Eigenface - PCA

목차

1. Init Project

- 많은 face sample data를 통해 각 얼굴을 구현할 때의 Base인 Eigenface를 구할 수 있습니다.
- Eigenface를 통해 Test image를 분석, 재구성을 할 수 있습니다.
- 재구성을 할 때 쓰이는 Coefficients의 비교를 통해 test 이미지가 동일 인물인지 유사도를 측정할 수 있습니다.
- Sample Data의 양과 Eigenface의 개수를 조절해 가면서 동일 인물과 다른 인물에 대한 유사도 분석을 통해 Face recognition의 정확도를 비교할 것입니다.

2. Collect Face Image

```python
def change(path):
    path_obj = Path(path)
    idx = 1
    for item in path_obj.iterdir():
        person = cv2.imread(str(item))
        if person is None:
            break
        person = cv2.resize(person, dsize=(32, 32))
        person = cv2.cvtColor(person,  cv2.COLOR_BGR2GRAY)
        cv2.imwrite('./images/'+str(idx)+'.png', person)
        # cv2.imshow("person", person)
        # cv2.waitKey()
        # cv2.waitKey()
        idx += 1

def readImages(folder):
    images = []
    idx = 1
    for filename in os.listdir(folder):
        img = cv2.imread(os.path.join(folder,filename), cv2.IMREAD_GRAYSCALE)
        if idx > 3000:
            break  # 모든 하위 항목을 순회하며

        if img is not None:
            images.append(img)
        idx += 1

    return images
```

- http://vis-www.cs.umass.edu/lfw/의 cropped version인 13,233장의 사진을 얻어 왔습니다.
- 사진의 크기를 change 함수를 통해 32*32로 변환한 다음 다시 저장합니다.
- readImages 함수를 통해 32*32의 사진 모음을 Grayscale의 image array로 만듭니다.
- Idx를 이용해 1차 테스트는 3000장, 2차 테스트는 7000장, 3차 테스트는 13,233장을 이용할 것입니다.

3. Construct Data Matrix

```python
def createDataMatrix(images):
    print("Creating data matrix",end=" ... ")
    '''
    Allocate space for all images in one data matrix.
        The size of the data matrix is
        ( w  * h  * 3, numImages )

        where,

        w = width of an image in the dataset.
        h = height of an image in the dataset.
        3 is for the 3 color channels.
        '''

    numImages = len(images)
    sz = images[0].shape
    data = np.zeros((numImages, sz[0] * sz[1]), dtype=np.float32)
    for i in range(0, numImages):
        image = images[i].flatten()
        data[i,:] = image

    print("DONE")
    return data
```

- 2번 과정을 통해 생성된 Image array의 각 image가 data array에 들어가게 됩니다.
- 각 이미지마다 32 * 32이므로 배열에 들어갈 때 1 * 1024로 들어가게 됩니다.
- 따라서 이미지의 개수가 num일 때 최종적으로 생성되는 shape가 (num,1024)인 Data Matrix 됩니다.

4. Apply SVD(PCA)

```
# Number of EigenFaces
NUM_EIGEN_FACES = 30

# Maximum weight
MAX_SLIDER_VALUE = 255

# Compute the eigenvectors from the stack of images created
print("Calculating PCA ", end="...")
mean, eigenVectors = cv2.PCACompute(data, mean=None,
    maxComponents=NUM_EIGEN_FACES)
print ("DONE")
averageFace = mean.reshape(sz)

eigenFaces = [];

for eigenVector in eigenVectors:
    eigenFace = eigenVector.reshape(sz)
    eigenFace = eigenFace * 500 + 128
    eigenFace = eigenFace.astype(np.uint8)
    eigenFaces.append(eigenFace)
    output = cv2.resize(eigenFace, (0, 0), fx=10, fy=10)
    cv2.imshow("Result", output)
    cv2.waitKey()

# Create window for displaying Mean Face
cv2.namedWindow("Result", cv2.WINDOW_AUTOSIZE)
averageFace = averageFace.astype(np.uint8)
# Display result at 2x size
output = cv2.resize(averageFace, (0, 0), fx=10, fy=10)
#cv2.imshow("Result", output)
#cv2.waitKey()
```

- OpenCV Library에 있는 PCACompute 함수를 이용해서 SVD(PCA)를 진행합니다.
- SVD(PCA)를 수행하면 주어진 sample date의 평균 벡터 mean과 설정한 개수에 따른 eigenvector array가 반환이 됩니다.
- Eigenfaces의 개수를 30, 100, 250, 500개로 설정을 해 Eigenface의 개수와 coefficient의 상관관계를 그리고 인식의 정확도를 비교합니다.

5. Find the Coefficients

```
def findCoefficents(test_data):
    global NUM_EIGEN_FACES
    coef=[[0 for x in range(NUM_EIGEN_FACES)] for y in range(5)]
    for i in range(len(test_data)):
        prac = test_data[i] - mean

        for j in range(len(eigenVectors)):
            coef[i][j] = np.dot(prac, eigenVectors[j])

    return coef
```

- 주어진 Test data 5장을 eigenface를 통해 coefficients를 구합니다.
- Test data에 기존에 구했던 평균 mean vector를 빼고 eigenface를 내적을 해주면 해당 eigenface에 해당하는 coefficients 값이 나옵니다.

6. Test Face Recognition

```
def checkSimilarity(coef1, coef2):
    dist = [[0 for x in range(5)] for y in range(5)]
    for i in range(5):
        for j in range(5):
            dist[i][j] = np.linalg.norm((coef1[i])-(coef2[j]))

    for i in range(5):
        for j in range(5):
            dist[i][j] = np.dot(coef1[i], coef2[j]) \
            /(np.linalg.norm(coef1[i])*np.linalg.norm(coef2[j]))

    for i in range(5):
        print(dist[i])
```

- 두 사진 각각에 대한 coefficient 배열을 받습니다.
- 두 벡터 coef1, coef2 사이의 거리를 Euclidean distance로 구해 coefficients를 비교하고 유사도를 측정합니다.
- 벡터의 크기를 고려하지 않는 유사도 측정법인 cosine similarity를 이용해 유사도를 구하고 비교합니다.

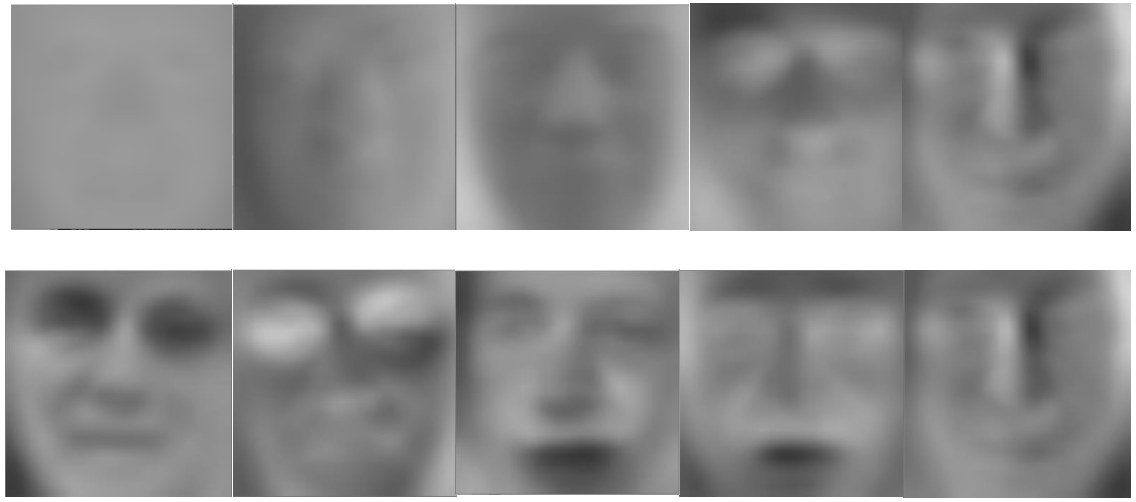
7. Generate Face Image using eigenfaces

```
def generateFaceImage(test_data, coef):
    for i in range(len(test_data)):
        print("image {}".format(i))
        recreate = np.zeros((1, 1024))

        for j in range(len(eigenVectors)):
            recreate += coef[i][j][0] * eigenVectors[j]

        recreate += mean
        rec_img = recreate.reshape(sz)
        rec_img = rec_img.astype(np.uint8)
        output = cv2.resize(rec_img, (0, 0), fx=10, fy=10)
        cv2.imshow("Result", output)
        cv2.waitKey()
```

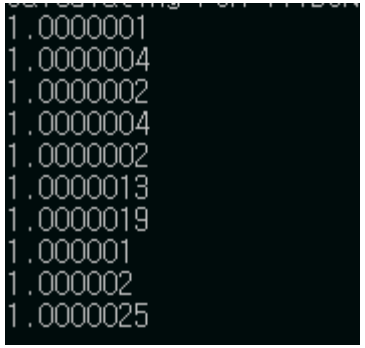
- Coefficients와 대응되는 eigenvector의 선형 결합한 것을 recreate에 더해줍니다.
- 평균 벡터인 mean을 더한 다음 32*32로 변환하여 화면에 출력합니다.

8. Eigenfaces

Eigenfaces



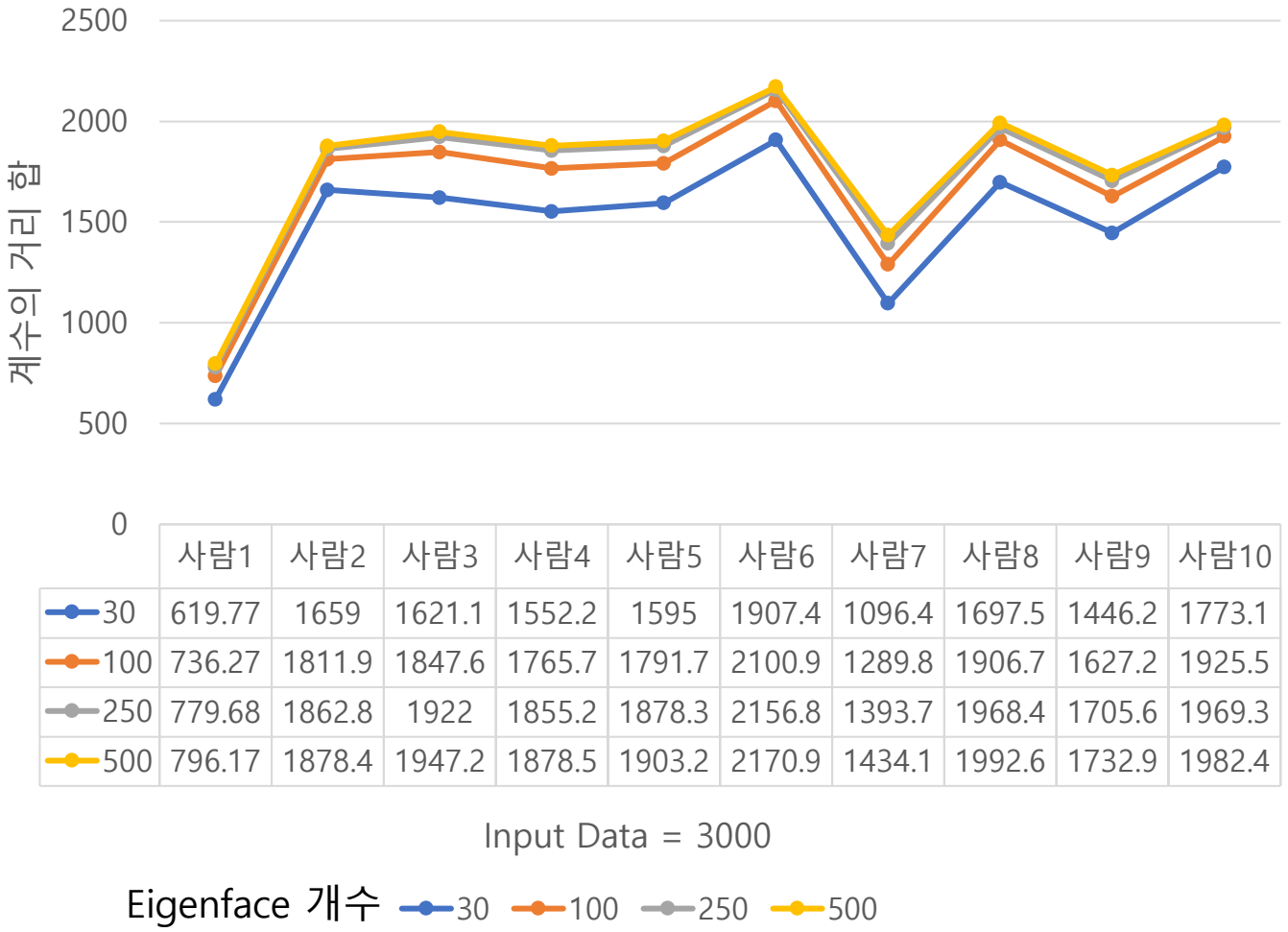
Eigenface의 크기



분석

- PCA를 통해 상위 singular value 10개에 해당하는 eigenface를 출력하였습니다.
- Eigenface는 범위가 -0.1 ~ 0.1 이었으므로 이 값을 적절히 0 ~ 255사이의 값이 되게 하기 위해 eigenface에 750을 곱한다음 128을 더하였습니다.
- Eigenvector의 크기를 구해보니 소수 오차를 감안하면 크기가 1인 것을 알 수 있습니다. 따라서 Orthonormal vector임을 알 수 있습니다.

9. Check Similarity



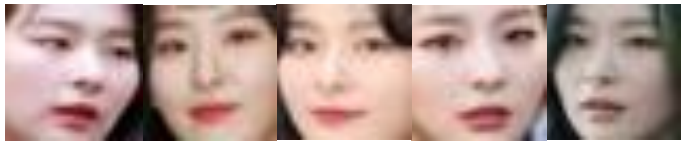
	사람1	사람2	사람3	사람4	사람5	사람6	사람7	사람8	사람9	사람10
30	619.77	1659	1621.1	1552.2	1595	1907.4	1096.4	1697.5	1446.2	1773.1
100	736.27	1811.9	1847.6	1765.7	1791.7	2100.9	1289.8	1906.7	1627.2	1925.5
250	779.68	1862.8	1922	1855.2	1878.3	2156.8	1393.7	1968.4	1705.6	1969.3
500	796.17	1878.4	1947.2	1878.5	1903.2	2170.9	1434.1	1992.6	1732.9	1982.4

Input Data = 3000

Eigenface 개수 30 100 250 500

사람1

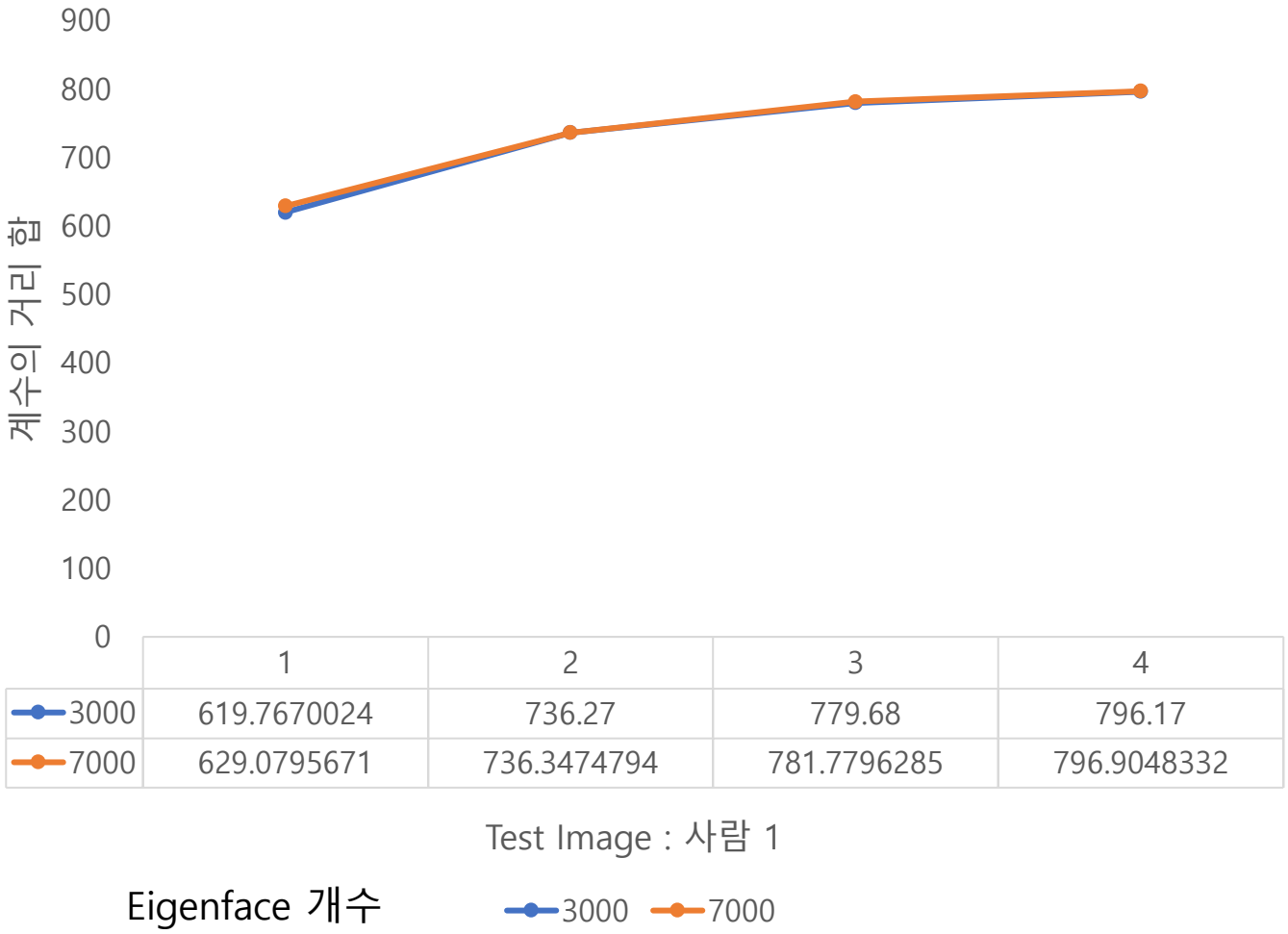
사람6



분석

- 6번 사람을 보면 화장법과 사진 찍은 각도 조명이 모든 사진마다 다르므로 유사도가 낮은 것을 볼 수 있습니다.
- 그에 반해 일정한 기준으로 얼굴을 찍은 1번 데이터의 유사도는 높은 것으로 나왔습니다.

9. Check Similarity

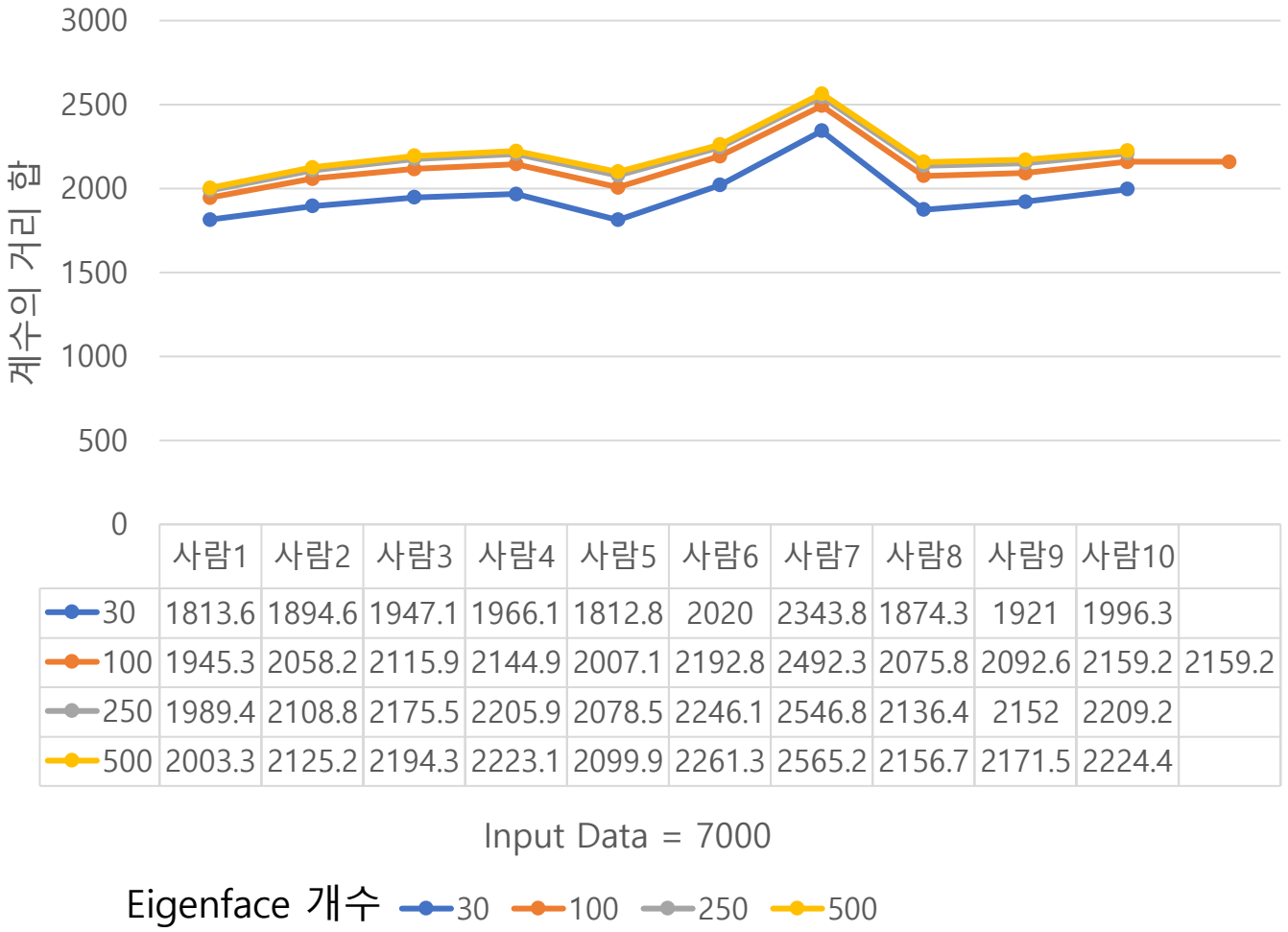


	1	2	3	4
3000	619.7670024	736.27	779.68	796.17
7000	629.0795671	736.3474794	781.7796285	796.9048332

분석

- Sample Data가 3000에서 7000으로 바뀌더라도 유사도가 크게 바뀌지 않는 것을 볼 수 있습니다.
- Data는 랜덤하게 들어오기에 평균벡터와 Eigenface가 크게 달라지지 않기 때문입니다.

9. Check Difference



	사람1	사람2	사람3	사람4	사람5	사람6	사람7	사람8	사람9	사람10	
30	1813.6	1894.6	1947.1	1966.1	1812.8	2020	2343.8	1874.3	1921	1996.3	
100	1945.3	2058.2	2115.9	2144.9	2007.1	2192.8	2492.3	2075.8	2092.6	2159.2	2159.2
250	1989.4	2108.8	2175.5	2205.9	2078.5	2246.1	2546.8	2136.4	2152	2209.2	
500	2003.3	2125.2	2194.3	2223.1	2099.9	2261.3	2565.2	2156.7	2171.5	2224.4	

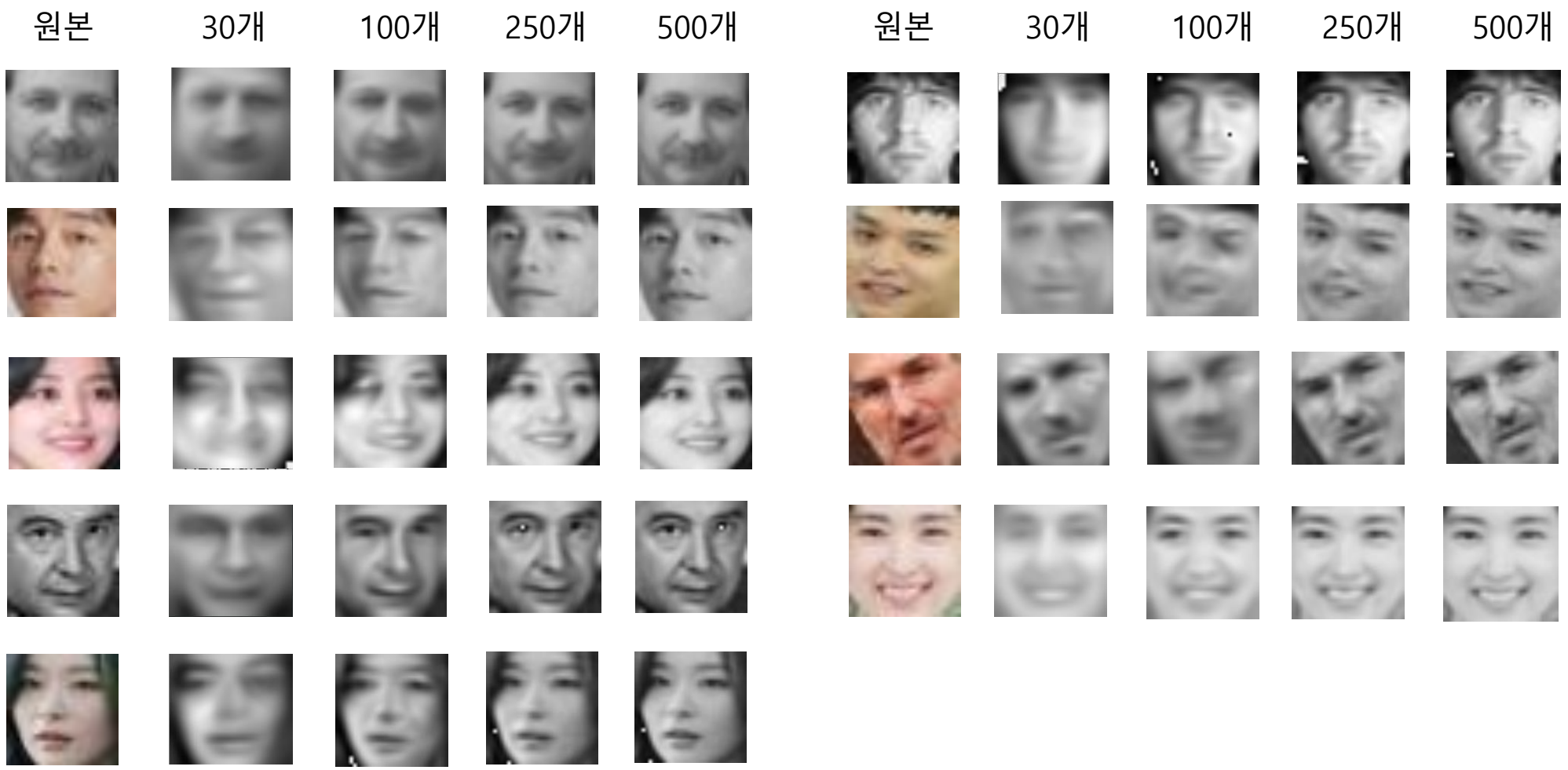
분석

- 이 결과는 사람 1명과 나머지 9명을 비교한 값의 평균을 각 eigenface 개수로 보여준 결과입니다.
- Eigenface의 개수마다 동일인물인지 아닌지 판단하는 수는 다를 것입니다.
- 500을 기준으로 동일인물인지 아닌지 판단하는 기준은 2000입니다.
- 거리의 합이 낮은 값일 수록 닮은꼴 모양의 얼굴임을 알 수 있습니다.

10. Generated Face Image

Sample Data : 10000개

개수 : Eigenface 개수



10. Generated Face Image

분석

- Eigenface 값이 증가하게 되면 꽤 근사하게 원본에 가까워지게 됩니다.

- Eigenface 개수에 따라 재현되는 이미지가 바뀌는 이유는 Basis face인 eigenface의 개수가 많아지므로 기존 얼굴을 표현할 수 있는 가짓수가 많아지게 되는 차이입니다.

- 그래도 1024 차원의 벡터 얼굴이지만 그보다 적은 양의 30개의 eigenface여도 얼굴형과 바라보는 위치, 이목구비의 위치를 잘 표현 하는 것을 볼 수 있습니다.

11. Conclude

- PCA기법을 통해 Eigenface를 추출하고 그것을 이용하여 test image를 인식하는 과정을 배웠습니다.
- 소수 계산의 오차, 수집한 자료의 양이 충분하지 않았던 것이 인식률을 낮추는 원인이었습니다.
- 1024차원의 얼굴을 표현할 때 1024개의 eigenface가 필요한 것이 아닌 그보다도 적은 숫자로 원본을 표현할 수 있다는 것을 알았습니다.
- 이것은 후에 방대한 양의 데이터가 들어오고 한정적인 시간이 있을 때 좋은 기법이 될 것 같습니다.
- 다양한 수치해석과 관련된 open library를 이용해 효율적으로 처리할 수 있었습니다.
- 유사도의 측정을 할 때 cosine similarity도 이용해 보았지만 coefficients 간에 상관관계가 있어 이번 과제와는 잘 맞지 않았습니다.

12. References

- http://vis-www.cs.umass.edu/lfw/
- https://www.learnopencv.com/eigenface-using-opencv-c-python/
- https://mikedusenberry.com/on-eigenfaces
- https://sandipanweb.wordpress.com/2018/01/06/eigenfaces-and-a-simple-face-detector-with-pca-svd-in-python/
- https://kh-kim.gitbook.io/natural-language-processing-with-pytorch/00-cover-4/07-similarity
- https://pythonmachinelearning.pro/face-recognition-with-eigenfaces/