수치해석 Project 1

Numerical analysis

JaeHoon KANG
강 재 훈

Project 1

eigenface

CONTENTS

Project 1

01 코드 설명

```
import numpy as np
import cv2
```

사용된 library

Numpy = 행렬의 계산을 위해 사용

Opencv = 이미지 처리를 위해 사용

Project 1

01 코드 설명

```
def makeDataSet():
    global curDir, srcDir, dstDir

    files = os.listdir(curDir+srcDir)
    files.sort()
    name = 0
    for i in range(len(files)):
        if( files[i].endswith(".pgm") ) :
            srcImg = cv2.imread(curDir+'/'+srcDir+'/'+files[i], cv2.IMREAD_COLOR)
            dstImg = cv2.cvtColor(srcImg, cv2.COLOR_BGR2GRAY)
            finImg = cv2.resize(dstImg, dsize=(32,32), interpolation=cv2.INTER_AREA)
            cv2.imwrite(curDir+'/'+dstDir+'/'+ str(name) +".pgm", finImg)
            name += 1
    print(name)
```

Data Set 만들기

R구글에서 찾은 이미지 data set을 32by32, grayscale한 이미지로 바꿔서 새로운 폴더에 저장

Project 1

01 코드 설명

```python
def makeA():
    global NUMBER_OF_DATA, WIDHT, HEIGHT, MEAN, A
    ori_A = np.ndarray( shape = (NUMBER_OF_DATA, WIDTH * HEIGHT), dtype = np.float64 )

    files = os.listdir(curDir+dstDir)
    files.sort()

    for i in range(len(files)):
        if( files[i].endswith(".pgm") ) :
            img = cv2.imread( curDir + '/' +dstDir + '/' + files[i], cv2.IMREAD_GRAYSCALE )
            ori_A[i] = img.flatten().astype('float64')

    MEAN = np.mean(ori_A, axis = 0)
    for i in range(NUMBER_OF_DATA):
        A[i] = ori_A[i] - MEAN

def makeEigenface():
    global A, eigenFace, COMPRESSION
    ATA = np.dot(A.T,A)
    U, s, V = np.linalg.svd(ATA, full_matrices = True)
    pyS = s.tolist() #s를 파이썬 리스트로 변경
    sortS = sorted(pyS, reverse=True) #eigen value가 큰 걸 찾기 위해 정렬
    for i in range(COMPRESSION):
        eigenFace[i] = V[ pyS.index(sortS[i]) ]
```

Eigenface 구하기

makeA = 32by32 이미지1285개를 읽어오고 그들의 MEAN 값을 빼주어 사용하고자 하는 A를 만들어 줍니다.

makeEigenface = ATA를 계산해 covariance matrix로 만들어준 후 numpy를 이용해 SVD를 실행합니다. 이후 eigen value가 큰 값들을 뽑아서 eigenface로 이용합니다. (1024개 중 40개만 이용)

Project 1

01 코드 설명

```
def makeOriImage(coeffi, filename) :
    global eigenFace, MEAN, curDir, recovery
    A = np.ndarray( shape = (1024), dtype = np.float64 )
    for i in range(len(eigenFace)):
        A += np.dot(coeffi[i], eigenFace[i])
    A += MEAN
    A = A.reshape(32,32).astype('int32')
    cv2.imwrite(curDir+'/'+recoveryDir+'/'+ filename, A)
    print("IN makeOri : ", filename)
    print(A)
```

Eigenface 활용

makeOriImage = 우리가 구한 coefficient와 eigenface로 원래 사진으로 복구해주는 함수입니다.

Project 1

01 코드 설명

```
def calCoeffi(ori_face):
    global eigenFace, COMPRESSION
    cList = []
    for i in range(COMPRESSION):
        cList.append( np.dot(ori_face, eigenFace[i]) )

    return cList
```

```
def onePersonManyImg(name) : #사람 한명에 대해 여러장의 사진을 이용하여 coefficient들의 값 비교
    global curDir, testDir, MEAN
    nowDir = curDir + testDir +'/' + name
    nowfile = os.listdir(nowDir)
    coeffiS = []
    for i in range(0, len(nowfile),1 ) :
        if( nowfile[i].endswith(".pgm") ) :
            srcImg = cv2.imread(nowDir+'/'+nowfile[i], cv2.IMREAD_COLOR)
            dstImg = cv2.cvtColor(srcImg, cv2.COLOR_BGR2GRAY)
            finImg = cv2.resize(dstImg, dsize=(32,32), interpolation=cv2.INTER_AREA)
            print("IN onePerson : ", nowfile[i])
            print(finImg)

            img = finImg.flatten().astype('float64')
            img -= MEAN

            coeffi = calCoeffi(img)
            makeOriImage(coeffi, nowfile[i])

            coeffiS.append( coeffi )
```

Eigenface 활용

onePersonManyImg = 인자로 해당 사람의 name을 받으면 eigenface를 이용해 coefficient를 계산해줍니다. 만약 한 사람당 사진이 5장이 있다면 서로 다른 조합의 coefficient list가 5개 들어 있는 coeffiS가 만들어집니다.

calCoeffi = 1개의 사진에 대해 coefficient 들을 계산해 주는 함수입니다.

Project 1

01 코드 설명

```
def cmpMe():
    global everyCoeffi, meResult
    everyCoeffi = np.array(everyCoeffi)
    pNum = 10
    iNum = 5
    for i in range(pNum):
        for j in range(iNum):
            for k in range(4,j,-1):
                tmp = everyCoeffi[i][j] - everyCoeffi[i][k]
                meResult[i].append(np.linalg.norm(tmp))

    for i in range(10):
        print("Person number", i+1 )
        print(meResult[i])
        print("MEAN : ",np.mean(meResult[i]))
        print()
```

```
def cmpPerson() : #다른 사람이면 coefficient가 어떻게 다른가 비교
    global everyCoeffi, manyReult, manyResult2
    everyCoeffi = np.array(everyCoeffi)
    pNum = 10
    iNum = 5
    tmp_k, tmp_l = 0,0
    for i in range(pNum):
        for j in range(i+1,pNum,1):
            tmpN = 10e9
            for k in range(iNum):
                for l in range(iNum):
                    tmpL = everyCoeffi[i][k] - everyCoeffi[j][l]
                    a = np.linalg.norm(tmpL)
                    if(tmpN > a):
                        tmp_k, tmp_l = k,l
                        tmpN = a
            manyResult[i].append([tmp_k,tmp_l,tmpN])

    tmp_k, tmp_l = 0,0
    for i in range(pNum):
        for j in range(i+1,pNum,1):
            tmpN = -1
            for k in range(iNum):
                for l in range(iNum):
                    tmpL = everyCoeffi[i][k] - everyCoeffi[j][l]
                    a = np.linalg.norm(tmpL)
                    if(tmpN < a):
                        tmp_k, tmp_l = k,l
                        tmpN = a
            manyResult2[i].append([tmp_k,tmp_l,tmpN])
```

manyResult: list

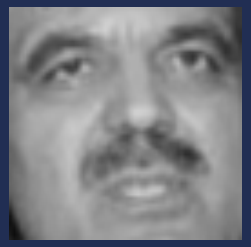
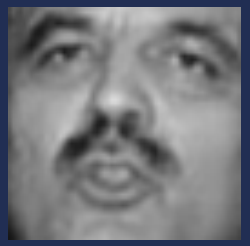
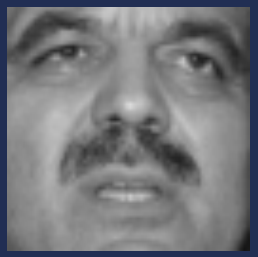
Eigenface 활용

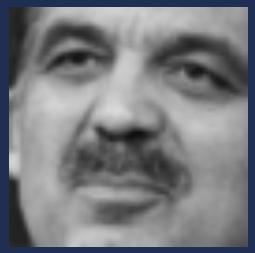
cmpMe = 1 사람당 5개의 coefficients 조합(5개의 이미지를 사용하니까)을 비교하여 분석하는 함수 입니다.

cmpPerson = 다른 사람과의 coefficients 조합 경우를 모두 조사하고 비교하여 한 사람과 다른 사람의 coefficients 조합의 차가 최소, 최대가 되는 경우를 저장하고 그 값을 비교합니다.

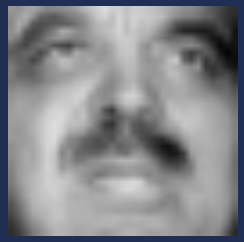
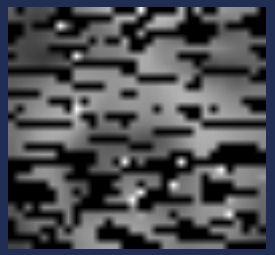
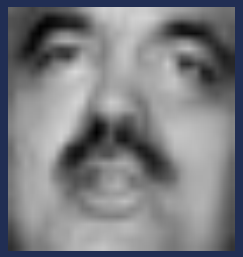
Project 1

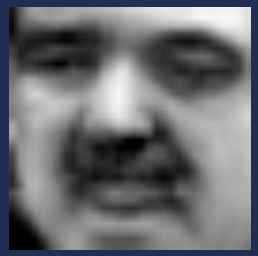
02 한 사람의 사진 5장에 대해



위의 사진들은 원본 이미지이고
아래 사진들은 eigenface와 coefficient로 구한 이미지 입니다.



Project 1

02 한 사람의 사진 5장에 대해

```
IN onePerson :  Abdullah_Gul_0001.pgm
[[141 140 146 ... 107  84  58]
 [138 139 144 ... 101  79  58]
 [136 137 135 ... 110  80  55]
 ...
 [ 28  83 111 ... 131 125 125]
 [ 19  59  97 ... 124 121 119]
 [ 23  34  82 ... 116 116 109]]
IN makeOri :  Abdullah_Gul_0001.pgm
[[152 146 161 ...  72  53  47]
 [141 158 175 ...  68  51  41]
 [136 172 151 ...  73  50  29]
 ...
 [ 63  91 125 ... 155 129  74]
 [ 51  73 107 ... 145 110  52]
 [ 46  60  89 ... 129  82  40]]
```

```
IN onePerson :  Abdullah_Gul_0002.pgm
[[ 83  80  92 ... 141 136 138]
 [ 69  76  86 ... 137 131 134]
 [ 71  78  88 ... 129 132 124]
 ...
 [ 54  87 110 ... 120 106  96]
 [ 41  71  99 ... 111  97  82]
 [ 28  54  86 ... 101  86  74]]
IN makeOri :  Abdullah_Gul_0002.pgm
[[                94 -9223372036854775808                  80 ...
                 140                 137 -9223372036854775808]
 [                90                  84                  72 ...
  -9223372036854775808 -9223372036854775808 -9223372036854775808]
 [-9223372036854775808 -9223372036854775808                  69 ...
  -9223372036854775808 -9223372036854775808    10857207337622810]
 ...
 [-9223372036854775808                  80                  98 ...
  -9223372036854775808                 107 -9223372036854775808]
 [-9223372036854775808 -9223372036854775808 -9223372036854775808 ...
                 112                 102                  89]
 [                67                  68                  82 ...
  -9223372036854775808 -9223372036854775808 -9223372036854775808]]
```

```
IN onePerson :  Abdullah_Gul_0003.pgm
[[110 103 127 ... 106  94  85]
 [ 99 117 144 ... 100  91  81]
 [ 98 135 125 ... 102  88  72]
 ...
 [112 127 134 ... 122 101  53]
 [ 99 115 126 ... 114  83  34]
 [ 89 102 117 ... 101  57  24]]
IN makeOri :  Abdullah_Gul_0003.pgm
[[ 98  87  80 ... 119 112 113]
 [ 81  79  67 ... 111 105 105]
 [ 81  76  66 ...  96 101  90]
 ...
 [ 70 104 132 ... 119  97  77]
 [ 59  87 117 ... 112  88  61]
 [ 47  70 102 ... 102  76  53]]
```

```
IN onePerson :  Abdullah_Gul_0004.pgm
[[103 155 185 ... 209 201 191]
 [ 91 156 171 ... 211 204 192]
 [ 90 141 132 ... 202 198 192]
 ...
 [ 54  38  21 ... 128 133 132]
 [ 45  26  13 ... 127 133 128]
 [ 33  23  11 ... 127 128 120]]
IN makeOri :  Abdullah_Gul_0004.pgm
[[192 182 166 ... 267 286 296]
 [175 161 136 ... 258 277 286]
 [164 150 151 ... 248 259 265]
 ...
 [ 38  25  23 ... 123 129 116]
 [ 41  29  21 ... 116 116 110]
 [ 34  34  23 ... 106 114 107]]
```

```
IN onePerson :  Abdullah_Gul_0005.pgm
[[ 41  39  67 ... 160 162 152]
 [ 40  39  71 ... 156 157 147]
 [ 40  36  67 ... 152 143 136]
 ...
 [ 36  34  96 ... 120 110 100]
 [ 37  35  58 ... 109 101  92]
 [ 40  37  45 ...  96  87  81]]
IN makeOri :  Abdullah_Gul_0005.pgm
[[ 26  89 142 ... 244 236 224]
 [ 12  89 129 ... 249 242 224]
 [ 11  76  93 ... 240 235 221]
 ...
 [-13 -11   7 ... 142 145 141]
 [-24 -31 -16 ... 138 141 132]
 [-32 -38 -31 ... 135 128 117]]
```

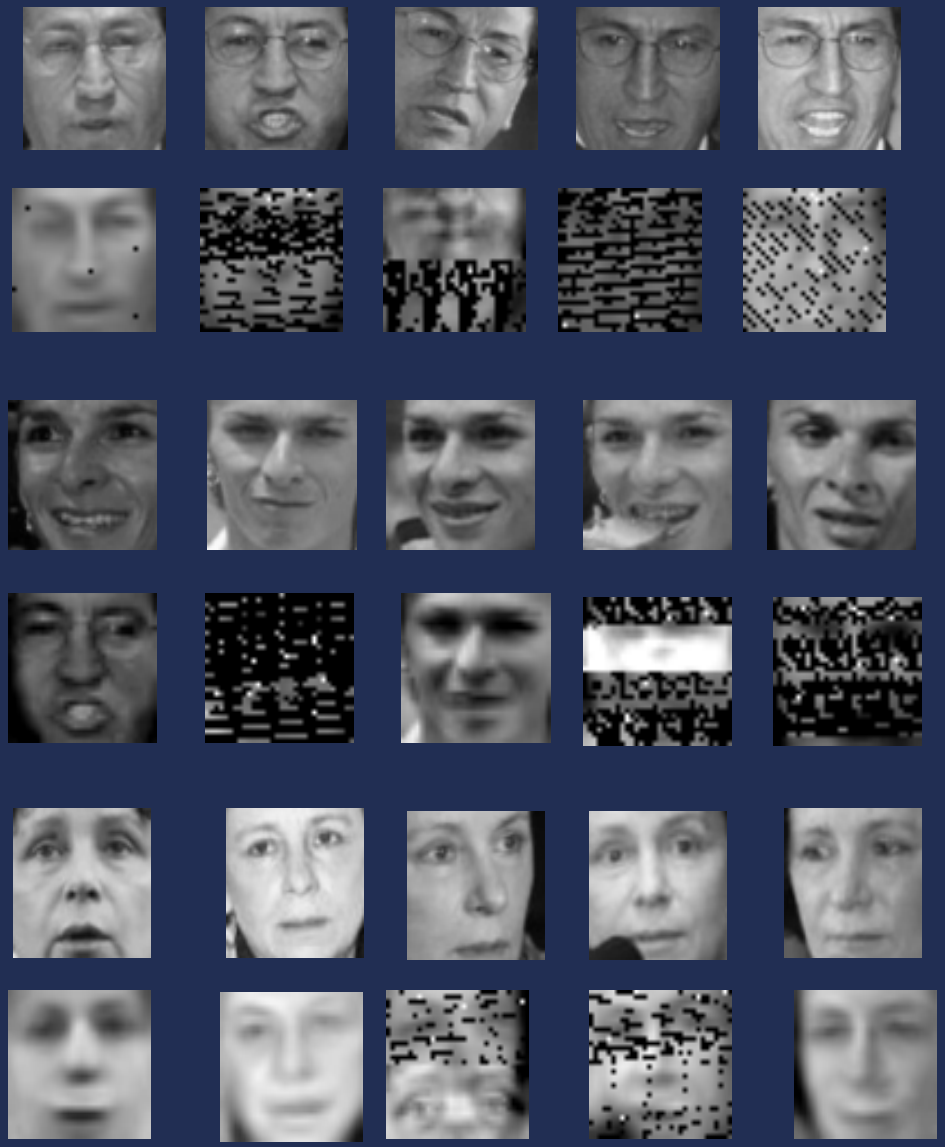
사진에서 위의 값이 원래 이미지의 값이고
아래 값이 복원한 이미지의 값입니다.

Project 1

02 한 사람의 사진 5장에 대해

Project 1

02 한 사람의 사진 5장에 대해



Project 1

02 한 사람의 사진 5장에 대해

```
Person number 1
[1727.1284733524299, 1576.0442544673765, 1447.4122904882927, 1543.1890807577006, 1093.8989587454682, 971.6729997938398, 905.4262461370843, 911.0471728047352, 567.3242535
944736, 788.4316977955917]
MEAN :  1153.1575427936991

Person number 2
[1323.5850912222568, 1573.5469111737048, 1271.0080877362457, 636.213426664497, 1660.107855289526, 1774.267922311473, 1530.2039618513224, 1432.6087708836385, 1766.1980951
489775, 996.1977184934893]
MEAN :  1396.393784077513

Person number 3
[1255.792349871548, 1226.6912539518169, 1114.9891538604036, 1088.101321130411, 1235.3607410832328, 1483.6363296934735, 1482.5588317726326, 1228.923531004801, 929.4119170
555142, 1174.3536157193635]
MEAN :  1221.9819045143197

Person number 4
[1267.9291886467809, 1099.6514227937826, 1989.813130858351, 927.2861730996357, 975.6143506021025, 1233.3844704919795, 1804.8123111485734, 1312.687829079434, 2202.2259561
295614, 1472.3717913619719]
MEAN :  1428.5776624212172

Person number 5
[1429.1853770591758, 1750.6933729638292, 864.6240112898562, 828.8685226137828, 1121.4706665333215, 1504.9824243705561, 932.8383825368007, 956.9509014455238, 1314.5813789
149763, 839.0799722167159]
MEAN :  1154.3275009944539
```

```
Person number 6
[1775.0589100463128, 1148.5783530235724, 1604.3211409640912, 1126.5965443049636, 2349.1200640721054, 1248.246504418737, 920.4597092507731, 2961.836566218457, 1317.204224
8857704, 2315.3823196584485]
MEAN :  1676.680433684323

Person number 7
[1779.4700554238889, 1337.6182337226737, 1318.7460899796852, 1238.2110936702074, 1118.505305406964, 952.1042068457342, 736.4395269342883, 1058.1425876160308, 912.8171569
647458, 1286.4562156308016]
MEAN :  1173.8510472195019

Person number 8
[1791.982253209551, 1608.182471034229, 1438.9358195656828, 1209.1169172269394, 1017.5766471655263, 1123.9707309175778, 654.7527741663067, 749.0534601933689, 1035.6411197
423297, 1131.563518228887]
MEAN :  1176.0775711450399

Person number 9
[1481.0984784884436, 1701.388589678298, 1342.029011472725, 1857.2379302332517, 1179.7343970843685, 851.4596867186815, 1016.7571563305547, 1143.041701468103, 983.1752370087017, 1154.7855010943347]
MEAN :  1271.0707689577462

Person number 10
[1610.0846479427253, 1402.6387354444846, 1865.9306971548622, 1979.9664013012596, 2074.336727435702, 1731.5993156008747, 2328.684194140404, 1554.5853648119974, 1468.5079478254047, 1555.6301499383376]
MEAN :  1757.1964181596054
```

1 사람당 5개의 img를 사용한다고 했을때
5개의 coefficient 조합이 나올 것이고 모든 경우의 수를
따져 그들의 차이의 크기를 구한 것입니다.
MEAN은 그들의 평균 값입니다.

Project 1

03 여러 사람의 coefficients 비교

```
Min result
Person number 1
[[3, 0, 840.8974149674643], [4, 4, 1047.9021363635884], [2, 4, 850.3237871365673], [3, 3, 959.8788706420339], [3, 3, 910.0969261312032], [0, 3, 1214.35133746222], [4, 0,
 849.3519551573163], [3, 1, 1038.5783578118114], [4, 4, 1139.3365533912822]]

Person number 2
[[1, 2, 1159.855783128138], [2, 2, 1079.2081131707118], [2, 4, 937.9204989697406], [0, 3, 935.2846855155926], [2, 1, 1244.3832406667698], [0, 0, 971.9940722239552], [2,
3, 930.1711994052388], [1, 2, 887.2482910803385]]

Person number 3
[[4, 4, 1075.326324726961], [2, 3, 1013.3681919645916], [2, 0, 973.8402634658996], [1, 3, 1424.5164417358428], [4, 1, 945.3307325978398], [4, 3, 1145.942745141838], [4,
4, 1089.109193941188]]

Person number 4
[[2, 4, 1216.7776565209142], [1, 1, 886.2449160984157], [2, 1, 1221.4033677066095], [4, 1, 960.548601427436], [2, 3, 752.4204989038851], [1, 0, 1145.8218535627652]]

Person number 5
[[4, 3, 869.2673128667525], [2, 0, 898.928285214439], [4, 2, 713.2040700385346], [0, 0, 913.7076356331291], [4, 4, 1370.7412986397085]]
```

```
Person number 6
[[3, 0, 1456.1530637896226], [3, 1, 723.1125036005827], [3, 1, 891.9412332367991], [1, 0, 996.6704817248074]]

Person number 7
[[0, 4, 945.3003786502004], [2, 0, 900.7514971428989], [3, 4, 1879.7961923303385]]

Person number 8
[[1, 3, 991.7552902870764], [0, 2, 1105.5089292873427]]

Person number 9
[[3, 4, 1260.3244592934277]]

Person number 10
[]
```

가능한 모든 경우의 수를 탐색해 한 사람에 대해 나머지 사람들과의 coefficient의 차가 최소가 되는 사진의 조합을 계산해 저장합니다. (ex) Person number 1의 [3,0,840...]은 첫번째 사람과 두번째 사람은 첫번째 사람의 네번째 사진, 두번째 사람의 첫번째 사진일 경우 가장 유사한 coefficient 조합을 갖는다.)

Project 1

03 여러 사람의 coefficients 비교

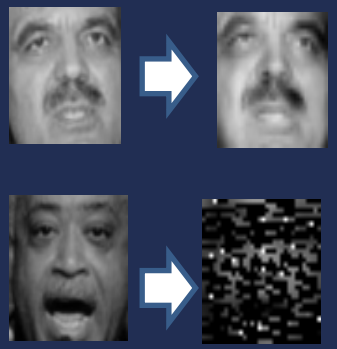
```
Max result
Person number 1
[[0, 1, 2294.165755247129], [0, 3, 2021.9440081984567], [0, 3, 2399.596241321612], [4, 0, 2165.2484142612657], [0, 4, 3291.6700269417242], [4, 4, 2242.6945916281825], [0
, 0, 2095.66493168867961, [4, 0, 2245.5735583260794], [0, 1, 3008.719062186371]]

Person number 2
[[3, 0, 1932.6147048277105], [2, 3, 2211.202600382821], [1, 0, 2264.511862856351], [2, 4, 2771.678365240963], [1, 4, 2659.4218057411636], [3, 4, 1696.8189976871465], [1,
 0, 2488.2656045179338], [2, 1, 2726.6217880988065]]

Person number 3
[[0, 3, 2293.855565761166], [2, 0, 2419.075271917537], [1, 4, 2631.2429954487043], [3, 4, 2567.561279393657], [3, 4, 1868.5148654494396], [2, 0, 2586.1722791787383], [1,
 1, 2652.35534828147]]

Person number 4
[[3, 0, 3035.7113200964964], [2, 4, 3008.2631294433486], [3, 0, 3144.6300393419061, [3, 4, 2541.7939090571062], [3, 0, 3175.840825599118], [2, 1, 2835.9870255575447]]

Person number 5
[[0, 4, 3487.0637508422556], [3, 4, 1873.9708714654907], [0, 0, 2494.8047731835577], [3, 0, 1885.0858368003157], [0, 1, 3577.9351437717305]]
```

```
Person number 6
[[4, 4, 3845.2579612535956], [4, 4, 2971.1996031423914], [4, 0, 3765.8508948727317], [2, 1, 2743.5518984591095]]

Person number 7
[[4, 0, 2554.308633382379], [4, 4, 1880.7252251238006], [4, 1, 3617.0840536791607]]

Person number 8
[[0, 0, 2616.8744806721443], [4, 1, 3100.0465481826595]]

Person number 9
[[0, 1, 3777.7040039629824]]

Person number 10
[]
```

가능한 모든 경우의 수를 탐색해 한 사람에 대해 나머지 사람들과의 coefficient의 차가 최대가 되는 사진의 조합을 계산해 저장합니다. (ex) Person number 1의 [0,1,2294...]은 첫번째 사람과 두번째 사람은 첫번째 사람의 첫번째 사진, 두번째 사람의 두번째 사진일 경우 가장 다른 coefficient 조합을 갖는다.)

Project 1

04 결과분석 및 한계점



```
Coefficients :  -170.212703214682
EigenFace :  [-3.76321794e-02 -2.66827289e-02  8.25059259e-05 ... -2.01898752e-02
 -2.34199523e-02 -5.81109085e-03]

Coefficients :  -170.31958519972463
EigenFace :  [0.00593483 0.02681586 0.03626254 ... 0.02609102 0.04916518 0.06145025]

Coefficients :  102.79698877196546
EigenFace :  [-0.02475892 -0.02118273  0.00624634 ... -0.02856221 -0.03536568
 -0.02636251]

Coefficients :  156.57751832437447
EigenFace :  [ 0.04472446  0.03123843 -0.00115623 ... -0.01134799 -0.00354303
 -0.0023119 ]

Coefficients :  74.64273711680747
EigenFace :  [-0.03089908 -0.00965445  0.0018237  ... -0.05262379 -0.08446929
 -0.08412762]

Coefficients :  22.532057885428113
EigenFace :  [ 0.05462253  0.02486372 -0.01617225 ...  0.05073966  0.05225009
  0.03453744]

Coefficients :  -46.65153613212884
EigenFace :  [-0.04120284 -0.05519393 -0.04325092 ... -0.02344733  0.00795332
  0.04373875]

Coefficients :  64.53415208224837
EigenFace :  [ 0.04013849  0.03151845  0.01793378 ... -0.01437513  0.01189276
  0.02706826]
```

```
IN makeOri :  Ahmed_Chalabi_0002.pgm
[[              184             188             185 ...
                103              98             100]
 [              185             186             182 ...
                104             100             104]
 [              184             184             178 ...
                 97              99             105]
 ...
 [              110             118             123 ...
                112             109             103]
 [              108             109             118 ...
  -9223372036854775808             103              95]
 [              114             108             111 ...
  -9223372036854775808              98              90]]
```

먼저 한사람에 대해 사진 5장을 사용하여 coefficient 조합들을 구하고 이것을 이용해 원래 사진을 복원해 보았습니다.

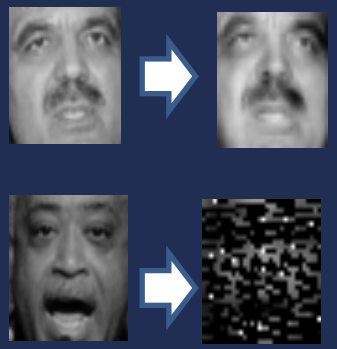
그 결과 50장의 사진에 대해 원본과 결과가 비슷한 경우도 있었지만 많은 경우에서는 노이즈가 껴있거나 아예 다른 사람(혹은 사진)으로 복원하는 경우를 볼 수 있었습니다. 이는 1024개의 eigenface를 사용하지 않고 40개의 eigenface만을 사용하기 때문이기도 하지만 더 큰 문제는 floating point의 연산 과정 중 발생하는 문제입니다.

A와 coefficient, eigenface 등을 계산 하는 과정에서 이미 정확한 값을 계산하지 못하고 이후 연산과정에서도 정확한 값을 계산하지 못해 문제가 생깁니다.

실제로 coefficient와 eigenface 연산의 과정중 floating point 연산 처리의 문제로 엉뚱한 값으로 바뀌는 경우도 발생합니다.

Project 1

04 개선방안



```
Coefficients :  -170.212703214682
EigenFace :  [-3.76321794e-02 -2.66827289e-02  8.25059259e-05 ... -2.01898752e-02
 -2.34199523e-02 -5.81109085e-03]

Coefficients :  -170.31958519972463
EigenFace :  [0.00593483 0.02681586 0.03626254 ... 0.02609102 0.04916518 0.06145025]

Coefficients :  102.79698877196546
EigenFace :  [-0.02475892 -0.02118273  0.00624634 ... -0.02856221 -0.03536568
 -0.02636251]

Coefficients :  156.57751832437447
EigenFace :  [ 0.04472446  0.03123843 -0.00115623 ... -0.01134799 -0.00354303
 -0.0023119 ]

Coefficients :  74.64273711680747
EigenFace :  [-0.03089908 -0.00965445  0.0018237  ... -0.05262379 -0.08446929
 -0.08412762]

Coefficients :  22.532057885428113
EigenFace :  [ 0.05462253  0.02486372 -0.01617225 ...  0.05073966  0.05225009
  0.03453744]

Coefficients :  -46.65153613212884
EigenFace :  [-0.04120284 -0.05519393 -0.04325092 ... -0.02344733  0.00795332
  0.04373875]

Coefficients :  64.53415208224837
EigenFace :  [ 0.04013849  0.03151845  0.01793378 ... -0.01437513  0.01189276
  0.02706826]
```

```
IN makeOri :  Ahmed_Chalabi_0002.pgm
[[              184              188              185 ...
                103               98              100]
 [              185              186              182 ...
                104              100              104]
 [              184              184              178 ...
                 97               99              105]
 ...
 [              110              118              123 ...
                112              109              103]
 [              108              109              118 ...
  -9223372036854775808              103               95]
 [              114              108              111 ...
  -9223372036854775808               98               90]]
```

먼저 data set의 수를 늘려 더 정확한 eigenface를 구해내는 것도 하나의 방법이 될 것입니다.

두번째로는 사람 사람을 더 잘 구별하기 위해선 여러 사람에 대해 적은 수의 사진을 구하는 것이 아니라 같은 사람에 대해 적절한 수의 사진을 구해 data set으로 사용 하는 것입니다.

세번째로는 같은 사람의 사진을 구할때 적절한 사진을 구하는 것입니다. 정면을 바라보고 있는가 ?, 측면을 바라보고 있는가?, 안경을 착용하고 있는가 등은 그 사람을 구별해내는데 중요한 요인으로 미쳤습니다. 따라서 목적에 맞는 사진을 구해야 합니다.

네번째로는 주어진 조건에서 최대한의 성능을 갖는 정확한 수의 eigenface를 설정해 compression 하는 것입니다. (우리는 단순히 eigenface를 40개만 썼는데 그러지 말고 최적화된 갯수를 구하면 성능개선)

다섯 번째로는 어느 eigenface가 그 사람을 결정하는데 얼마 만큼의 영향을 주는지는 파악해 해당 eigenface에 곱해지는 c에 가중치를 주는 방법입니다.

마지막으로 floating point 연산을 할때 숫자에 조금씩의 가공을 더해줘 유효한 숫자를 가지도록 설정해주는 것입니다.

Project 1

04 결과분석 및 한계점

```
Person number 1
[1727.1284733524299, 1576.0442544673765, 1447.4122904882927, 1543.1890807577006, 1093.8989587454682, 971.6729997938398, 905.4262461370843, 911.0471728047352, 567.3242535
944736, 788.4316977955917]
MEAN :  1153.1575427936991
```

```
Person number 6
[[3, 0, 1456.1530637896226], [3, 1, 723.1125036005827], [3, 1, 891.9412332367991], [1, 0, 996.6704817248074]]

Person number 7
[[0, 4, 945.3003786502004], [2, 0, 900.7514971428989], [3, 4, 1879.7961923303385]]
```

```
Person number 6
[[4, 4, 3845.2579612535956], [4, 4, 2971.1996031423914], [4, 0, 3765.8508948727317], [2, 1, 2743.5518984591095]]

Person number 7
[[4, 0, 2554.308633382379], [4, 4, 1880.7252251238006], [4, 1, 3617.0840536791607]]

Person number 8
[[0, 0, 2616.8744806721443], [4, 1, 3100.0465481826595]]
```

사진1 – 한사람에 대해 여러 사진, 사진2 – 다른사람과의 비교 중 최소값, 사진3 – 비교중 최대값

한 사람에 대한 사진들의 coefficient 조합의 차이를 계산해보았더니 평균적으로 1100~ 1600에서 정도 차이가 나는 것을 확인 할 수 있었습니다.

또한 한사람과 다른 사람들에 대한 coefficient 조합의 차이를 계산해보았더니 가장 작은 경우에는 900정도, 가장 큰 경우에는 3900정도의 차이를 보이는 것을 확인하였습니다.

따라서 같은 사람의 경우 어느 ci(0<=i<40)가 크고 작냐에 따라 조금씩의 차이는 있겠지만 전반적으로 coefficient 조합들이 크게 엇나가지 않는다고 생각 할 수 있고
다른 사람들을 비교한 경우 같은 사람을 비교하였을 때보다 작아지는 경우도 있지만 이 경우보단 매우 커지는 경우의 두 조합의 차이가 훨씬 큽니다.

따라서 앞서 원래 이미지로 복구하는 과정에서 정확하게 복원하는 경우도 있었지만 전혀 쌩뚱맞은 사람으로 복구되는 경우도 볼 수 있었습니다.

Project 1

04 개선방안

```
Person number 1
[1727.1284733524299, 1576.0442544673765, 1447.4122904882927, 1543.1890807577006, 1093.8989587454682, 971.6729997938398, 905.4262461370843, 911.0471728047352, 567.3242535
944736, 788.4316977955917]
MEAN :  1153.1575427936991
```

```
Person number 6
[[3, 0, 1456.1530637896226], [3, 1, 723.1125036005827], [3, 1, 891.9412332367991], [1, 0, 996.6704817248074]]

Person number 7
[[0, 4, 945.3003786502004], [2, 0, 900.7514971428989], [3, 4, 1879.7961923303385]]
```

```
Person number 6
[[4, 4, 3845.2579612535956], [4, 4, 2971.1996031423914], [4, 0, 3765.8508948727317], [2, 1, 2743.5518984591095]]

Person number 7
[[4, 0, 2554.308633382379], [4, 4, 1880.7252251238006], [4, 1, 3617.0840536791607]]

Person number 8
[[0, 0, 2616.8744806721443], [4, 1, 3100.0465481826595]]
```

사진1 – 한사람에 대해 여러 사진, 사진2 – 다른사람과의 비교 중 최소값, 사진3 – 비교중 최대값

한사람의 여러 사진에 대해 각각의 사진들이 많은 영향을 받는 eigenface를 조사하고 해당 eigenface에 곱해지는 c 값에 가중치를 줌으로써 한 사람의 여러사진들의 차이 값을 줄일 수 있을 것입니다.

위의 방식이 이루어지고 나면 각각의 사람들 간의 c값들의 차가 더욱 벌려질 것이고 같은 사람으로 분류할 가능성은 더 낮아질 것입니다.

또한 각각 다른 사람이라면 유사한 이미지는 최대한 피해주고 어느 사람인지 구별하기 어려운 애매한 이미지는 사용하지 않도록 합니다.

Project 1

05 Goal of project1

Understand the basis vectors and linear combination in real data

-> 32by32 pgm 이미지를 1024 길이의 벡터로 변경할 수 있었고 여러장의 이미지를 모아 만든 1024 길이의 basis vector들 중 몇개를 선정해 eigenface로 사용하였다. 이후 이 eigenface를 linear combination해서 원래의 real data로 복원 할 수 있었다.

Practice to find orthonormal basis vector using SVD

-> numpy library를 이용해 SVD 하는 법을 배웠고 그냥 A행렬을 넣는 것이 아니라 ATA로 만들어 SVD를 함으로서 연산을 효율적으로 할 수 있다는 것을 이해했다. 또한 1024개의 vector를 사용하지 않고 몇개의 vector를 뽑아 사용함으로서 1개의 이미지를 복원 할때 coefficient들과 1024보다 훨씬 적은 수의 eigenface를 이용하니 data compression과 memory reduction이 구현 된다는 것을 깨달을 수 있었다.

감사합니다

THANK YOU

JaeHoon KANG
강 재 훈