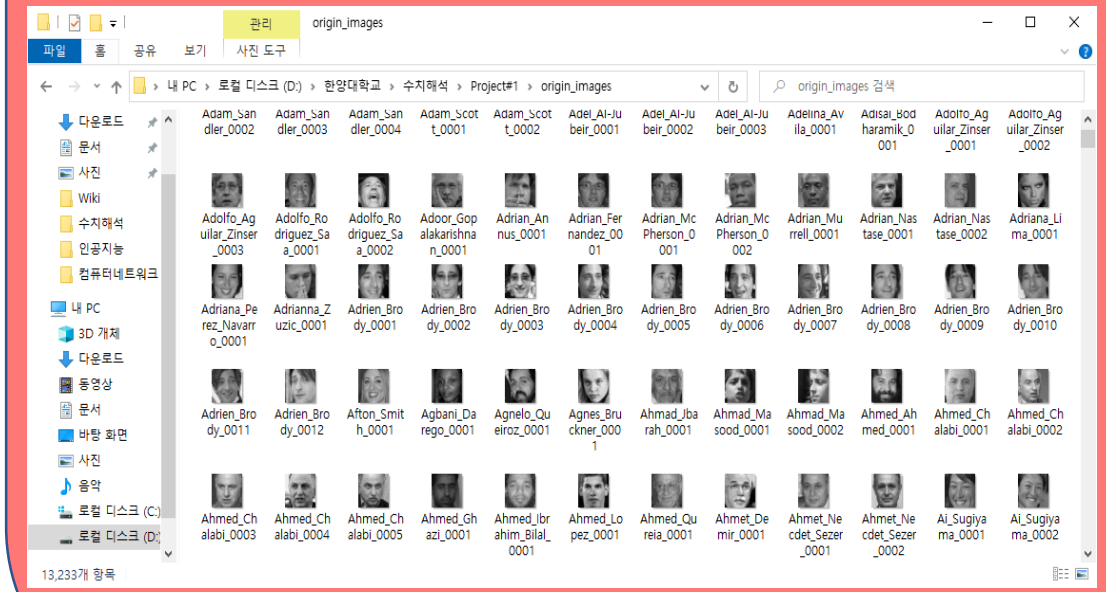
목차

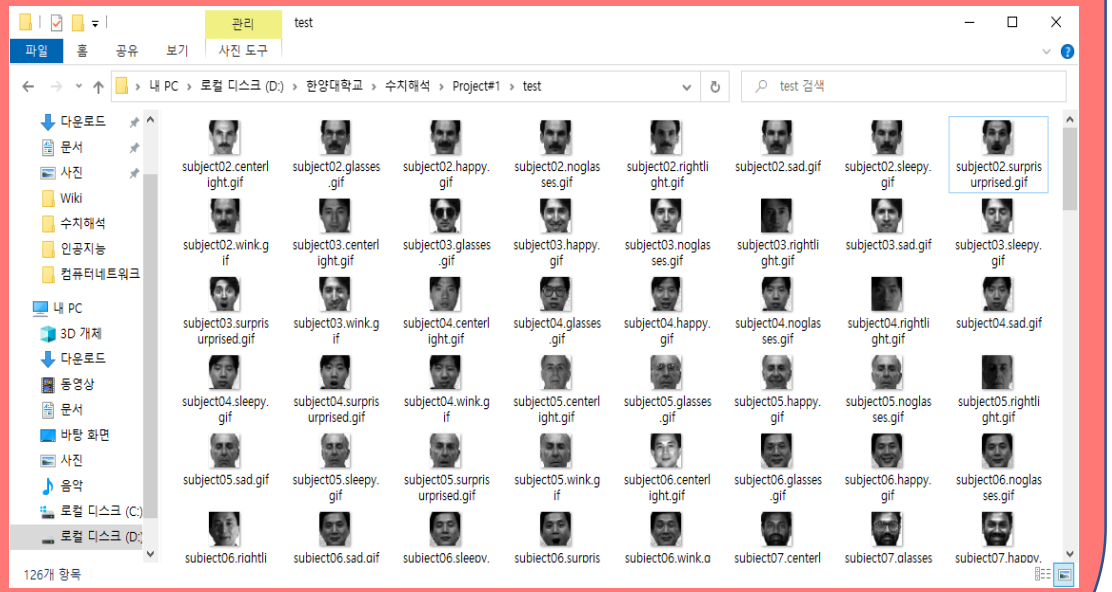
이미지 확보 및 가공 이미지 확보

같이 수업을 수강하는 친구와 함께 Project에 필요한 이미지를 구글링하여 찾아냈다.

총 13,233개의 이미지를 확보하였다.

테스트로 사용할 이미지는 14명의 사람과 각각의 서로 다른 얼굴 이미지 9장 총 126개의 이미지를 확보하였다.





이미지 확보 및 가공 이미지 가공

opencv를 활용하여 확보한 이미지의 사이즈를 조절하였다. 여러 사이즈를 시도해보았으나 사이즈를 키울수록 눈에 가시적으로 보이기는 잘 보이지만 여러 사이즈의 사진으로 PCA를 돌려본 결과, 계산량이 많아 작업이 오래걸려 교수님게서 제시해주신 32 * 32로 사이즈로 설정하였다.

이미지 ID	
사진 크기	32 x 32
너비	32픽셀
높이	32픽셀
수평 해상도	96 DPI
수직 해상도	96 DPI
비트 수준	8

```
imagePath = os.path.join(path, filePath)
src = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
im = cv2.resize(src, dsize=(32, 32), interpolation=cv2.INTER_AREA)
cv2.imwrite("test/" + filePath, im)
```

13,233개의 이미지 중 같은 사람의 다른 사진인 경우가 많았다. 데이터의 개수가 많을수록 연산 과정이 오래 걸리므로 데이터의 개수를 줄이고 전부 다른 사람의 사진으로 만들기 위하여 중복된 사람은 제거하여 총 5,749개의 이미지만을 남겼다.

PCA과정과 EigenFace 선택 PCA과정

1. 폴더 내 전체 Training이미지를 opencv를 활용하여 각각의 픽셀의 값을 나타낸 행렬의 형태로 불러온다.
```
im = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
```

2. 32*32사이즈였으므로 32*32 행렬로 불러온다. 이를 flatten()을 활용하여 1024*1벡터로 만든다.
```
image = images[i].flatten()
```

3. PCA에 필요한 데이터 행렬 A를 만들기 위해 2에서 만든 이미지들을 하나의 행렬로 만든다.
```
matrix_a[i,:] = image
```

4. opencv에 내장된 PCACompute함수를 이용하여 전체 데이터의 평균과 원하는 개수의 EigenVector들을 얻는다.
```
mean, eigenVectors = cv2.PCACompute(data, mean=None, maxComponents=NUM_EIGEN_FACES)
```

PCA과정과 EigenFace 선택 EigenFace 선택

1. 앞서 구한 EigenVector중에서 Principal Component를 선택하기 위해 데이터행렬 A를 SVD 하여 각각의 Singular Value를 얻었다.

2. 1에서 얻은 Singular Value중 10,000이상인 개수가 16개가 있었고, 이를 Principal Component로 설정하였다.

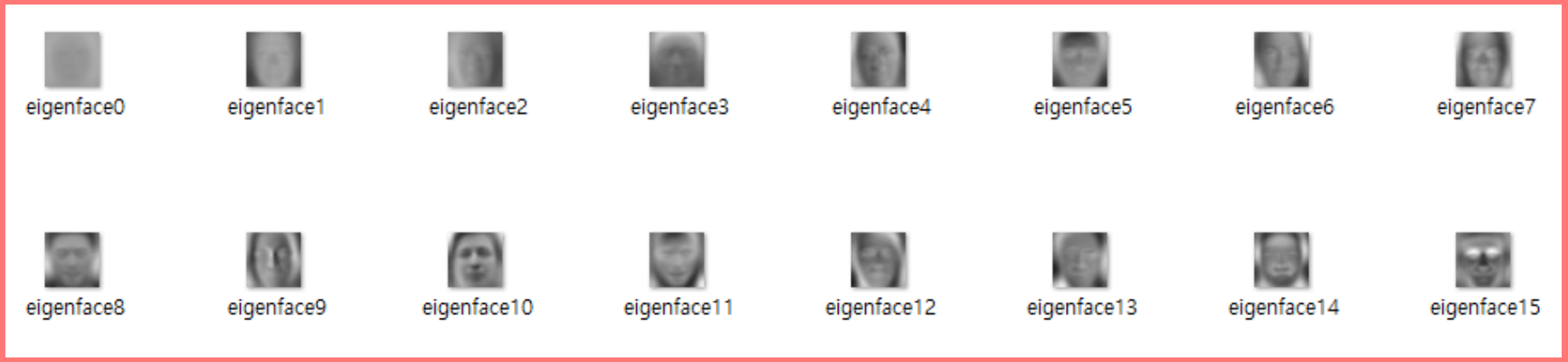
```
u, s, vt = np.linalg.svd(a_matrix)
s = s.flatten()
j = 1
for i in s:
    if(i > 10000):
        print("Singular Value : " + str(i))
        print("Number of Singular Value : " + str(j))
        j = j + 1
```

```
Singular Value : 10337.741
Number of Singular Value : 16
```

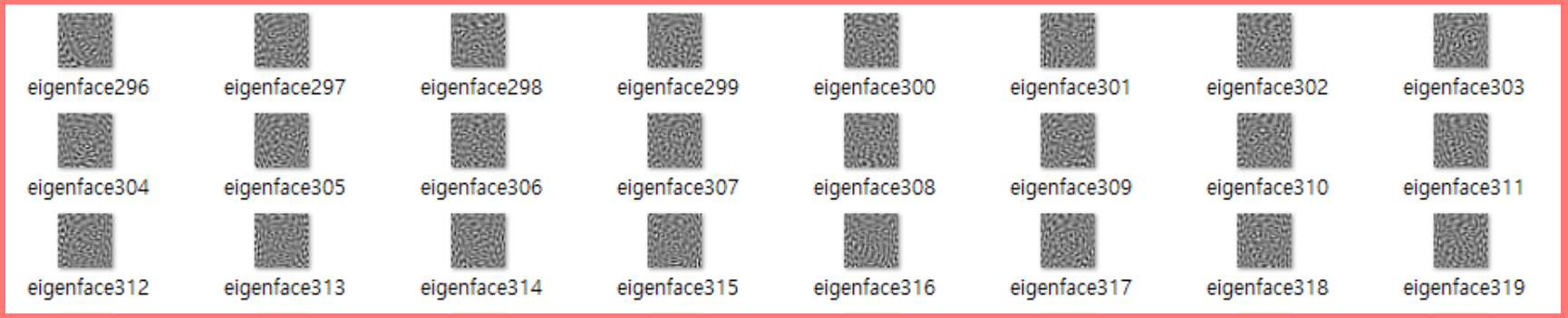
PCA과정과 EigenFace 선택 EigenFace 선택

3. 16개의 EigenVector를 EigenFace로 변환하여 이미지화하였을 때, Singular Value가 낮은 EigenVector들과는 확연히 다른 차이를 확인할 수 있었다.

- 선택한 EigenFace



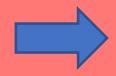
- 영향력이 별로 없는 EigenFace



Test 이미지의 c_k값 구하기 계산과정

1. 테스트 이미지들을 flatten()을 활용하여 1024*1의 벡터로 만들어 하나의 행렬로 저장한다. 이 때, 각각의 이미지들은 원점을 포함하지 않아 벡터공간을 이루지 않으므로 각각의 평균을 구하고 빼서 원점을 포함하는 벡터공간으로 만든다.

```
im = cv2.imread(imagePath, cv2.IMREAD_GRAYSCALE)
im = im.flatten()
mean = im.mean()
im = im - mean
```



```
images[i,:] = im
i = i + 1
```

2. c_k는 원래 이미지에서 평균을 뺀것과 해당하는 EigenFace를 내적하면 결과를 얻을 수 있다. 1에서 얻은 이미지의 행렬성분과 PCACompute()로 얻은 EigenFace들을 내적하여 c_k를 구한다. 다른 사람의 이미지임을 알리기 위해 각각의 사람은 9장의 다른 이미지를 갖고 있으므로 9장마다 구분지었다.

```
ck = np.zeros((126, NUM_EIGEN_FACES), dtype=np.float32)
test_images = readTestImages("test")
for i in range(0, 126):
    for j in range(0, NUM_EIGEN_FACES):
        ck[i][j] = np.dot(test_images[i], eigenFaces[j])
count = 1
for i in range(0, 126):
    print(ck[i])
    if(count % 9 == 0):
        print("----------------------")
    count = count + 1
```

Test 이미지의 c_k값 구하기 결과예시

원본 이미지

c_k값

```
[   41.320774  -735.7529    -150.4732    -805.1382    -311.78842
 -1175.991      224.79594    308.99213    708.62463     76.038536
   260.78607   -379.08752   -125.49937   -137.66254    218.86322
  -254.76396 ]
```

```
[-110.68897  -305.19427   656.1328   -907.3788    151.07533  -976.4293
  509.19412   -60.39447   148.72244   148.93253   -53.67522    23.401722
 -156.29367   -34.55419  -155.94101  -103.24802 ]
```

```
[  -83.59227   -465.922     -64.98332  -1117.0896    -327.37952
  -970.22205    58.11895   412.71817    347.96716     7.1860504
    34.008347  180.76901   158.53499   -167.4831    -67.72662
  -129.17632  ]
```

원래 이미지 복원해보기 Linear Combination

앞서 선택한 EigenFace 16개로 원래의 이미지를 복원해본다.

1. 앞서 구한 c_k와 k번째 EigenFace를 곱한 것들을 더하여 Linear Combination한다.
2. 원래 이미지의 평균을 Linear Combination한 이미지에 더한다.
3. 이미지화한다.

```
means = ImageMean("test")
for i in range(0, 126):
    recovery = np.zeros((32 * 32,), dtype=np.float32)
    for j in range(0, NUM_EIGEN_FACES):
        multiple = ck[i][j] * eigenFaces[j]
        recovery = recovery + multiple
    recovery = recovery + means[i]
    output = recovery.reshape(sz)
    cv2.imwrite("recovery/recovery_" + str(i + 1) + ".jpg", output)
```

- 결과예시  <- 원본 이미지

원래 이미지 복원해보기 All EigenFace

앞서 보았듯이 EigenFace 16개로는 비슷하긴 하지만 완전히 똑같다고 육안으로 판별하기는 어려운 수준이다. 그래서 1024개의 모든 EigenFace로 Linear Combination하여 복원해보았다. 당연한 결과지만 원본과 육안으로 식별하기에 동일한 결과물이 나왔다. 이로 인해 EigenFace의 개수가 많아질수록 원본과 똑같은 이미지를 얻는 다는 것을 알 수 있다.

- 결과예시

원본 이미지

복원한 이미지

원래 이미지 복원해보기 c_k를 정수로 하기

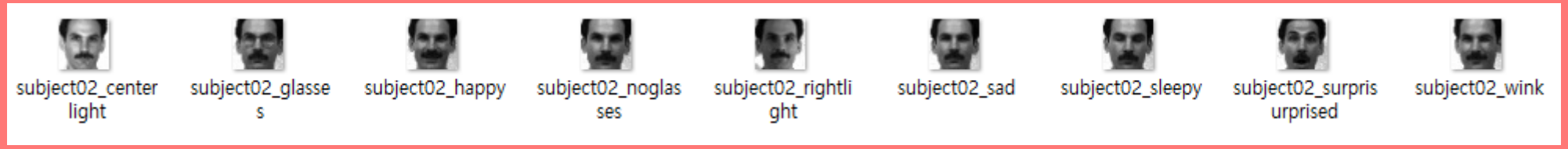
처음에는 c_k 를 판별하기 쉽게 하기 위해 정수로 설정하고 원래 이미지를 복원했었다. 하지만 앞서 복원한 이미지에 훨씬 못미치는 결과가 나타났다. 이로 인해 0이하의 소수점 자리지만 버린다면 연산 결과에는 크게 영향을 미친다는 것을 알 수 있다. 교수님께서 강의에서 말씀하신 Precision의 오류가 결과에 영향을 미친다는 것을 알 수 있었다.

- 결과예시 실수로 복원한 이미지 정수로 복원한 이미지






c_k값 비교 같은 사람, 다른 사진끼리의 비교




```
[   41.3208  -735.7529  -150.4732  -805.1382  -311.7884 -1175.991    224.7959  308.9921  708.6246   76.0385  260.7861  -379.0875  -125.4994  -137.6625  218.8632  -254.764 ]
[   92.6809  -883.178    654.9197  -332.3031   771.5175 -1252.5326   101.2417  383.8459  294.5484  166.4441  558.915    -80.5055   190.9084  -258.2345  207.4628  -240.1309]
[   95.6483  -883.3784   441.9628  -401.6793   491.5426 -1371.3499   108.9272  320.278   408.4369  173.7807  458.9501  -176.7124   -4.7322  -249.9845   53.5005  -275.1119]
[   50.5301  -501.8955   253.9373  -305.3358   193.0868 -1647.9939   175.1952   61.8023  285.3084   79.5481  234.5789   49.106   -133.814   -262.0563   36.571   -335.9267]
[  163.034  -1000.3743 -1050.7229  -778.3414    85.2772  -976.1514   246.2326  157.1737  282.9595 -334.8091  274.1398  -328.4744  -220.5437   44.0175  -122.4351  -127.2315]
[   64.9637  -761.8469   529.5779  -393.966    563.134  -1410.0903   166.4077  333.9767  365.6844  182.3644  575.7018   -57.9657   34.1217  -201.444   142.5627  -261.9993]
[   72.0765  -640.2489   295.0836  -324.9737   270.2822 -1596.9023   172.7902  110.0455  358.2988   99.4095  276.1875   -17.4418  -63.1519  -271.99     49.813   -301.413 ]
[  112.5743 -1142.0413    90.224   -555.8997    51.0332 -1013.8657   368.1688  512.5769  418.2113   57.9735  678.5316  -252.0489  -322.7261   -96.5532  261.4619  -166.2825]
[   95.3517  -924.1326   793.2505  -308.8723   755.8936 -1226.9249   127.1005  335.2348  383.4756  151.8181  539.3788   -16.364   336.4204  -191.8833  190.6214  -263.7462]
```

같은 사람의 각기 다른 사진 9장에 대한 c_k값에 대한 비교 예시이다. 각각의 열끼리 비교해보았을때, 1, 2번째 열처럼 비슷한 값을 보이는 열이 있는 반면에 3, 13번째 열처럼 일정범위를 벗어나는 값이 나타나거나 들쑥날쑥한 것을 확인할 수 있다. 다른 사람도 마찬가지의 결과를 보였다.

c_k 값 비교 다른 사람끼리의 비교

```
[   41.3208  -735.7529  -150.4732  -805.1382  -311.7884 -1175.991    224.7959   308.9921   708.6246    76.0385   260.7861  -379.0875  -125.4994  -137.6625   218.8632  -254.764 ]
[   92.6809  -883.178    654.9197  -332.3031   771.5175 -1252.5326   101.2417   383.8459   294.5484   166.4441   558.915    -80.5055   190.9084  -258.2345   207.4628  -240.1309]
[   95.6483  -883.3784   441.9628  -401.6793   491.5426 -1371.3499   108.9272   320.278    408.4369   173.7807   458.9501  -176.7124    -4.7322  -249.9845    53.5005  -275.1119]
[   50.5301  -501.8955   253.9373  -305.3358   193.0868 -1647.9939   175.1952    61.8023   285.3084    79.5481   234.5789    49.106   -133.814   -262.0563    36.571   -335.9267]
[  163.034  -1000.3743 -1050.7229  -778.3414    85.2772  -976.1514   246.2326   157.1737   282.9595  -334.8091   274.1398  -328.4744  -220.5437    44.0175  -122.4351  -127.2315]
[   64.9637  -761.8469   529.5779  -393.966    563.134   -1410.0903   166.4077   333.9767   365.6844   182.3644   575.7018   -57.9657    34.1217  -201.444    142.5627  -261.9993]
[   72.0765  -640.2489   295.0836  -324.9737   270.2822 -1596.9023   172.7902   110.0455   358.2988    99.4095   276.1875   -17.4418   -63.1519  -271.99      49.813   -301.413 ]
[  112.5743 -1142.0413    90.224   -555.8997    51.0332 -1013.8657   368.1688   512.5769   418.2113    57.9735   678.5316  -252.0489  -322.7261   -96.5532   261.4619  -166.2825]
[   95.3517  -924.1326   793.2505  -308.8723   755.8936 -1226.9249   127.1005   335.2348   383.4756   151.8181   539.3788   -16.364    336.4204  -191.8833   190.6214  -263.7462]
```

```
[   65.4715  -846.2003  -434.2378  -787.2926  -157.8937 -1178.3612  -178.7477   331.3075   174.4114    37.8617  -226.3181   128.861   -192.9077  -138.64     13.6475   -50.5861]
[    3.4725  -662.559     48.6287  -756.1008    41.5118 -1438.804     46.4236    59.6693   299.1167   -42.8467   167.88     354.6047  -162.6194  -135.5984   169.3404  -181.5706]
[   -7.5788  -757.8049   165.9855  -849.8706   192.1481 -1330.219    179.4676   125.6942   257.7405   -57.6929   199.7283   349.6719     2.3839  -153.134    153.6086  -181.5056]
[    2.3456  -861.9829   174.0662  -955.7893   118.5038 -1325.5531   182.8864    21.1279   317.9448   -75.8871   392.943    291.2399    -3.4942   -72.8148   154.3503  -321.4298]
[   15.7005  -847.1649  -188.8083 -1008.3315   -99.5577 -1278.1685   -44.8499   133.2028   335.1426   -87.7259   144.8424   289.3825  -304.9489  -163.6158   196.1974   -96.8826]
[   14.2171  -654.2738    89.21    -623.7736    86.2556 -1495.9585    99.1596    76.2939   262.7351   -14.1163    89.9181   349.3678   -75.5535  -110.1455   157.806   -231.3508]
[   49.2315  -867.332     40.3797  -730.7146    77.067   -1455.6417   90.3666    26.2537   329.4567   -30.1234   102.7858   250.6936   -33.32    -72.931     69.8651  -138.1776]
[   -1.2507  -977.7523   112.2965 -1060.6724    83.3958 -1132.787     91.8717   115.6512   245.4329   -18.1077   193.4477   346.7078   -19.6735  -62.9585    59.5718   -65.0525]
[   36.2636  -757.6664    46.9063  -663.6824   114.0035 -1432.2532     4.1438    91.1133   297.9495   -53.9766    82.6464   324.7727  -155.8748  -268.8533   118.2425  -184.1127]
```

```
[  -43.9338  -656.4193  -324.2878 -1283.4324    63.3118 -1034.7058   100.4054   355.0637   328.9313    81.6237   -92.8417    74.131    -1.6695  -154.3308  -118.3849   -86.3061]
[  -34.4478  -803.4783   105.2311 -1138.2869   390.1499  -977.0109   239.2083   223.2483   188.7392    25.6934   219.8269   364.6442    94.1395  -290.6017    13.3544  -103.7853]
[ -123.1884  -476.893     28.4823 -1267.0476   242.4085 -1010.5936   281.7852   165.8566   209.3726    21.6172   341.4257   546.3828   -88.8579  -215.8546   -88.8586  -219.6235]
[  -73.4627  -772.3533   273.3148 -1215.3914   572.1681  -795.2792   277.4028   328.3189   294.8296    59.2932   473.6688   463.824    108.731   -197.7991    67.22    -168.5157]
[   65.1644  -319.2784 -1129.5176  -563.0649    78.5845  -607.4739  -200.0808     7.5178    -9.1104  -189.279    -80.6762   215.3511   198.8332   -65.8723   -94.3143  -143.3905]
[  -91.1885  -529.7997    13.1488 -1167.0195   223.2086 -1065.6797   287.6245    69.5604   238.3815    10.1656   231.9843   576.2878   -96.8497  -174.3696    59.1779  -156.6342]
[ -136.5955  -408.994     123.8541 -1192.5347   375.0256  -874.8735   362.3954   169.6806   119.5077    85.2407   347.5808   574.5413  -116.4726  -137.3759    55.3594  -136.8928]
[  -92.5718  -596.5552    72.8774 -1156.1189   348.9058  -962.3719   259.2277   299.4902   135.1888    55.4146   522.1281   555.3757  -199.4101  -147.6372    52.7312   -31.5308]
[  -85.2251  -628.5325   306.0194 -1092.7476   651.0393  -679.1899   360.3656   259.356    195.4741   140.709    430.7586   508.9172    38.8959  -165.8004    71.8339  -166.757 ]
```

c_k 값 비교 다른 사람끼리의 비교

```
[   37.7545  -673.2709  -198.8069  -543.8369  -191.2318 -1096.3347   665.8331   602.2695   502.341    14.8289   623.0293  -163.0594  -203.836   -35.2782    8.6046  -141.6699]
[  105.7999  -712.7767   202.8787   -86.4512   134.1862 -1562.0586   557.4863   424.563    547.8695  -32.8997   614.1395  -233.0336  -108.6437  -41.2798  -189.3451  -299.5906]
[   67.1974  -554.6424   180.1036   -99.1874   145.452  -1351.1813   749.8033   433.3866   479.4809    6.2659   850.176   -213.6802  -239.3403   22.3862  -215.6718  -336.1949]
[   81.393   -671.2877   396.2714   -99.8092   340.9983 -1138.2771   771.6743   480.9816   672.5532   83.3965   931.0955  -355.1742  -200.2342  123.8371   -80.3835  -251.1577]
[  105.3228  -378.6184 -1115.2325  -437.8067   391.1711  -733.7572    -4.7026     5.836    281.0999  -308.9755   388.8804  -124.1836   -56.1347  132.168   -305.9758  -110.3736]
[   60.3631  -255.9766   391.9332    17.6751   365.4448 -1392.4746   473.4137  -265.3387   601.8801   153.8138   158.4233   -95.9872  -154.0999  135.0992  -192.6911  -457.4247]
[   65.0942  -467.6815   631.492    -44.3567   660.1561 -1148.3163   572.806     55.8549   735.3811   224.6342   526.8807  -415.8861   -24.709   122.8671  -104.9159  -296.9942]
[  133.498   -758.4549  -178.9623     9.9669  -302.8886 -1206.5996   968.548    396.2718   499.8093    3.6837   829.1411  -297.5166  -371.1094  138.581    76.8083  -211.6502]
[   65.9041 -725.1012  879.2976 -219.7015  930.8114 -623.598   439.6256  321.5237  627.0405   15.4564  960.5853 -375.7036  -51.6238  124.2973  167.1512 -168.7035]
```

```
[ -59.325  -265.9139  278.219  -613.095   320.6092 -981.4248  400.2088  620.7311  489.5728  181.2234  269.4367  -71.0307   47.4866 -100.8768 -332.1653 -137.0942]
[ 126.9124 -972.808   297.2256  -64.2467   87.3487 -836.9064  646.3574  804.8959  418.5344   62.269   580.2036 -182.7843 -136.5287 -273.6158 -272.6194 -213.6738]
[  71.0425 -561.035   437.823   218.8355  393.559  -873.2564  675.9291  553.9471 -342.774    89.6851  747.5839  -79.4283 -394.4356 -124.3889 -351.191  -170.232 ]
[  90.0643 -766.5428  648.0154   26.2052  593.9252 -790.1139  450.837   717.2585  145.1686  113.691   698.1406  -42.1764 -180.1768 -454.5199 -141.8052 -205.4352]
[ 126.8931 -786.9139 -610.2418 -604.7332  586.046  -600.985   293.8706  419.8469  729.6328 -312.3719  412.4849 -271.9994    6.4158 -186.4392 -508.3777 -117.8702]
[  82.1413 -838.3706  775.9773 -140.2707  629.8781 -866.4417  606.661   383.2791  336.0246   70.1044  717.5594 -293.7864   10.3741  -59.1588 -161.0171 -201.6375]
[ 100.6856 -899.5439  637.3646  -63.1814  458.7889 -794.1701  749.1285  552.491   268.8224   18.4344  758.0548 -290.5046  -68.9516  -50.1597 -188.4932 -147.8965]
[  186.3539 -1253.9832  381.2729   95.3265  130.2682 -572.407   871.3331  684.7855  102.0727   51.8202  710.558  -245.8456 -133.4272  -45.2817   -4.3448  -94.8438]
[   39.5919  -532.7025  660.6555 -175.579   496.2451 -1212.7599  543.1707  153.1054  304.2349  121.8692  316.8801 -120.0393  -33.3816 -150.9697 -296.9193 -239.1022]
```

```
[   32.0897  -630.6647  -369.2693  -668.5107  -241.708  -1027.352   288.3527  440.4044  217.5707  -42.077    63.4228  247.3692  -162.6502  -60.3011  -230.8555  -204.5678]
[  204.335  -1198.5562  -153.8141  -193.8116   -91.5712  -953.409   533.2166  513.4015  907.6662  -73.3157  538.6819 -208.5913  -138.513  -135.6953  162.2981  -257.1462]
[  123.7588  -676.2      -52.968    -62.4662  -199.6508 -1429.0216  542.0209  223.8322  881.4548 -167.2758  356.5691   24.8141  -205.4357  -40.624   -284.3024  -531.2003]
[  103.3884  -850.5956   81.9385  -354.621    95.8603 -1050.7972  558.0737  459.5218  996.2916  -65.7134  716.8214   42.8384   -93.7483  -41.2787   38.2696  -398.3458]
[  102.9547  -259.9876 -1409.3595  -400.7695  136.9886  -575.4068 -125.3157  171.4859  325.6539 -451.3325  178.7778   33.7225   165.2463  124.6157 -253.3336  -104.7712]
[   58.4464  -482.1593   206.1812  -207.7567  175.5117 -1499.729   400.2668  167.9725  711.3323  -76.9911  416.2975  162.4922   -99.5503  -76.1087  -48.7921  -430.8931]
[   90.9796  -763.2203    77.3428  -309.9025   62.248  -1116.7979  494.6873  472.1859  936.3806  -45.7683  659.0745   67.4453  -116.8444  -87.8915  -47.0523  -384.33  ]
[ 124.4954 -994.2679  -99.8577 -396.4412 -143.7961 -953.6685  665.0517  409.2108  968.6747  -81.7582  707.2399   54.3617 -212.9092  -32.5141   79.3546 -288.1552]
[   39.7752  -530.2493   286.0347  -409.3656  378.7139 -1281.6526  196.8184   90.5062  647.895    17.1474  374.6453  320.023     -3.5439  -159.3448  -166.4301  -394.9861]
```

c_k값 비교 다른 사람끼리의 비교

1. 앞선 2장의 슬라이드에서 총 6명의 사람을 비교해보았다. 아무래도 얼굴을 이루는 Principal Component로 비교한 것이기에 다른 사람이지만 각각의 EigenFace별로 비슷한 분포를 보이는 열이 있는가 하면 전혀 다른 분포를 보이는 열도 존재하는 것을 확인할 수 있다.

2. 각각의 EigenFace별로 비슷한 분포를 보이더라도 각 분포의 중심축이 조금씩 다른 것을 확인할 수 있다.

-805.1382	-787.2926	-1283.4324
-332.3031	-756.1008	-1138.2869
-401.6793	-849.8706	-1267.0476
-305.3358	-955.7893	-1215.3914
-778.3414	-1008.3315	-563.0649
-393.966	-623.7736	-1167.0195
-324.9737	-730.7146	-1192.5347
-555.8997	-1060.6724	-1156.1189
-308.8723	-663.6824	-1092.7476

Face Recognition Idea

Face Recognition은 총 2가지를 실험해보았다.

- 새로운 사진이 들어왔을 때, 어떤 사람인지를 판별한다.
- 두 개의 사진이 있을 때, 서로 같은 사람인지 다른 사람인지를 판별한다.

첫 번째를 수행하는 방법이다.

1. 값이 너무 들쑥날쑥한 열은 비교의 대상에서 제외한다.
2. 분포가 균일한 열도 극단적이 값은 조금씩 있으므로 최대값과 최소값을 제외하고 남은 값에서의 최대값과 최솟값 사이를 범위로 한다.
3. 새로운 사진이 들어왔을 때, 2에서 만들어낸 범위에 속하는 c_k값이 10개 이상이면 해당하는 사람으로 판별한다.
4. 새로운 사진이 필요하므로, 각 사람마다 8개의 사진으로 진행한다.

Face Recognition Idea

두 번째를 수행하는 방법이다.

1. 각 사람의 c_k값의 평균을 구한다. 각 평균의 차이들을 구한다. 이 차이들의 평균을 기준으로 정한다.
2. 두 사진이 들어왔을 때, 두 사진의 c_k값의 차이가 1에서 구한 기준보다 큰 것이 7개 이상이라면 다른 사람으로 판별한다. 그렇지 않으면 같은 사람으로 판별한다.

Face Recognition First

극단적인 값과 값이 들쑥날쑥한 열을 어떻게 처리할까를 고민하였다.

- 새로운 사진이 들어왔을 때, 어떤 사람인지를 판별한다.

1. 분포가 균일한 열도 극단적이 값은 조금씩 있으므로 최대값과 최소값을 제외하고 남은 값에서의 최대값과 최솟값을 뽑아낸다.
2. 새로운 이미지의 c_k값을 뽑아낸다.
3. 2에서 뽑아낸 값이 1에서 뽑아낸 범위에 속하는지를 판별한다.
4. 범위에 속하는 c_k값이 10개 이상이면 일치한다고 판별한다.
5. 들쑥날쑥한 열인 13열은 제외하였다.

```
for i in range(0, 14):
    true_count = 0
    for j in range(0, NUM_EIGEN_FACES):
        if ck_new[i, j] <= ck_min_max[i,0,j] and ck_new[i, j] >= ck_min_max[i,1,j]:
            if(j != 12):
                true_count = true_count + 1
    if true_count >= 10:
        print("Matching" + str(i + 1))
    print("------------------------------")
```

Face Recognition 결과

새로운 사진은 1사람당 1장씩 총 14장의 이미지로 테스트하였다. 아래와 같이 총 14장의 사진 중 7장의 사진이 동일한 인물로 판별되는 결과가 나왔다.

일치하는 c_k값을 7개 이상으로 줄여보았더니 14장의 사진 중 10장의 사진이 동일한 인물로 판별되는 결과가 나왔다.

10개 이상

7개 이상

Face Recognition Second

각 사람들의 일반적인 얼굴 즉, c_k값의 평균을 이용하면 어떨까?를 고민하였다.

- 두 개의 사진이 있을 때, 서로 같은 사람인지 다른 사람인지를 판별한다.

1. 각 사람의 c_k값의 평균을 구한다. 극단적인 값은 제외하기 위해 최대값과 최소값을 제외하고 평균을 구하였다. 각 평균의 차이들을 구한다. 이 차이들의 평균을 기준으로 정한다.
2. 두 사진이 들어왔을 때, 두 사진의 c_k값의 차이가 1에서 구한 기준보다 큰 것이 7개 이상이라면 다른 사람으로 판별한다. 그렇지 않으면 같은 사람으로 판별한다. 7개는 16의 절반에 못미치는 숫자여서 50%가 조금 안되는 정확도를 기대하고 설정하였다.

```
ck_except_m = np.zeros((14, 7, 16),dtype=np.float32)
ck_mean = np.zeros((14, NUM_EIGEN_FACES),dtype=np.float32)
ck_mean_difference = np.zeros((14 * 13, 16), dtype=np.float32)

for i in range(0, 14):
    for j in range(0, 16):
        tmp = np.sort(ck3d[i,:,j])
        tmp = tmp[1:8]
        ck_except_m[i,:,j] = tmp

for i in range(0, 14):
    ck_mean[i] = ck_except_m[i].mean(axis = 0)
print(ck_mean)

num = 0
for i in range(0, 14):
    for j in range(0, 14):
        if i != j:
            ck_mean_difference[num] = abs(ck_mean[i] - ck_mean[j])
            num = num + 1
ck_mean_difference_mean = ck_mean_difference.mean(axis = 0)
print(ck_mean_difference_mean)

ck_compare = np.zeros((2, NUM_EIGEN_FACES), dtype=np.float32)
compare_images = readTestImages("compare_two_images")
for i in range(0, 2):
    for j in range(0, NUM_EIGEN_FACES):
        ck_compare[i, j] = np.dot(compare_images[i], eigenFaces[j])
ck_difference = abs(ck_compare[0] - ck_compare[1])
print(ck_difference)

different_count = 0
for i in range(0, 16):
    if ck_mean_difference_mean[i] <= ck_difference[i]:
        different_count = different_count + 1
if different_count >= 7:
    print("Different Count : " + str(different_count))
    print("Different Person")
else:
    print("Different Count : " + str(different_count))
    print("Same Person")
```

Face Recognition 결과

총 10번의 테스트를 하였으며, 다른사람이지만 같다는 결과와 다르다는 결과, 같은 사람이지만 다르다는 결과와 같다는 결과 이렇게 총 4가지의 경우가 있다. 다른 사람이어도 비슷한 얼굴형을 갖고 있으면 일치하다는 결과를, 같은 사람이어도 조명과 안경 유무 등에 따라서 다른사람이라는 결과가 나오는 것을 알 수 있다. 기준보다 큰 것의 개수를 줄인다면 정확도가 올라갈 것이다.

다른 사람이지만 같다는 결과

다른 사람이 다르다는 결과

같은 사람이 같다는 결과

같은 사람이 다르다는 결과

마무리 느낀점

1. 이번 프로젝트는 데이터가 구글링해서 나왔으니 다행이지 일반적인 실험이었으면 데이터 수집부터가 고비였을 것 같다. 데이터 수집의 중요성을 알게 되었다.

2. 소수점 아래자리라도 결과에 큰 영향을 미칠 수 있다는 것을 알게 되었다. Precision의 오류를 체감하였다.

3. 조명과 안경의 유무, 얼굴형 등에 따라서 Face Recognition의 정확도가 많이 달라짐을 체감하였다.

4. 선형대수의 활용예시를 직접 경험해봄으로써 선형대수가 정말 중요함을 느꼈고 이런 분야에 수학이 적용된다는 것이 놀라웠다.

마무리

감사합니다.

https://github.com/cheol-hoon/Numerical_Analysis