Mini Project # 3

Due: Oct. 15/2025

James Levi

Note: I changed everything to 0-index notation, as I am using python.

1. Loading and Exploring the Data

Get the data file `allFaces.mat` from Canvas (you can read .mat files both in Matlab and Python). Load the file:

```
load allFaces.mat
```

Face pictures of 38 people with several different room lightings are stored in this file. The size of each picture is n=192 by m=168. Each picture is reshaped as a vector of 32256 by 1 and stored as a column in the matrix named `faces`. When you load the file, you get this matrix and also a vector named `nfaces`. The vector `nfaces` is a 1 by 38 row vector. The first element of `nfaces` shows how many pictures of person #0 are stored in the faces matrix. For instance, `nfaces(0)=64`. This means that from column 0 to column 64 we have pictures of person #0 with 64 different room lightings. `nfaces(1) = 64` as well. This means that in the matrix faces from column 65 to 128 we have 64 pictures of person #1, and so on. Not all elements of `nfaces` are equal.

A. Reading first pictures of person #1 to person #36:

```
#plotting first pictures of person #0 to person #35
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

data = loadmat("allFaces.mat")
faces = data["faces"]
nfaces = data["nfaces"][0]
n, m = 192, 168

print("faces shape:", faces.shape)
print("nfaces shape:", nfaces.shape)

plt.figure(figsize=(8, 8))
for i in range(36):
    col = int(sum(nfaces[:i]))
    img = faces[:, col].reshape(n, m, order="F")
```

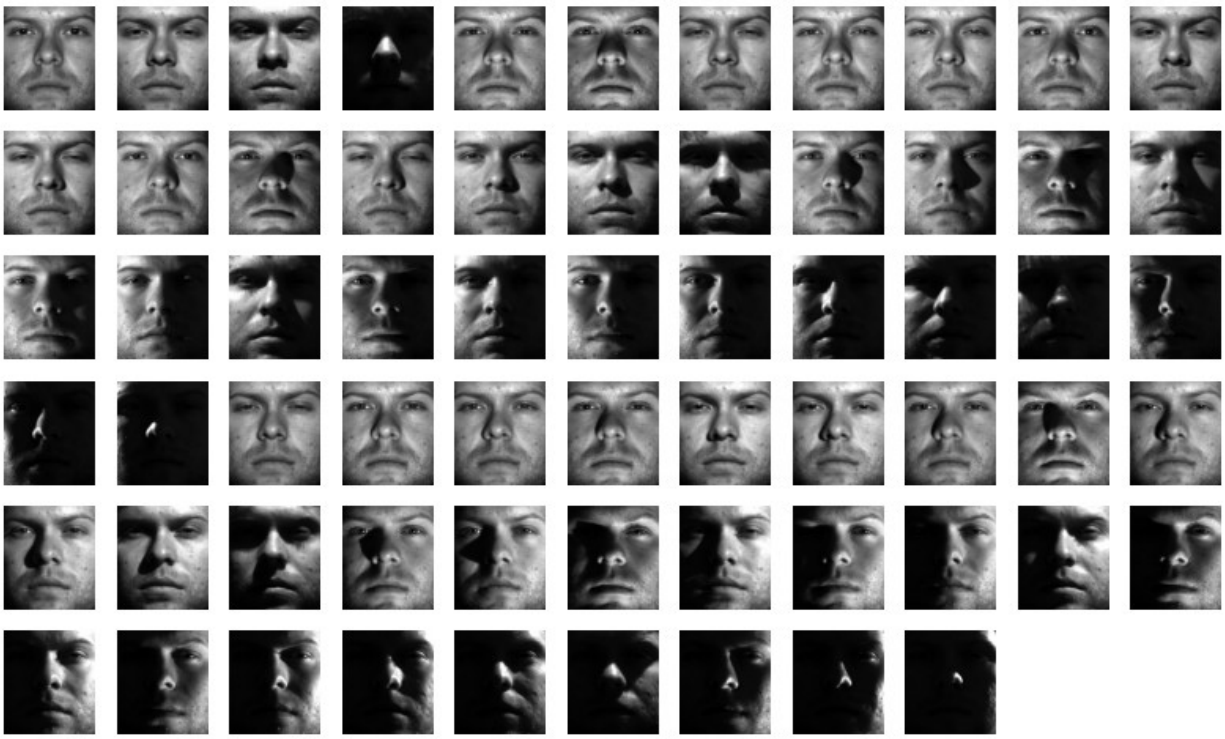
```
    plt.subplot(6, 6, i + 1)
    plt.imshow(img, cmap="gray")
    plt.axis("off")
plt.show()
```

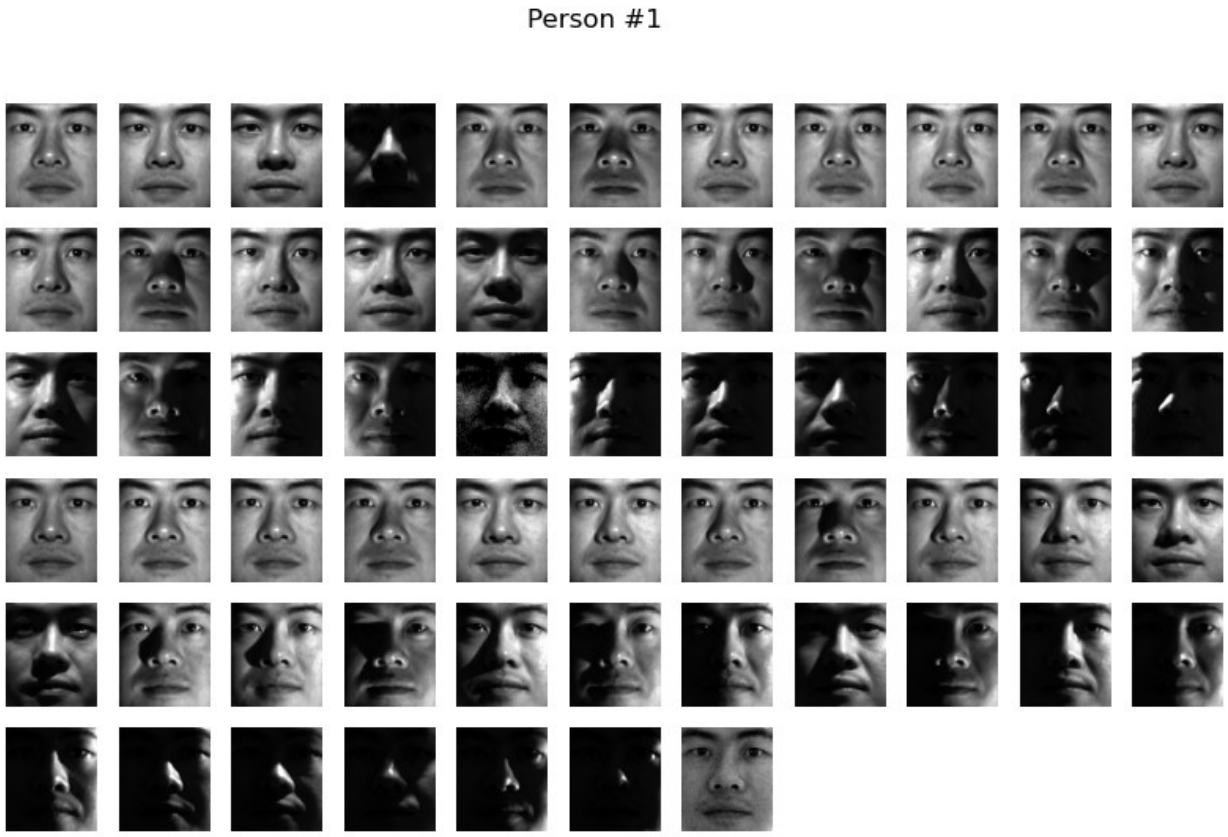
```
faces shape: (32256, 2410)
nfaces shape: (38,)
```

Reading 64 different pictures of Person #1, then Person #2...

```
#Reading 64 different pictures of Person #0, then Person #1
import numpy as np
from scipy.io import loadmat
import matplotlib.pyplot as plt

data = loadmat("allFaces.mat")
faces = data["faces"]
nfaces = data["nfaces"][0]
n, m = 192, 168

for i in range(len(nfaces)-36): #show only the first 2 out of 28
people
    start = int(sum(nfaces[:i]))
    end = int(sum(nfaces[:i + 1]))
    subset = faces[:, start:end]

    plt.figure(figsize=(10, 6))
    for j in range(subset.shape[1]):
        img = subset[:, j].reshape(n, m, order="F")
        plt.subplot(6, 11, j + 1)
        plt.imshow(img, cmap="gray")
        plt.axis("off")
    plt.suptitle(f"Person #{i}")
    plt.show()
```

Person #0





2.Average Subtracted Faces

Load all faces with all different lightings from person # 0 to person # 35. Calculate the average face picture. Subtract that average from all pictures. Store these average subtracted faces in matrix X.

```
def printFace(column, title=None):
    plt.imshow(column.reshape(192, 168, order="F"), cmap="gray")
    if title is not None:
        plt.title(title)
    plt.show()

faces_36 = faces[:,:(sum(nfaces[:36]))]

mean_face = faces_36.mean(axis =1,keepdims= True) #returns a column,
its the mean of all rows, where each row is a pixel position
X = faces_36 - mean_face

(mean_face, "MEAN FACE")

plt.figure(figsize=(8, 8))
for i in range(36):
```

```
    col = int(sum(nfaces[:i]))
    img = X[:, col].reshape(n, m, order="F")
    plt.subplot(6, 6, i + 1)
    plt.imshow(img, cmap="gray")
    plt.axis("off")
plt.suptitle("Average Subtracted Faces: First Image of Each Person (0-35)")
plt.show()
```

3. SVD and Eigenfaces

Calculate all singular values and singular vectors of X (in Matlab: `[U,S,V]=svd(X,'econ')`). Reshape these U vectors to 2-dimensional pictures of size 192 by 168. These are our eigenfaces. Plot the first 54 eigenfaces.

Carefully investigate these eigenfaces. Do you think each eigenface is trying to detect certain features?

- **U:** matrix *(pixels × r)* — each column is an eigenface.
- **S:** vector *(r,)* — importance of each eigenface.
- **V^T:** matrix *(r × images)* — how much each eigenface appears in each image.
- **r:** number of **eigenfaces (components)** kept; it equals the smaller of *(pixels, images)* and represents how many unique patterns the data contains. Together they make (X = U \ times S \times V^T).

```
U, S, VT = np.linalg.svd(X, full_matrices=False)

print("X shape:", X.shape)
print("U shape:", U.shape)
print("S shape:", S.shape)
print("VT shape:", VT.shape)

plt.figure(figsize=(12, 10))
for i in range(54):
    plt.subplot(6, 9, i + 1)
    plt.imshow(U[:, i].reshape(n, m, order="F"), cmap="gray")
    plt.axis("off")
    plt.title(f"{i+1}", fontsize=8)
plt.suptitle("First 54 Eigenfaces")
plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

```
X shape: (32256, 2282)
U shape: (32256, 2282)
S shape: (2282,)
VT shape: (2282, 2282)
```

Compute the inner product of the first and the 5th eigenfaces (do the computations with vectors, before reshaping to 2-dimensional pictures). Are these eigenfaces orthogonal? How about the 10th eigenface and 15th eigenface? Are they all orthogonal?

```
inner_product_1_5= (U[:,0]).T @ U[:,4]
print(inner_product_1_5)


inner_product_10_15= (U[:,9]).T @ U[:,14]
print(inner_product_10_15)
```

```
-3.469446951953614e-17
2.0122792321330962e-16
```

Yes, these vectors are orthogonal to eachother. As we see, their values are in the magnitude of power of -16 and -17, thus, it is essentially zero. As we know, the inner product will be zero when two values are orthogonal (90° angle between them, cosine=0)

4. Face Reconstruction Using Eigenfaces

Now let's decompose the first picture of person #36 to these eigenfaces.

If we assume that vector V is the first picture of person #36, then:

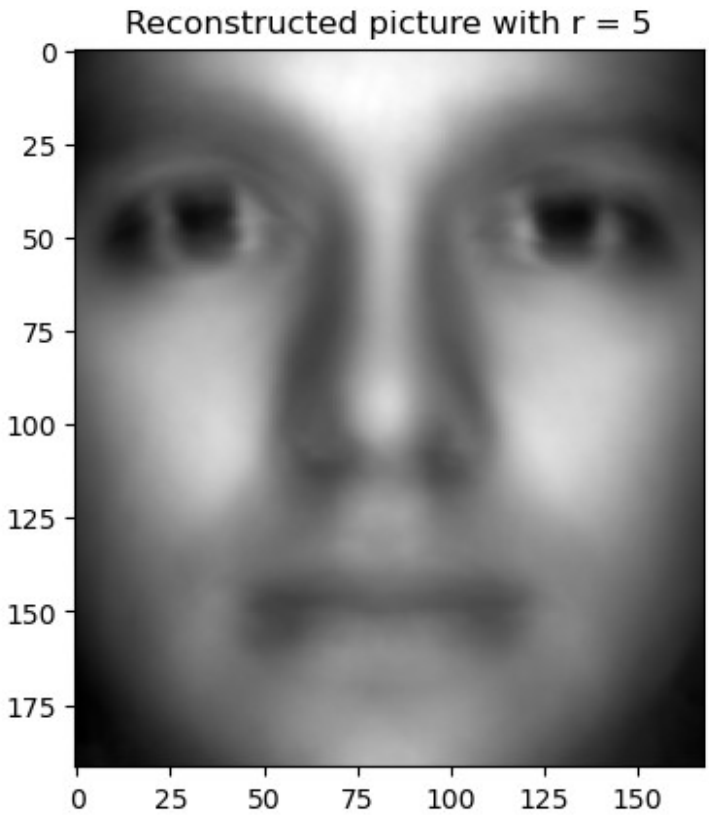
$$V \approx \sum_{i=1}^{r} \alpha_i U_i \text{ where } \alpha_i = \left(U_i\right)^T V$$

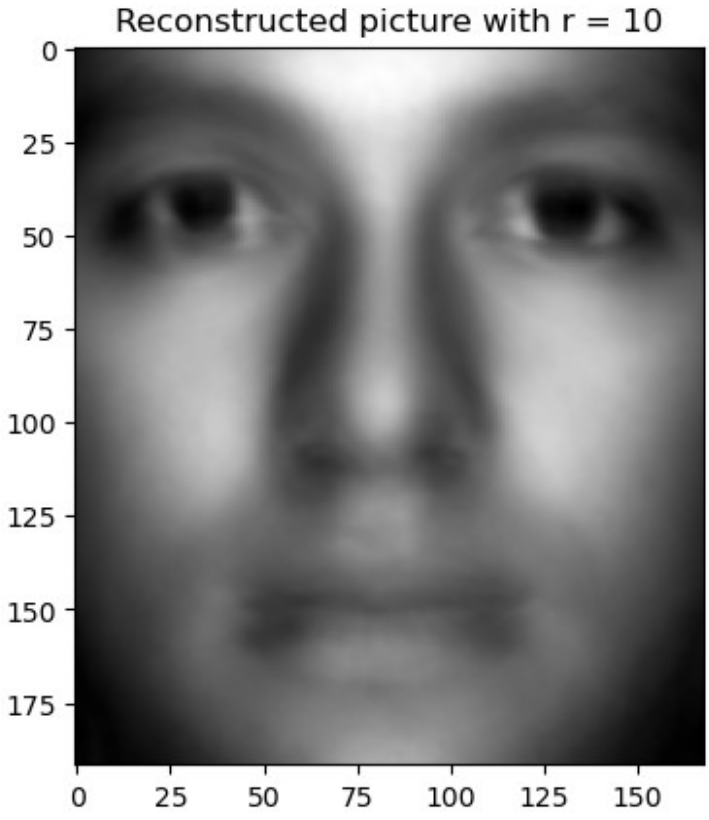
We are truncating at some r. What is the maximum value of r?

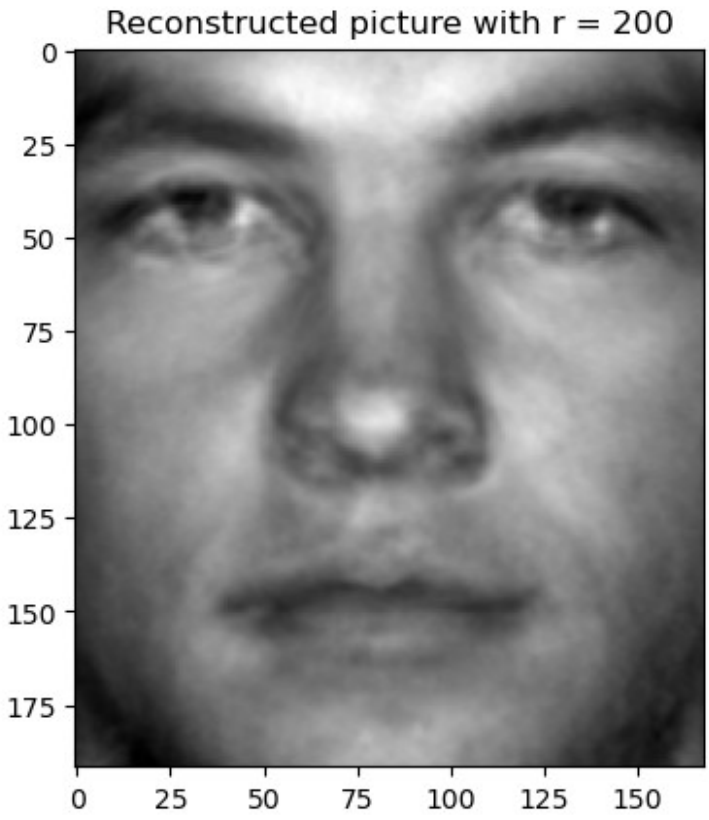
Plot the approximation of vector V as a 2-dimensional picture for r=5,10,200,800,100.

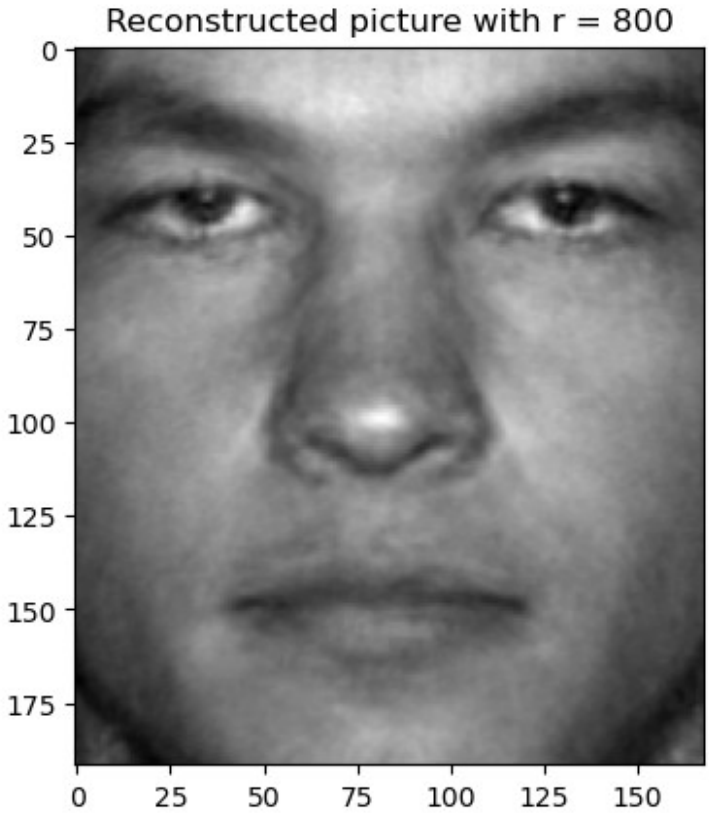
The maximumm value of r is the amount of eigen faces, which was found to be 2410. So, since we will be approximating a face as a linear combination of the eigenfaces, we must determine how many eigenfaces to use to be satisfactory.

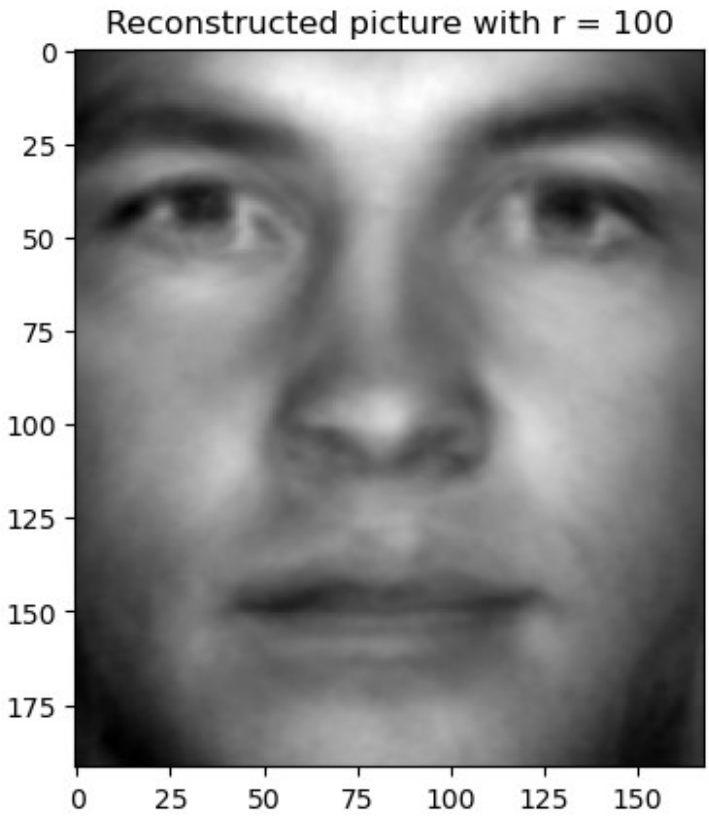
```
def reconstruct(U, centered_face, mean_face, r):
    U_sub = U[:, :r]                        # (32256, r)
    alpha = U_sub.T @ centered_face         # (r, 1)
    recon = U_sub @ alpha + mean_face       # (32256, 1)

    title = f"Reconstructed picture with r = {r}"
    printFace(recon, title=title)

index = sum(nfaces[:36])                        # start col for person 36
(0-based)
face_36 = faces[:, index:index+1]               # (32256, 1)
centered_face_36 = face_36 - mean_face          # (32256, 1)

r_values = [5, 10, 200, 800, 100]
for r in r_values:
    reconstruct(U, centered_face_36, mean_face, r)
```


Reconstructed picture with r = 5
0
25
50
75
100
125
150
175
0
25
50
75
100
125
150


Reconstructed picture with r = 10
0
25
50
75
100
125
150
175
0
25
50
75
100
125
150

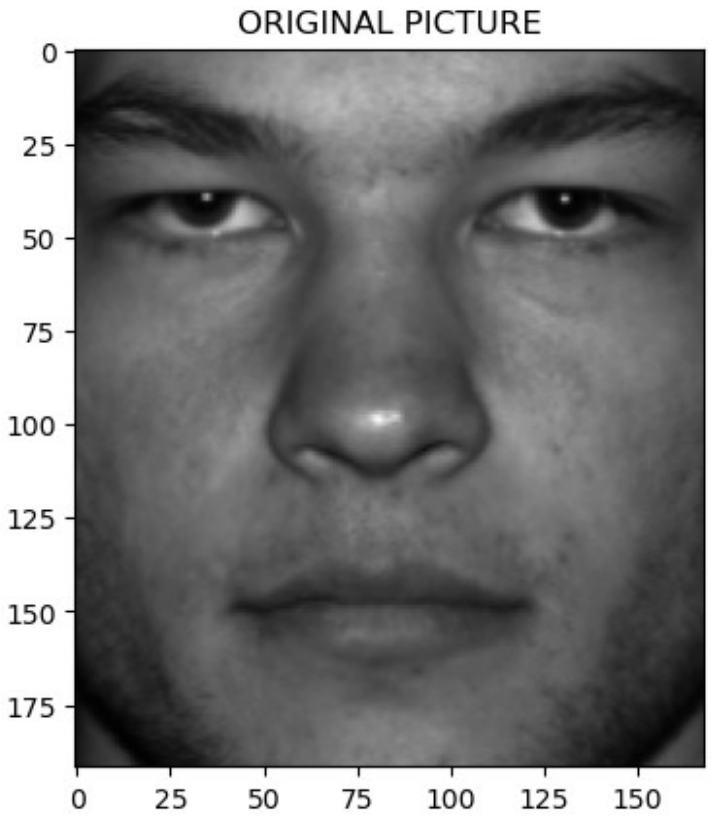

Reconstructed picture with r = 200
0
25
50
75
100
125
150
175
0
25
50
75
100
125
150


Reconstructed picture with r = 800
0
25
50
75
100
125
150
175
0
25
50
75
100
125
150




```
print("here is the original image of the face")
printFace(face_36, "ORIGINAL PICTURE")
```

```
here is the original image of the face
```



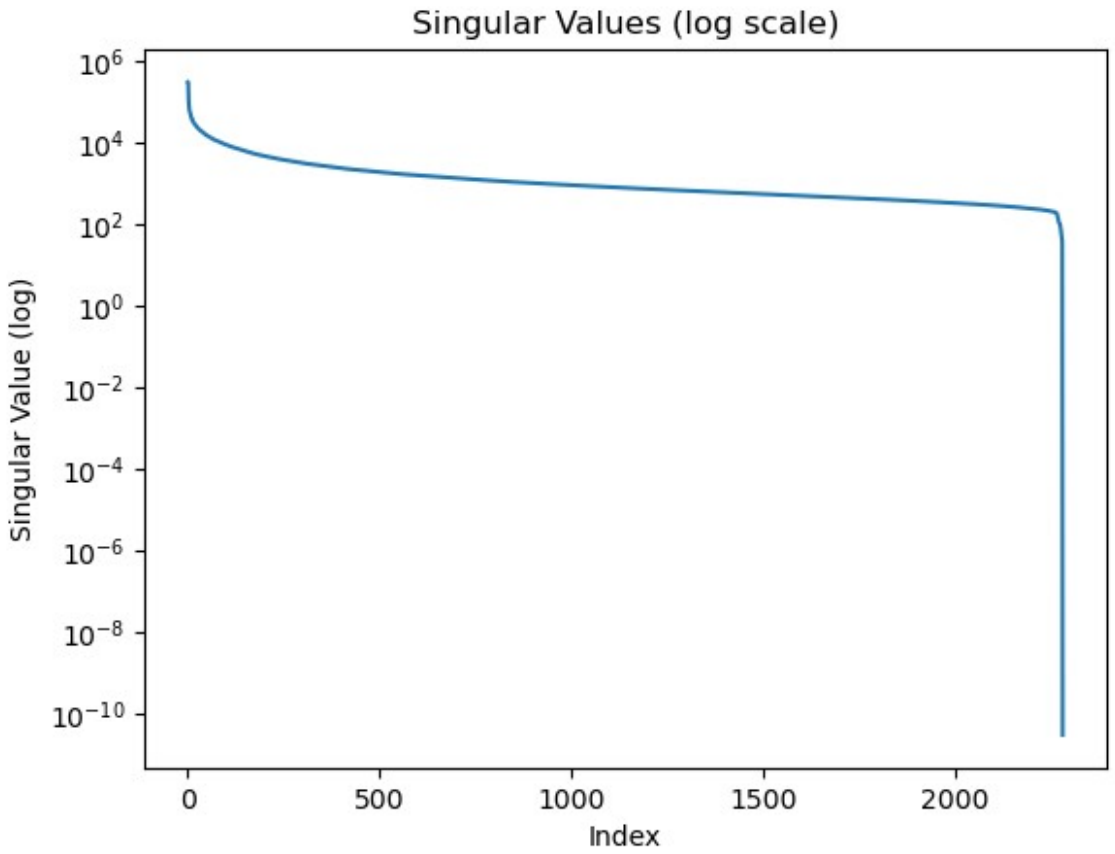
Do you think you can find a good approximation of the picture with r=100?

When comparing the original image with the approximations with r= 5, 10, 200, 800, we see that there is an obvious increase in the quality of the image. At r=100, I beleive that the appriximation is adequate, however, it is blurry, and eyes from other eigenfaces are somewhat present below the individual's eyes. Depending on the purpose of the image, it may be considered a good approzimation.

5. Singular Value Analysis

Plot singular values (in a semi-logarithmic scale, horizontal axis representing the index, vertical axis representing the value of the singular value, vertical axis is scaled logarithmically). Do you see a good point for truncation?

```
plt.semilogy(S)
plt.title("Singular Values (log scale)")
plt.xlabel("Index")
plt.ylabel("Singular Value (log)")
plt.show()
```



As we can see, truncation is very important, as the singular value has an exponential dropoff. At first, adding more singular values to the appriximation does a significant improvement, but after index 2300, there is no apparent benefit as the singular value is essentially zero. Truncation is very important as it allows us to be efficient by using the significant singular values, while not having to overwork to use the samller ones.

6. Eigenface Classification

Get all 64 different pictures of Person #1. How much of eigenface number 5 do you have in each picture of person #1? (For instance, if V_1 is the vector that represents the first picture of person #1, and $U^{(5)}$ is the 5th eigenface, $\alpha = (V_1)^T U^{(5)}$, we have α amount of 5th eigenface in the first picture of person 1).

```
def get_alphas(pid, faces, nfaces, mean_face, U):
    start = int(sum(nfaces[:pid]))
    end = start + int(nfaces[pid])
    mf = mean_face if mean_face.ndim == 2 else mean_face[:, None]
    X_p = faces[:, start:end] - mf
    alphas = U.T @ X_p          # each row = eigenface, each column =
image
```

```
    return alphas               # shape: (num_eigenfaces,
num_images_for_person)

alpha_person2 = get_alphas(1, faces, nfaces, mean_face, U)
print(alpha_person2.shape)  # (r, number_of_images_for_person2)
```

```
(2282, 62)
```

How much of each eigenface 5 do we have in every picture of person #1? How much of each eigenface 6 do we have in every picture of person #1?

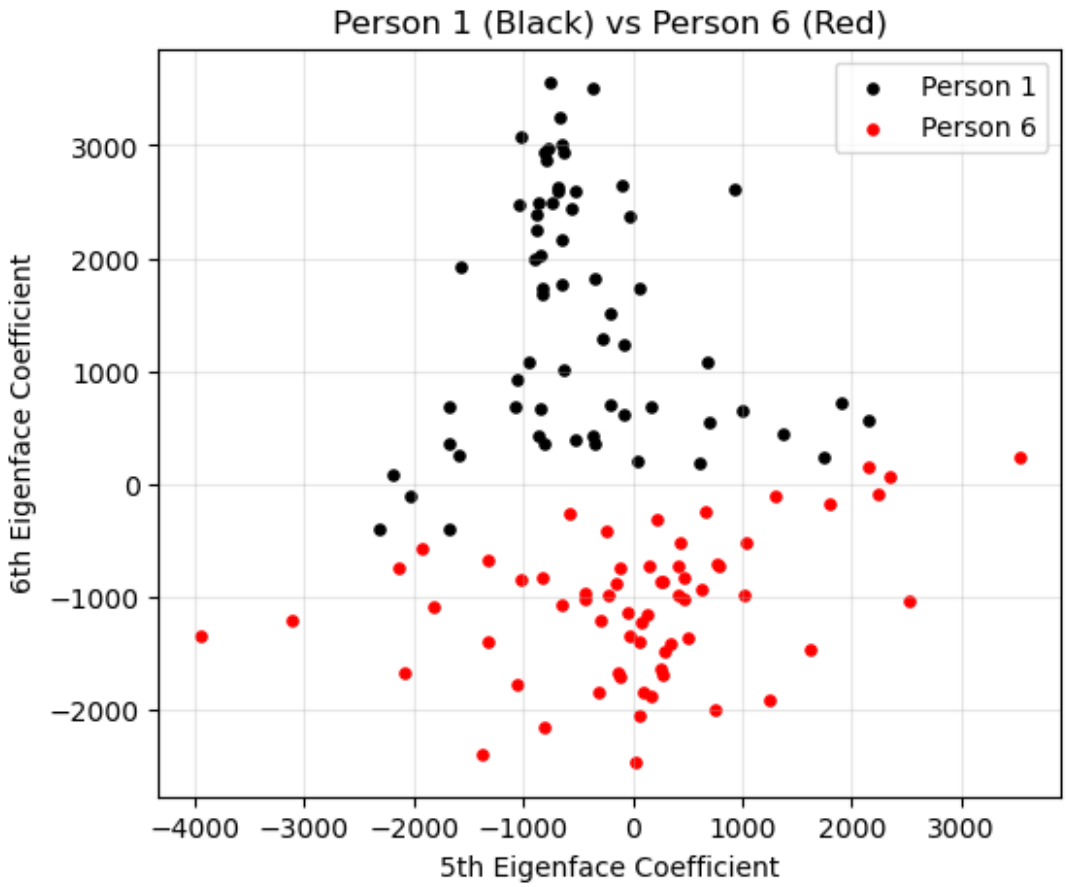
```
alpha5 = alpha_person2[4, :]
alpha6 = alpha_person2[5, :]

print("Alpha 5 (amount of 5th eigenface in each picture of person
#1):")
print(alpha5)

print("\nAlpha 6 (amount of 6th eigenface in each picture of person
#1):")
print(alpha6)
```

```
Alpha 5 (amount of 5th eigenface in each picture of person #1):
[ -557.29522605  -647.61373337  -747.45048853 -1075.11961009
  -655.24734863  -621.73192515  -795.71406033  -644.54463402
  -686.03991529  -342.11797085  -636.26343532  -519.12489143
  -198.33104868  -673.90985765 -1027.09070474 -1038.88562711
   -88.06001014  -203.55549371   601.21482347  -897.01728288
   701.84543787   675.29877989 -1576.14993772  1735.57381265
  -283.39082535  1895.55577849  -342.18113133   991.41670117
  -850.84046978 -1678.04643266  2149.23451662  1362.73306566
    43.65686496  -744.39705741  -872.39117669  -853.7456663
  -831.30237412  -661.89329747  -763.83585054  -841.37507173
  -860.85552447  -881.29819948  -804.27076146  -365.84090841
   -29.13366879 -1049.64498195  -821.39148379 -1672.7362692
   -97.46201606 -1587.73907342  -942.98467323   937.56277085
 -2317.13701809    65.3284845  -2188.00333892 -1677.19107682
  -520.59300888   171.84256411   -79.34859892 -2034.31919841
  -804.3729214   -360.81626306]

Alpha 6 (amount of 6th eigenface in each picture of person #1):
[2432.21511341 3005.77893593 3553.39287736  687.48811779 1776.81260942
 1016.65276686 2865.92603004 2162.70141378 2597.94437395 1824.97358906
 2930.71736478 2600.19572451  698.86337913 2625.43285785 3070.10940333
 2477.27948227 1230.69958356 1514.22918272  194.89738568 1999.24197829
  544.0870895  1088.01720157 1926.9940242   239.55780643 1291.42768436
  716.9699832   355.54435025  653.58019208  677.70557281  687.75874975
  571.57458044  453.06446382  197.59157021 2496.77295284 2243.11370185
 2499.55291374 1684.75673803 3246.72241893 2974.47091867 2024.36018872
  427.87495817 2381.04108737 2946.22526196 3512.97428256 2367.0898362 ]
```

```
  935.51452749 1741.84061397 -394.73970803 2645.27827323  262.63595
 1075.05439345 2608.43197068 -393.36086    1731.75761359   79.93231768
  363.09944272  395.54920466  691.6410837   614.40501587 -110.06540177
  363.95494503  436.22178209]
```

Plot a 2-dimensional graph in which the horizontal axis is the 5th eigenface and vertical axis is the 6th eigenface. In this plane mark how much of each eigenfaces 5th and 6th exists in all pictures of person 1 (black color) and also person 6 (red color). Can you use this graph to implement a good classification algorithm for face recognition? Can you do this with SVM?

```
# person 7 (0-based index = 6)
alpha_person7 = get_alphas(6, faces, nfaces, mean_face, U)
alpha5_p7 = alpha_person7[4, :]
alpha6_p7 = alpha_person7[5, :]

# Plot EF5 vs EF6 for both persons
plt.figure(figsize=(6,5))
plt.scatter(alpha5, alpha6, c='k', s=14, label='Person 1')
plt.scatter(alpha5_p7, alpha6_p7, c='r', s=14, label='Person 6')
plt.xlabel("5th Eigenface Coefficient")
plt.ylabel("6th Eigenface Coefficient")
plt.title("Person 1 (Black) vs Person 6 (Red)")
plt.legend()
plt.grid(alpha=0.3)
plt.show()
```



Yes, this graph can be used to implement a good classification algorithm for face recognition. As we can see, it clearly separates, for each person, which individual has more of a resemblance to the 5th eigenface or the 6th eigenface. An eigenface can be considered synonymous with an arbitrary facial feature. We see how the different eigenfaces show that different people have stronger resemblances to different features, just as we see with person 2 and person 7. A support vector machine can generate a hyperplane with margin that can separate this 2D space.