

Bases 찾는법

• Gram-Schmidt Orthogonalization

2개의 vector 선택 \rightarrow proj으로 사자로 벡터 찾기



Bases

• Eigen vector decomposition

\downarrow Bases

square vector ($n \times n$), n 개의 eigen vector

• Singular Value Decomposition (SVD)

\rightarrow 어떤 행렬 type도 다 구할 수 \circ \rightarrow 양의 good

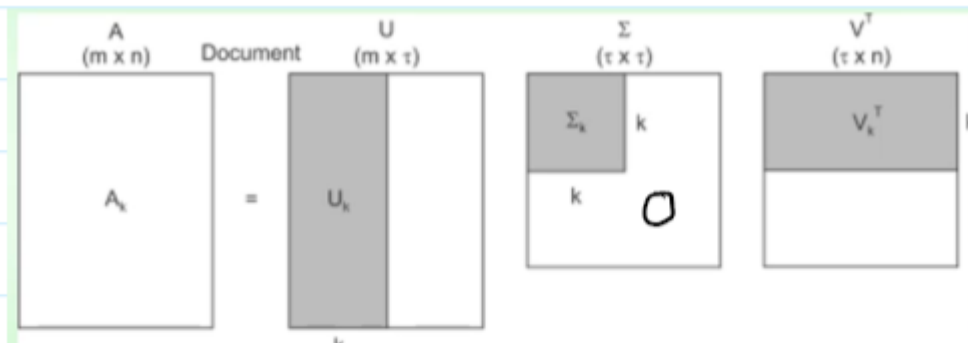
\rightarrow 어떤 방향이 중요한지 V (더 적은 수의 basis vector로 구할 수 \circ)

Eigenvalue Decomposition

$$A = P P P^T = P D P^T$$

$$= (v_1 v_2 \dots) \begin{bmatrix} \lambda_1 & & 0 \\ & \lambda_2 & \\ 0 & & \ddots \end{bmatrix} (v_1 v_2 \dots)^T$$

SVD

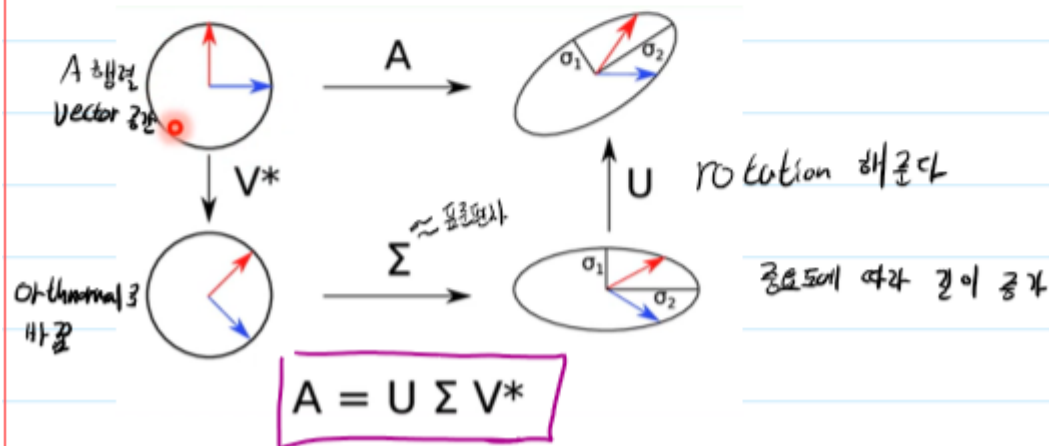


U : A 의 column space의 Bases (rank k)

V : A 의 row space의 Bases (rank k)

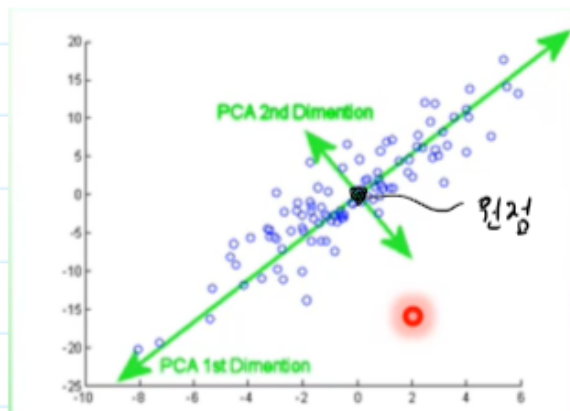
Σ : $A^T A$ 의 Singular value $\Rightarrow \sqrt{A^T A}$ 의 eigen value

U, V 는 eigen vector로 구함 \Rightarrow orthonormal 임 \triangleleft 기하학으로 Rotation



Principal Component Analysis : vector를 가장 잘 표현하는 방법

A는 전체 벡터의 평균에서
A에서 빼낸 값



- Covariance matrix : $R = A^T A$
(vector는 반드시 원점 포함)
- singular value가 클수록 더 많은 dominant가 basis vector이다.

ex) Eigen face (34분경)

사진 pixel을 인렬로 4열 \rightarrow PCA

\hookrightarrow 20x20 사진 \rightarrow $\left\{ \begin{array}{l} \text{각 사진의 데이터 100개씩} \\ \text{사진들 400} \end{array} \right\}$

\Rightarrow Average 구해서 빼 & $A^T A$ (covariance matrix 만들어)

\Rightarrow Eigen Value & Eigen Vector로 SVD 해

\hookrightarrow U와 V 만들 \Rightarrow Eigen face 만들
column row

장점 : 몇개의 특징만 추출 (basis로 추출)

Ex) face reconstruction (synthesis)

□ $N < 1024$

$$\boxed{A^T} \boxed{A} = \boxed{}$$

□ $N > 1024$

$$\boxed{A^T} \boxed{A} = \boxed{}$$

$$\text{Face Image} - \mathbf{M} = \mathbf{c}_1 \text{Feature}_1 + \mathbf{c}_2 \text{Feature}_2 + \mathbf{c}_3 \text{Feature}_3 + \dots + \mathbf{c}_n \text{Feature}_n$$

$$\mathbf{c}_k = \left[\text{Face Image} - \mathbf{M} \right] \bullet \text{Feature}_k$$

$$\text{Face Image} \approx \mathbf{c}_1 \text{Feature}_1 + \mathbf{c}_2 \text{Feature}_2 + \mathbf{c}_3 \text{Feature}_3 + \dots + \mathbf{c}_n \text{Feature}_n + \mathbf{M}$$

Project 1 Report

□ ppt presentation within 20 pages

- Collected images, eigenfaces
- Synthesis of face images using your eigenfaces
- Experiments of face recognition
 - How to compare with the coefficients of faces
- Analysis and conclusion

□ Due:

- 1월 7일 (Thu) 22:00
- Submission to LMS with pdf file

과정

- Colab 라이브러리에 있는 fetch_lfw_people에 사진을 모았다.
- 이 사진을 people에 넣었다.
- People이라는 데이터는 총 13233개의 사람이 있었고 87×65의 구조로 되어있었다.



```
from sklearn.datasets import fetch_lfw_people
import matplotlib.pyplot as plt
import matplotlib
import numpy as np

people=fetch_lfw_people(min_faces_per_person=1, resize=0.7,color=False)
#1명당 가지고 있는 최소한의 서로 다른 사진 수, 사진 비율, 흑백 여부
print(people.images)
len(people.images)
print(people.images.shape)
print(people.images[0].shape)
print(len(people.images))
type(people.images)
#shape=>a,b,c      b*c 행렬이 a개 존재
```

Downloading LFW metadata: <https://ndownloader.figshare.com/files/5976012>
Downloading LFW metadata: <https://ndownloader.figshare.com/files/5976009>
Downloading LFW metadata: <https://ndownloader.figshare.com/files/5976006>
Downloading LFW data (~200MB): <https://ndownloader.figshare.com/files/5976015>

```
[[[ 32.333332  31.666666  25.333334 ...  24.333334  29.666666  31.
    [ 36.666668  36.333332  27.333334 ...  22.666666  27.
    [ 37.666668  35.333332  29.333334 ...  25.333334  30.666666  35.
    ...
    [ 94.333336  56.
    [ 39.
    [ 28.333334  28.333334  27.
    ...  15.
    ...  12.666667]]

[[155.666667 161.
    [156.33333 161.33333 157.
    [151.
    ...
    [126.
    [126.666664 125.
    [129.666667 127.666664 121.333336 ... 139.33333 132.33333 129.33333 ]

[[ 76.
    [ 72.
    [ 78.666664  85.333336  94.666664 ...  68.333336  74.666664  65.333336]
    ...
    [ 20.666666  30.
    [ 78.333336 101.666664 115.
    [118.666664 123.666664 124.
    ... 173.33333 104.666664  53.333332]]
```


과정

- 옆에 그림과 같이 얼굴이 잘렸거나 얼굴을 가리고 있거나 옆모습이 보이는 데이터를 삭제하였다.
- 그 후 데이터의 윗부분과 아랫부분을 살짝 잘라서 최대한 앞술부터 이마 중간까지 보이도록 만 들어서 last_images에 담았다.
- Last_images에 있는 데이터 중에서 1200개만 뽑아서 이미지 크기를 32×32로 바꿔서 last_img에 넣었다.



```
img_arr=images_arr.copy()
del img_arr[91]
del img_arr[291]
del img_arr[297]
del img_arr[714]
del img_arr[729]
del img_arr[1172]
del img_arr[1176]
del img_arr[1180]
```

#이미지 자르기

```
import numpy as np
import cv2
from google.colab.patches import cv2_imshow

last_images=[]

for i in range(len(img_arr)):
    dst=img_arr[i].copy()
    dst=img_arr[i][18:80,0:65]
    last_images.append(dst)
```

```
imgs_arr=last_images[0:1200]

last_img=[]
for i in range(len(imgs_arr)):
    a=cv2.resize(imgs_arr[i],(32,32))
    last_img.append(a)
```

과정

- last_img에 있는 배열의 평균을 구해서 avg라고 명명했다.
- last_img에 데이터에 avg를 뺀 값에 뭐든 원소가 128로 되어있는 32×32 행렬을 더하여 그림을 그렸다.
- last_img에 데이터의 개수가 1024보다 많으므로 last_img에 있는 32×32 데이터를 row vector로 만들어서 A에 저장하였다.
- A는 1200×1024꼴로 되어있다.



평균 이미지(avg)

#mean vector 구해보자

```
avg=last_img[0]
for i in range(1,len(last_img)):
    avg=avg+last_img[i]
avg=avg/len(last_img)
```

#F-Mean vector--img_A

import copy

```
img_A=copy.copy(last_img)
for i in range(len(img_A)):
    img_A[i]=img_A[i]-avg+128*np.ones((32,32))
img_A=np.array(img_A)
```

```
plt.figure(figsize=(5,5))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(img_A[i],cmap='gray')
plt.show()
```



#1200*1024 꼴로 바꾸기

```
A=[]
for i in range(0,len(img_A)):
    a=img_A[i].reshape(1,1024)
    A.append(a[0])
A=np.array(A)
A.shape
```

(1200, 1024)

과정

- A의 행렬은 1200×1024 구조($N > 1024$)이므로 correlation matrix를 만들 때 A의 전치행렬과 A를 내적하여($A^T \cdot A$) 구하였다. 그리고 이 correlation matrix를 cor_arr 이라고 명칭했다.
- cor_arr을 가지고 eigen value와 eigen vector를 구하였다. 이때 numpy에 있는 linalg 라이브러리를 사용하였다.
- eigen value의 최솟값은 0보다 크다. (모두 양수이다.)
- Eigen value중에서 상위 4%(40개)만 뽑았다.
- 그리고 4%에 해당하는 eigen value의 index를 eigen value가 큰 순서대로 num_arr에 넣었다.

```
#cor_matrix 구해  
AT=np.transpose(A)  
cor_arr=np.dot(AT,A)
```

```
import numpy.linalg as la
```

```
eigvals, eigvecs = la.eig(cor_arr)  
print(min(eigvals))
```

```
65.77857738910069
```

>0

=> positive

```
#40개만 고르기  
sort_eigvals=sorted(eigvals)  
choose_eigvals=[]  
for i in range(len(sort_eigvals)-40, len(sort_eigvals)):  
    choose_eigvals.append(sort_eigvals[i])  
print(choose_eigvals)  
min(choose_eigvals)
```

```
[6232628.741219328, 6336133.714203643, 6605813.095803877, 6807305.914539728, 70262  
6232628.741219328
```


과정

- linalg 라이브러리를 사용하여 U , σ , V^T 를 구했다.
- correlation matrix를 구할때 $A \cdot A^T$ 로 구한 것이 아니라 $A^T \cdot A$ 로 구했으므로 eigen face는 V^T 에 존재하는 vector라고 판단했다.
- Eigen value에서 큰 값을 가진 것의 index를 순서대로 num_arr에 넣어놨었다. 그 index에 맞는 V^T 에 있는 vector들을 골라서 eig_face에 넣었다.
- eig_face에 있는 vector들을 32×32 꼴로 바꾸고 아래와 같이 40개의 그림을 그렸다.
(큰 것-> 작은 것)

#Eigen Face 는 Vt vector -큰거 40개만 모아서 뽑아서 그려봤다.

```
U, S, Vt = la.svd(A)
```

```
eig_face=[]
```

```
for i in num_arr:
```

```
    eig_face.append(Vt[i])
```

```
plt.figure(figsize=(10,10))
```

```
for i in range(len(eig_face)):
```

```
    plt.subplot(10,10,i+1)
```

```
    plt.imshow(eig_face[i].reshape(32,32),cmap='gray')
```

```
plt.show()
```



과정

- 3개의 함수를 만들었다.
- list_sum은 리스트 내에 모든 원소를 다 더해 주는 함수이다.
- distance는 벡터의 거리를 구해주는 함수이다.
- re_cos는 2개의 값을 받는다. 이때 2개의 값은 사진의 경로여야 한다. 2개의 사진이 얼마나 유사한지 구해주는 함수이다. (cos값이 유사도)
- coeffi는 리스트를 받으면 리스트에 아까 구한 mean vector를 뺀 후 eigen face와 내적을 통하여 각각의 coefficient를 구한다.

```
#리스트 원소 다 더해주는 함수
def list_sum(a):
    sum=0
    for i in range(len(a)):
        sum+=a[i]
    return sum
```

```
import math
def distance(a):
    sum=0
    for i in range(len(a)):
        sum+=(a[i]**2)
    sq=math.sqrt(sum)
    return sq
```

#사이트 2개를 주고 cos 구하기

```
def re_cos(a1,a2):
    coef1=[]
    coef2=[]

    a1=cv2.imread(a1, cv2.IMREAD_GRAYSCALE)
    a1=cv2.resize(a1,(32,32))
    a1=a1.reshape(1024)

    for i in range(len(eig_face)):
        D=list_sum(a1*eig_face[i])
        coef1.append(D)

    a2=cv2.imread(a2, cv2.IMREAD_GRAYSCALE)
    a2=cv2.resize(a2,(32,32))
    a2=a2.reshape(1024)

    for i in range(len(eig_face)):
        D=list_sum(a2*eig_face[i])
        coef2.append(D)

    coef1=np.array(coef1)
    coef2=np.array(coef2)
    return (list_sum(coef1*coef2)/distance(coef1))/distance(coef2)
```

```
def coeffi(a):
    coef1=[]
    a1=cv2.imread(a, cv2.IMREAD_GRAYSCALE)
    a1=cv2.resize(a1,(32,32))
    a1=a1-avg
    a1=a1.reshape(1024)

    for i in range(len(eig_face)):
        D=list_sum(a1*eig_face[i])
        coef1.append(D)
    print(coef1)
```

과정

- Eigen face의 값이 올바른지 파악하기 위해 원본 이미지와 eigen face로 복원한 이미지를 비교하였다.
- Eigen face를 40개로 잡고 복원해보니 복원한 이미지가 너무 흐릿하여 임의로 eigen face를 500개로 잡고 원본 데이터와 비교를 해보았다.
- 오른쪽 그림을 보면 꽤 잘 복원됨을 알 수가 있다.

원본
이미지



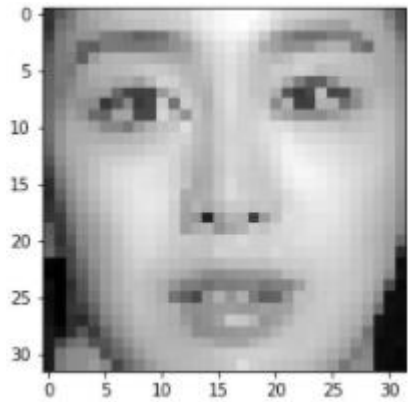
복원한
이미지



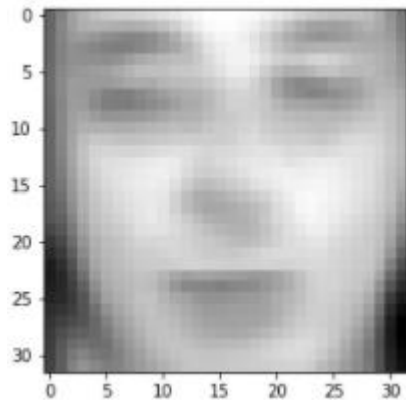
- 옆에 그림과 같이 총 10명의 사람 얼굴 사진 각각 5개씩 모았다.
(위에 순서대로 김태희, 송혜교, 박민영, 전지현, 양세찬, 양세형, 원빈, 장동건, 정우성, 조인성)
- 각 사람에 사진에 1,2,3,4,5 라고 번호를 붙였다.
- 10개의 리스트를 만들고 각 리스트에 사람의 얼굴 5개씩 넣었다.
- 데이터를 모을 때 2가지 특이한 것을 넣었다.
 1. 양세찬3과 양세형2는 어렸을 때 사진을 가지고 왔다.
 2. 양세형1은 수염을 그리고 있다.
 3. 양세찬과 양세형은 형제이다.
 4. 원빈2는 얼굴 양쪽에 멍암이 다르다.

결과와 분석

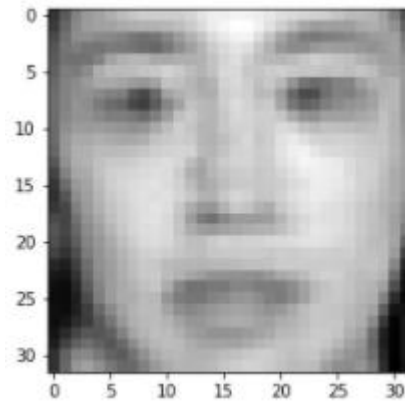
- Eigen face 몇 개를 선택했는지에 따라 그림을 합쳤을 때 어떻게 변하는지 보았다.
- 원본은 원래 이미지를 32*32로 resize한 이미지이다.
- Eigen face가 500개정도 있을 때는 이목구비가 보이고 1000개가 있을 때는 거의 원본데이터와 동일하게 나왔다.



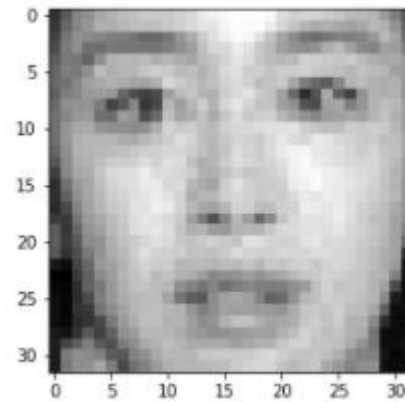
원본 (김태희4)



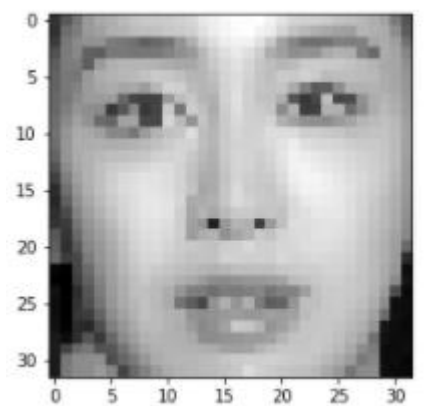
Eigen face
=> 40개



Eigen face
=> 200개



Eigen face
=> 500개



Eigen face
=> 1000개

결과와 분석

- Eigen face 40개를 가지고 동일한 사람의 사진끼리 비교를 해 보았다.
- 0.92를 동일 인물 기준으로 두면 거의 대부분의 값이 동일 인물이라고 파악 가능하다.
- 그러나 원빈2와 장동건5는 정확도가 낮았다.(수염의 유무나 어렸을 때와 어른 될 때 얼굴 차이 보다 얼굴의 명암이 더 정확도가 낮다)
- 수염의 유무는 유사도에 크게 영향을 미치지 않았다.
- 어렸을 때 외모와 어른이었을 때의 외모는 유사도에 크게 영향을 미치지 않았다.
- 동일한 이미지는 유사도가 1.0이 나온다.

동일한 이미지 유사도

e = '/content/drive/MyDrive/SVD 얼굴 인식/김태희/4.JPG'

re_cos(e,e)

1.0

수염 유무
어른vs아이
얼굴 명암

	1vs2	1vs3	1vs4	1vs5	2vs3	2vs4	2vs5	3vs4	3vs5	4vs5	평균
김태희	0.979384	0.968258	0.971092	0.966398	0.971330	0.989488	0.978650	0.956246	0.945578	0.980749	0.970717
송혜교	0.970149	0.932036	0.976267	0.957706	0.962914	0.963381	0.937832	0.948396	0.922626	0.955187	0.952649
박민영	0.946874	0.944805	0.922592	0.962237	0.956300	0.971523	0.980155	0.906742	0.976732	0.940905	0.950886
전지현	0.953018	0.986292	0.965432	0.962624	0.962553	0.943340	0.909356	0.966874	0.964283	0.956272	0.957004
양세찬	0.947152	0.967844	0.964240	0.955097	0.947078	0.976892	0.982028	0.972362	0.966569	0.989420	0.966868
양세형	0.976737	0.979499	0.986902	0.983935	0.973873	0.977629	0.979032	0.982958	0.982526	0.990931	0.981402
원빈	0.931841	0.993996	0.950746	0.964619	0.933531	0.843177	0.850706	0.949369	0.957098	0.973324	0.934841
장동건	0.944250	0.940511	0.926701	0.943627	0.939202	0.961749	0.880750	0.951542	0.908842	0.896842	0.929402
정우성	0.954736	0.938655	0.974821	0.950004	0.959360	0.972171	0.970676	0.937205	0.918726	0.976631	0.955298
조인성	0.974878	0.976024	0.964750	0.988593	0.948564	0.944532	0.972719	0.952933	0.966036	0.965308	0.965434

결과와 분석

- 같은 인물일 경우 어떤 eigen face가 영향을 많이 미치는지 알기 위해서 표준편차를 구해 보았다. (표준 편차가 크다는 것은 동일 인물을 다르다고 판단하게 영향을 미친다.)
- 대체적으로 eigen value가 클수록 표준편차가 큰 경향이 있었다. (eigen value가 크다는 것은 이미지를 판단하는데 주요한 요인이다.)
- 특히 40개의 eigen face중에서 eigen value가 가장 큰 것과 2번째로 큰 것에 해당하는 eigen face가 표준편차가 컸다.

	1	2	3	4	5	6	7	8	9	10
김태희	1463.50	418.38	240.73	3.99	119.94	16.74	91.45	69.98	6.18	54.35
송혜교	354.45	660.60	22.96	265.41	80.17	282.79	201.95	43.90	93.38	4.42
박민영	929.37	88.94	106.30	11.38	35.37	220.43	69.43	81.68	103.79	50.50
전지현	251.99	896.76	16.86	320.91	198.36	109.91	102.95	176.88	40.23	107.54
양세찬	417.91	284.64	182.65	140.63	38.71	47.68	59.07	55.21	114.45	104.89
양세형	148.01	294.53	4.55	47.78	53.69	37.75	114.59	155.69	66.59	40.35
원빈	664.17	771.46	44.17	225.55	242.23	73.30	103.54	71.28	199.69	259.23
장동건	422.48	408.93	798.17	633.23	167.58	384.19	168.12	472.10	206.76	104.10
정우성	430.00	566.58	72.68	80.25	100.06	102.27	162.34	91.51	176.17	102.54
조인성	19.48	83.00	110.00	107.25	70.12	252.47	88.13	353.40	84.28	51.01

Eigen value 큼

Eigen value 작음

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
김태희	1463.50	418.38	240.73	3.99	119.94	16.74	91.45	69.98	6.18	54.35	63.88	116.15	60.67	115.27	147.14	28.07	23.43	139.43	149.03	66.84	40.74	127.01	155.49	23.18	130.18	16.83	87.24	128.99	72.18	7.48	64.62	63.59	51.62	32.98	121.92	98.05	3.29	5.90	21.17	60.91
송혜교	354.45	660.60	122.96	265.41	80.17	282.79	201.95	43.90	93.38	4.42	160.96	255.35	158.52	97.62	157.14	86.85	121.76	92.73	127.27	126.43	67.12	125.61	58.44	97.75	124.27	56.69	206.04	61.47	72.33	130.38	43.75	98.48	22.81	8.78	7.53	93.39	123.59	35.26	40.14	45.54
박민영	929.37	88.94	106.30	11.38	35.37	220.43	69.43	81.68	103.79	50.50	3.43	13.06	215.52	33.65	91.80	86.19	181.93	74.50	17.61	12.99	115.73	153.72	56.37	62.09	24.03	46.73	20.96	30.67	18.46	32.11	19.34	13.20	39.27	34.45	73.13	20.10	46.38	24.13	4.17	15.32
전지현	251.99	896.76	116.86	320.91	198.36	109.91	102.95	176.88	40.23	107.54	86.50	45.95	0.72	62.95	197.86	21.93	244.56	27.63	101.38	117.63	31.74	192.69	136.21	173.22	38.27	137.05	8.77	5.66	123.34	146.07	18.01	62.21	61.57	50.67	34.71	133.78	84.72	26.87	73.29	90.99
양세찬	417.91	284.64	182.65	140.63	38.71	47.68	59.07	55.21	114.45	104.89	244.16	81.90	63.63	58.96	44.96	9.19	161.19	44.59	71.29	73.62	177.15	55.93	18.51	83.84	9.05	56.55	162.48	67.00	92.76	92.87	114.35	0.32	58.68	3.74	72.53	28.40	56.45	117.69	22.28	19.19
양세형	148.01	294.53	4.55	47.78	53.69	37.75	114.59	155.69	66.59	40.35	57.63	105.10	8.28	5.74	49.86	19.01	58.60	77.64	3.15	17.41	30.17	11.29	120.63	17.12	94.85	19.20	11.28	8.93	108.07	61.29	22.37	53.28	36.03	97.61	82.25	132.22	49.84	180.64	35.25	35.70
원빈	664.17	771.46	244.17	225.55	242.23	73.30	103.54	71.28	199.69	259.23	90.73	98.14	176.64	21.03	59.85	114.54	6.40	130.67	85.66	136.48	72.00	156.69	183.76	9.60	32.63	11.41	159.01	85.50	185.99	76.17	12.80	18.74	19.24	51.43	7.55	87.19	70.28	67.70	28.46	68.39
장동건	422.48	408.93	798.17	633.23	167.58	384.19	168.12	472.10	206.76	104.10	72.25	57.51	77.73	362.62	104.93	304.75	369.11	219.10	238.83	18.83	337.69	65.05	104.52	111.01	13.20	56.02	107.65	136.33	31.23	15.79	45.34	116.58	3.54	87.97	16.42	59.31	138.92	85.58	41.35	43.67
정우성	430.00	566.58	372.68	80.25	100.06	102.27	162.34	91.51	176.17	102.54	38.82	6.09	28.43	135.56	17.10	37.59	126.02	114.28	118.77	126.72	2.63	82.73	45.27	161.61	61.43	184.04	7.00	15.80	94.65	23.27	13.18	44.12	77.64	43.70	79.65	5.47	60.50	72.47	104.95	23.89
조인성	19.48	83.00	110.00	107.25	70.12	252.47	88.13	353.40	84.28	51.01	78.28	50.51	85.79	71.47	5.13	42.90	91.83	82.67	11.99	37.46	234.85	74.10	18.36	159.66	87.97	25.53	18.59	66.81	59.85	95.02	128.79	11.48	4.23	31.88	17.98	108.71	101.43	35.03	4.34	49.56

결과와 분석

- 아래 그림은 김태희와 송혜교를 비교해 보았다. 그런데 김태희와 송혜교의 유사도도 0.92가 넘게 나왔다.
- K1 : S2는 김태희1 사진과 송혜교2 사진을 비교한 것이다.
- 무엇을 비교하든 다 이렇게 높은 값이 나오는지 궁금해서 고양이와 김태희를 비교해 보았다.
- 옆에 그림은 고양이와 김태희 사진을 비교한 것이다.
- 김태희3과 송혜교5와의 유사도와 고양이와 김태희의 유사도가 크게 다르지 않았다.
- 맨 아래 사진은 양세형과 양세찬의 닮은 정도를 비교한 것이다.
- 형제의 사진이라고 닮은 정도가 높지 않았다. (평균 94.7%)

```
a='/content/drive/MyDrive/train/cat.0.jpg'  
b='/content/drive/MyDrive/SVD 얼굴 인식/김태희/4.JPG'  
re_cos(a,b)
```

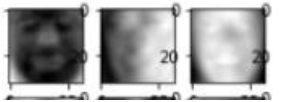
0.8735300989747867

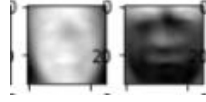
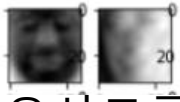

고양이와 김태희의 유사도

k1 : s1	k1 : s2	k1 : s3	k1 : s4	k1 : s5	k2 : s1	k2 : s2	k2 : s3	k2 : s4	k2 : s5	k3 : s1	k3 : s2	k3 : s3	k3 : s4	k3 : s5	k4 : s1	k4 : s2	k4 : s3	k4 : s4	k4 : s5	k5 : s1	k5 : s2	k5 : s3	k5 : s4	k5 : s5
0.964613	0.954514	0.911408	0.968416	0.920187	0.984808	0.970658	0.926026	0.97959	0.932388	0.954247	0.966849	0.912777	0.954955	0.884794	0.987638	0.9738	0.930489	0.975102	0.950977	0.985256	0.961582	0.914376	0.957762	0.941973
제1:형1	제1:형2	제1:형3	제1:형4	제1:형5	제2:형1	제2:형2	제2:형3	제2:형4	제2:형5	제3:형1	제3:형2	제3:형3	제3:형4	제3:형5	제4:형1	제4:형2	제4:형3	제4:형4	제4:형5	제5:형1	제5:형2	제5:형3	제5:형4	제5:형5
0.957664	0.94377	0.923867	0.948769	0.936336	0.90997	0.963621	0.902038	0.954284	0.948104	0.957233	0.954497	0.927487	0.952217	0.936015	0.967875	0.971122	0.950075	0.976735	0.967125	0.970249	0.927914	0.952029	0.951135	0.935304


결과와 분석

- 인물의 사진끼리 유사도가 너무 높아서 각 계수마다 가중치를 주기로 했다.
- 동일한 인물의 다른 이미지를 비교했을 때 각 eigen face에 해당하는 계수의 차이가 가장 큰 것을 찾았다. (계수의 차이가 크다는 것은 동일 인물을 다른 인물이라고 파악하게 하는 주요 요인이라고 생각했다.)

- 옆에 eigen face가  계수 차이가 컸다. (eigen value가 가장 큰 3개에 해당하는 eigen face)
- 반대로 다른 인물의 이미지를 가지고 위의 과정을 반복했다. (여기서 계수가 크다는 것은 다른 인물을 다른 인물로 파악하게 하는 주요 요인이라 여겼다.)

-  가 계수 차이가 컸다. (eigen value가 3번째로 큰 것과 4번째로 큰 것)
- 그래서  에 해당하는 계수는 절반으로  에 해당하는 계수는 2배로 늘리고 다시 유사도를 파악했다.

결과와 분석

- 가중치를 주었음에도 동일 인물에 관한 유사도는 거의 향상되지 않았다.
- 오히려 떨어진 확률 값도 많았다.  Eigen face의 2배를 한 것이 동일 인물의 유사도를 파악하는데 방해한다고 판단했다.
- 다른 인물과의 유사도에도 거의 변화가 없었다.
- 일부는 eigen face를 늘리고 일부는 감소시켜서 유사도에 변화가 없다고 생각해서 외에도 동일 인물을 다르게 파악하는데 영향을 많이 미치는 eigen face만 가중치 부여 등 다른 방식을 해보았다.
- 그러나 이러한 방식 모두 유사도에 큰 영향을 미치지 못했다.

	1vs2	1vs3	1vs4	1vs5	2vs3	2vs4	2vs5	3vs4	3vs5	4vs5
김태희	0.979224	0.968253	0.971108	0.966340	0.971744	0.989620	0.978992	0.956264	0.945427	0.980860
송혜교	0.970168	0.932042	0.976268	0.957713	0.962855	0.963308	0.937814	0.948322	0.922616	0.955190
박민영	0.946501	0.944720	0.922600	0.962019	0.956265	0.971664	0.980087	0.906962	0.976796	0.940657
전지현	0.952848	0.986708	0.965265	0.962780	0.962782	0.943430	0.909541	0.966544	0.964274	0.956379
양세찬	0.947060	0.967885	0.964305	0.954807	0.947084	0.976996	0.982064	0.972287	0.966072	0.989263
양세형	0.976756	0.979632	0.986802	0.983927	0.973953	0.977470	0.978971	0.982812	0.982301	0.990837
원빈	0.931891	0.994101	0.950643	0.965065	0.933684	0.842946	0.851277	0.949139	0.957281	0.973669
장동건	0.944111	0.940474	0.926455	0.943350	0.939237	0.961552	0.880499	0.951217	0.908522	0.896643
정우성	0.954433	0.938406	0.974674	0.949705	0.959557	0.972107	0.970635	0.937271	0.918565	0.976719
조인성	0.974896	0.975976	0.964693	0.988360	0.948480	0.944454	0.972439	0.953113	0.965964	0.965153

	k1 : s1	k1 : s2	k1 : s3	k1 : s4	k1 : s5	k2 : s1	k2 : s2	k2 : s3	k2 : s4	k2 : s5	k3 : s1	k3 : s2	k3 : s3
0	0.964332	0.954562	0.911306	0.968383	0.920171	0.984744	0.970722	0.926009	0.979642	0.932401	0.953919	0.9669	0.912658
	k3 : s4	k3 : s5	k4 : s1	k4 : s2	k4 : s3	k4 : s4	k4 : s5	k5 : s1	k5 : s2	k5 : s3	k5 : s4	k5 : s5	
	0.954914	0.884774	0.987451	0.973834	0.930425	0.975075	0.950968	0.98499	0.961561	0.914284	0.957633	0.941939	

결과와 분석

가장 자신의 사진과 유사도가 높은 사진의 번호를 찾아 보았다.

- 김태희1 사진은 자기 사진과 유사도가 모두 96%가 넘는다.
- 송혜교2 사진은 자기 사진과의 유사도가 1개 빼고 모두 96%가 넘는다.
- 박민영3 사진은 자기 사진과의 유사도가 1개 빼고 모두 95%가 넘는다.
- 전지현1 사진은 자기 사진과의 유사도가 모두 95%가 넘는다.
- 양세찬5 사진은 자기 사진과의 유사도가 모두 95%가 넘는다.
- 양세형2 사진은 자기 사진과의 유사도가 모두 95%가 넘는다.
- 원빈은5 사진은 자기 사진과의 유사도가 1개 빼고 모두 95%가 넘는다.
- 장동건4 사진은 자기 사진과의 유사도가 2개 빼고 모두 95%가 넘는다.
- 정우성2 사진은 자기 사진과의 유사도가 모두 95%가 넘는다.
- 조인성1 사진은 자기 사진과의 유사도가 모두 95%가 넘는다.

⇒ 그래서 각 인물의 사진을 위에 번호로 계수를 구하고 동일 인물일 확률을 95%로 정했다. 그리고 정확도를 측정하기로 하였다.

결과와 분석

- 가중치 유무로 확률 값을 구해 보았다. ($\cos\theta \times 100\%$)
- 가중치를 주지 않은 데이터와 가중치를 준 데이터 모두 장동건이 가장 높은 확률 값을 가졌다. (장동건은 자신의 이미지 포함 대부분의 사진이 자신이 아니라고 함)
- 가중치를 주면 어떤 값은 확률이 소폭 향상되고 어떤 값은 확률이 소폭 감소했다. 즉 가중치는 크게 영향을 미치지 않았다.
- 동일 인물을 95%를 기준으로 두니 대부분의 데이터가 60% 이상의 확률로 올바르게 판단했다. (모든 데이터가 찍는 확률(50%) 보다 높은 정답률을 보임)

확률 (%)		확률 (%)	
김태희	65.0	김태희	65.0
송혜교	62.5	송혜교	62.5
박민영	75.0	박민영	75.0
전지현	75.0	전지현	75.0
양세찬	62.5	양세찬	62.5
양세형	60.0	양세형	60.0
원빈	80.0	원빈	80.0
장동건	92.5	장동건	92.5
정우성	77.5	정우성	75.0
조인성	62.5	조인성	57.5

가중치(x)

가중치(o)

결과와 분석

- 장동건처럼 모든 사람이 장동건이 아니라고 판단하는 것을 방지하기 위해 동일 인물들의 사진의 계수들의 평균을 구해서 그 평균값으로 다른 인물의 이미지와 비교를 해 보았다. (임계값 0.8)

자기 판단 : 자신의 사진을 자신이라고 판단할 확률

다른 사람 : 다른 사람의 사진을 다른 사람이라고 판단할 확률

- 그러나 왼쪽 사진과 같이 송혜교는 자기 자신의 사진을 맞출 확률이 0%가 나왔다. (앞에서 발생한 장동건과 동일한 현상)
- 자기 판단을 0.8이 되도록 임계값을 데이터마다 다르게 변화를 주고 데이터를 다시 조사해보았다.
- 양세찬은 다른 사람을 자신이라고 판단할 확률이 매우 높았다.

자기 판단 다른 사람			자기 판단 다른 사람		
김태희	0.8	0.555556	김태희	0.8	0.844444
송혜교	0.0	0.977778	송혜교	0.8	0.311111
박민영	0.6	0.400000	박민영	0.8	0.400000
전지현	0.4	0.644444	전지현	0.8	0.466667
양세찬	0.6	0.600000	양세찬	0.8	0.155556
양세형	0.8	0.422222	양세형	0.8	0.866667
원빈	0.8	0.444444	원빈	0.8	0.844444
장동건	0.6	0.955556	장동건	0.8	0.311111
정우성	0.8	0.533333	정우성	0.8	0.777778
조인성	1.0	0.466667	조인성	0.8	0.622222

결론

- 이 보고서는 SVD를 사용하여 eigen value, eigen vector, eigen face를 구했으며 eigen face를 가지고 여러가지 사진을 비교하고 분석을 하였다.
- 선형 결합을 사용하여 이미지를 복원하였고 원본 이미지와 비교해 eigen face가 정확한지 파악했다.
- Eigen face에 있는 계수들을 가지고 cosine 함수를 사용하여 유사도를 파악했다.
- 표준편차를 사용하여 인물을 파악하는데 중요한 eigen face가 무엇인지 파악했다.
- 문제점
 1. 인물들끼리 유사도를 파악하니 너무 유사도가 높았다.
=> 가중치를 주었다. 그러나 크게 개선되지 않았다.
=> 동일 인물이라고 판단하는 유사도를 매우 높였다. 대부분의 인물을 60% 이상 올바르게 판단했다.
 2. 대부분의 인물 사진들(자기 자신을 포함) 다 자신이 아니라고 판단하는 경우가 발생했다.
=> 동일 인물의 계수들의 평균을 가지고 조사해 보았다. (효과가 없었다.)
=> 동일 인물을 정확히 판단할 확률을 80%로 잡게 했다. (다른 사람에 대한 이미지 판단이 부족하다.)