final_project_notebook

December 3, 2019

1 ECE 269 - Linear Algebra - Fall 2019 - Final Project - EigenFaces

EigenFaces project worked on by:-

Name :- Anirudh Swaminathan

PID :- A53316083

Email ID :- aswamina@ucsd.edu

Notebook created by Anirudh Swaminathan from ECE department majoring in Intelligent Systems, Robotics and Control for the course ECE269 Linear Algebra for Fall 2019

[1]:
```
%matplotlib inline

# for working with images
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

# randomness
import random

# matrices
import numpy as np

# eigen value computation
from scipy.linalg import eigh

# for rotating images
from scipy import ndimage as ndi
```

[2]:
```
# dataset directory specified here
data_dir = "../datasets/"
```

[3]:
```
# data stuff here
neutral_imgs = []
smiling_imgs = []
mean_img = None
dataset = []
image_size = None
eigen_faces = None

# set the random numpy seed to reproduce results between runs
np.random.seed(42)
random.seed(7)
```

1.0.1 Load the data

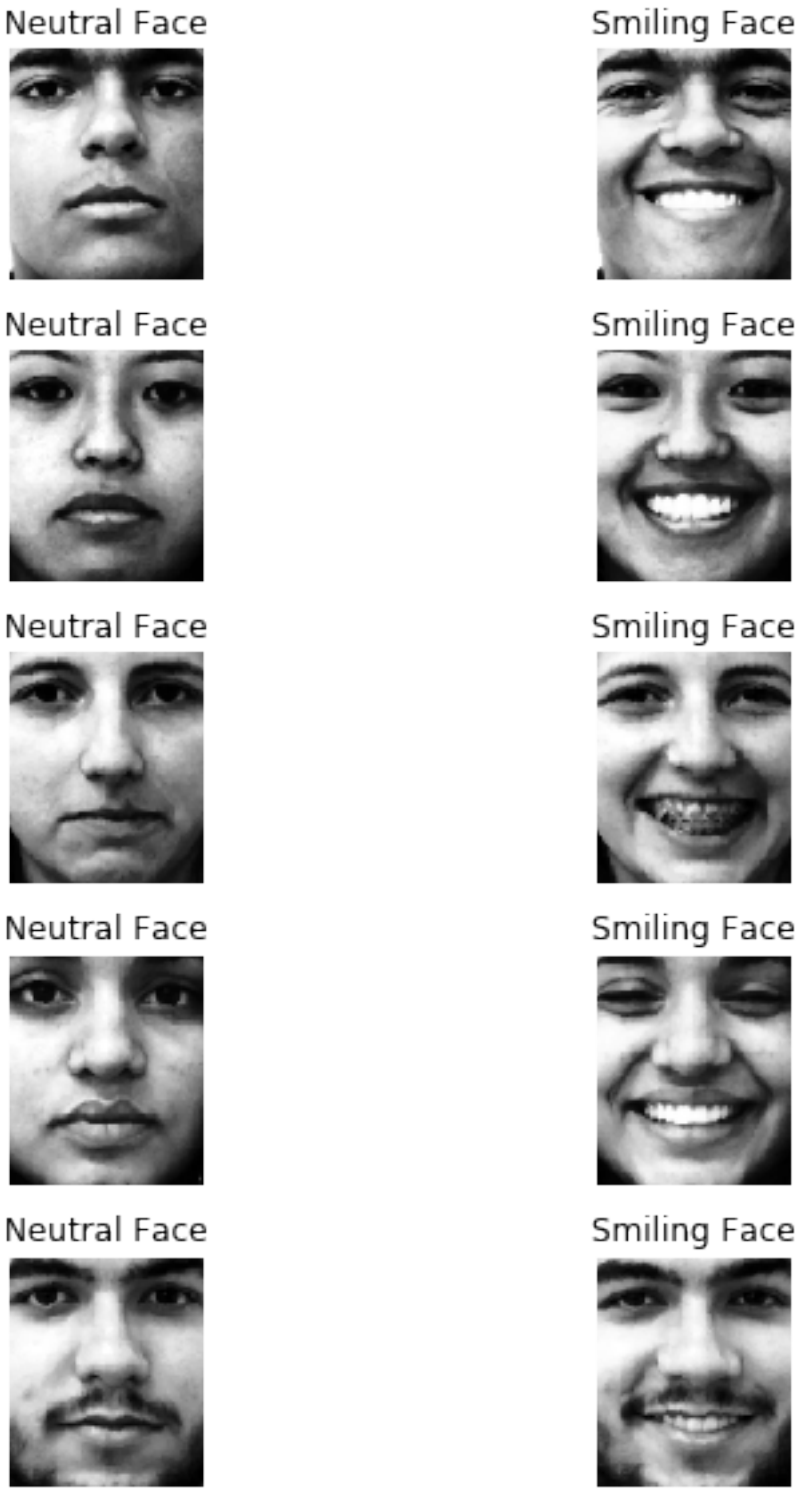
[4]:
```
for i in range(200):
    # Read the neutral images from the (i+1)a.jpg file
    # Read the smiling images from the (i+1)b.jpg file
    nimg = mpimg.imread(data_dir + str(i + 1) + "a.jpg")
    simg = mpimg.imread(data_dir + str(i + 1) + "b.jpg")
    neutral_imgs.append(nimg)
    smiling_imgs.append(simg)
dataset = neutral_imgs[:190]

# Convert the dataset of 190 neutral images to a numpy array
dataset = np.array(dataset)

# note down the image size
image_size = (dataset.shape[1], dataset.shape[2])
dataset = dataset.reshape(dataset.shape[0], dataset.shape[1] * dataset.shape[2])
# 31266 * 190 - dataset shape
dataset = np.transpose(dataset)
```

[5]:
```
# convert the dataset to float to avoid errors in processing later on
dataset = dataset.astype(np.float64)
```

[6]:
```
# convert the neutral and smiling images dataset to Numpy
neutral_imgs = np.array(neutral_imgs)
smiling_imgs = np.array(smiling_imgs)

# convert these images to float to avoid errors in processing later on
neutral_imgs = neutral_imgs.astype(np.float64)
smiling_imgs = smiling_imgs.astype(np.float64)
```

1.0.2 Display 5 random neutral and smiling images to verify if the data loading was correct

[7]:
```
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8, 8), sharex="all",␣
 ↪sharey="all")
for i in range(5):
    j = random.randint(0, 199)
    n = neutral_imgs[j]
    s = smiling_imgs[j]

    # display the neutral image
    axes[i][0].imshow(n, cmap="gray")
    axes[i][0].axis('off')
    axes[i][0].set_title("Neutral Face")

    # display the smiling image
    axes[i][1].imshow(s, cmap="gray")
    axes[i][1].axis('off')
    axes[i][1].set_title("Smiling Face")

plt.tight_layout()
fig.canvas.draw()
```


Neutral Face
Smiling Face
Neutral Face
Smiling Face
Neutral Face
Smiling Face
Neutral Face
Smiling Face
Neutral Face
Smiling Face

1.0.3 Question a) - Computing the PCs using 1st 190 individual's neutral expressions and plotting the singular values of the data matrix

Compute Mean Face

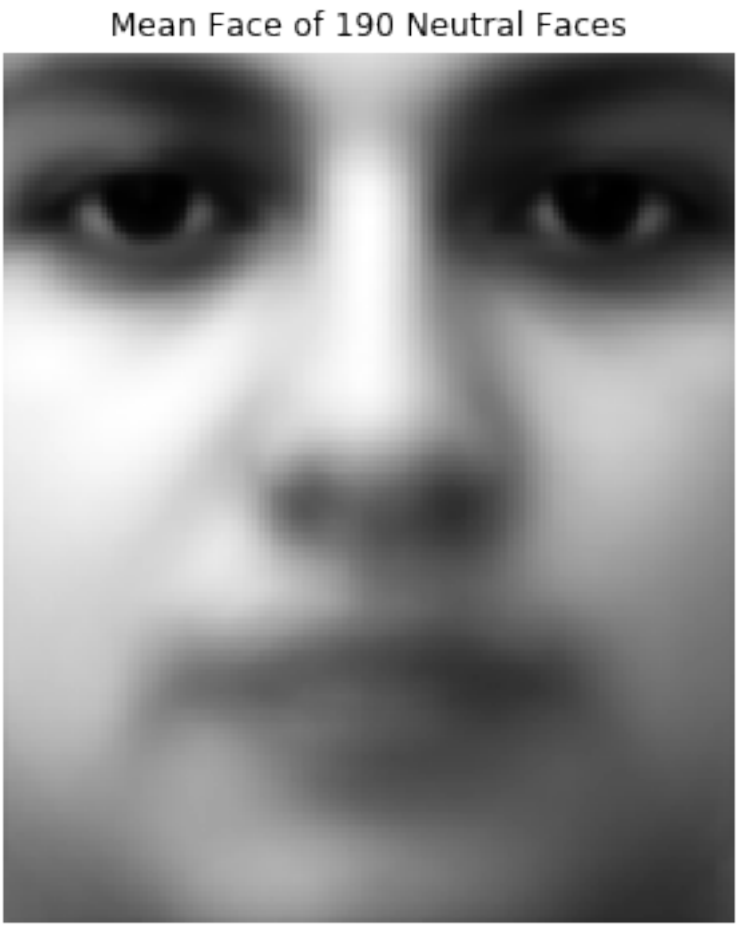
[8]:
```
# calculate the mean face for the dataset
num_imgs = dataset.shape[1]
mean_img = np.matmul(dataset, np.ones((num_imgs, 1))) / num_imgs
# 31266 * 1 - mean_img
mean_img = mean_img
print(mean_img.shape, mean_img.dtype, np.min(mean_img), np.max(mean_img))
```

```
(31266, 1) float64 3.057894736842105 247.05789473684212
```

As given in the paper "Face Recognition Using Eigenfaces" in Section 2.1, we compute the mean image from the dataset of images denoted by $\Gamma_1, \Gamma_2, ..., \Gamma_M$, where M is the total number of images in the training set. The mean is given by

$$\Psi = \frac{1}{M} \Sigma_{n=1}^{M} \Gamma_n$$

[9]:
```
# plot the mean face of the neutral expressions
plt_mean_img = mean_img.reshape(image_size[0], image_size[1])
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.imshow(plt_mean_img, cmap="gray")
axes.axis('off')
axes.set_title("Mean Face of 190 Neutral Faces")
fig.canvas.draw()
```



Compute the eigen values and the corresponding eigen faces

[10]:
```
# 31266 * 180
mean_offset = dataset - mean_img

# 180*180
mod_cov = np.matmul(np.transpose(mean_offset), mean_offset)

# check if this is a symmetric matrix
print(np.allclose(mod_cov, np.transpose(mod_cov)))
```

```
True
```

As given in the paper, we then compute the mean subtracted faces by:-

$$\Phi_i = \Gamma_i - \Psi$$

Then we compute the modified covariance as given in the Reference 11, which is the paper titled

"Eigenfaces for Recognition" by the following-

$$C' = A^T A$$

where

$$A = [\Phi_1 \Phi_2 ... \Phi_M]$$

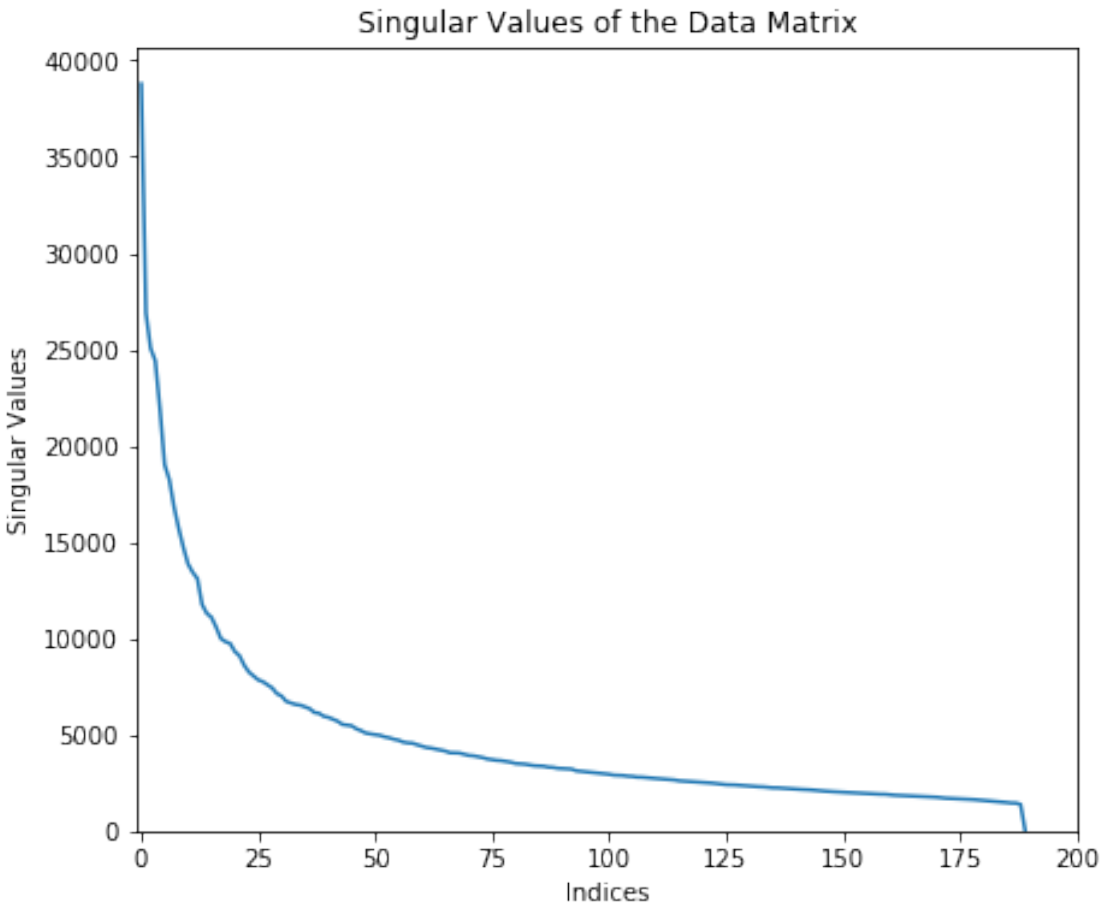
[11]:
```
# compute the eigen values and the corresponding eigenvectors in ascending order
# 190 eigen values
# 190 * 190 eigen vectors
# ith column - corresponding to the ith eigen vector
eig_vals, mod_eig_vecs = eigh(mod_cov)

# 31266 * 190 - eigen vectors of the original data
eig_vecs = np.matmul(mean_offset, mod_eig_vecs)

# normalize the eigen vectors now
# 31266 * 190 normalized eigen vectors
norm_cnst = np.sqrt(np.sum(np.square(eig_vecs), 0))
eigen_faces = np.divide(eig_vecs, norm_cnst)
```

As given in the paper, we compute the eigenvalues and the eigen vectors for C'. Using these, we compute the eigen vectors for $C = AA^T$ by simplying pre-multiplying all the eigen vectors of C' by A. Finally, we normalize the computed eigenvectors so that they all have unit norm.

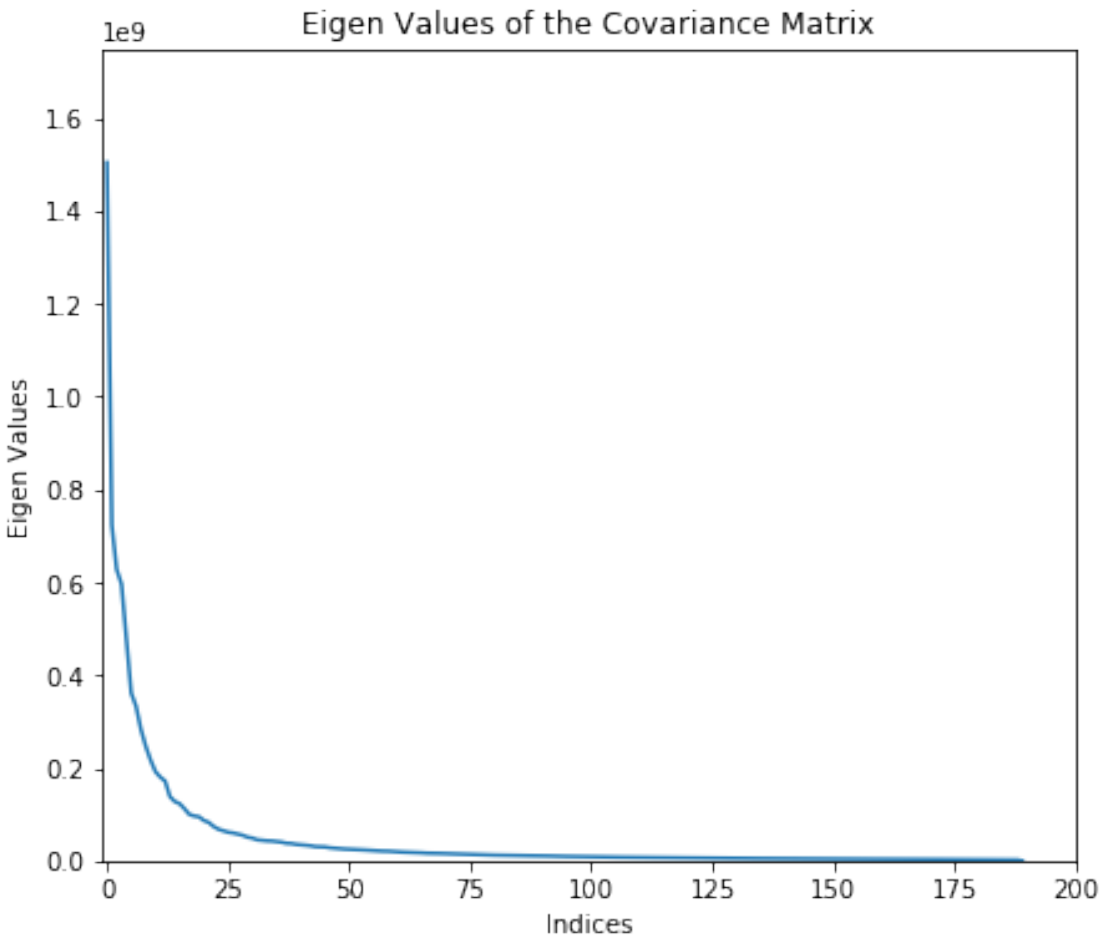
[12]:
```
# flip both eigen values and eigen vectors to ensure the largest eigen values␣
↪and their corresponding
# eigen vectors are at the front
eig_vals = np.flip(eig_vals)
eigen_faces = np.flip(eigen_faces, 1)
```

Plot the singular values of the data matrix The singular values of the data matrix A are the positive square roots of the eigen values of the symmetric matrix $A^T A$. Since the matrix $A^T A$ is symmetric, it's eigen values are the squares of the singular values of A and they are real.

[13]:
```
# plot the singular values of the data matrix
indices = [i for i in range(190)]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.plot(indices, np.sqrt(eig_vals))
axes.set_xlabel("Indices")
axes.set_ylabel("Singular Values")
axes.set_xlim(-1, 200)
axes.set_ylim(0,)
axes.set_title("Singular Values of the Data Matrix")
fig.canvas.draw()
```



[14]:
```
# plot the eigen values of the covariance matrix
indices = [i for i in range(190)]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.plot(indices, eig_vals)
axes.set_xlabel("Indices")
axes.set_ylabel("Eigen Values")
axes.set_xlim(-1, 200)
axes.set_ylim(0, 1.75e9)
axes.set_title("Eigen Values of the Covariance Matrix")
fig.canvas.draw()
```



Choosing number of eigen vectors to form PCs

As given in the discussion, we can fix our metric for choosing the number of eigen vectors to form the Principal Components. In my case, I am choosing to compute the minimum number of eigen values whose sum is greater than 95% of the sum of all the eigen values. As given in the paper "Face Recognition Using Eigenfaces" in Section 2, it states that "The faces can be approximated using only the best eigenfaces - those that have the largest eigenvalues, and which therefore account for the most variance within the set of face images". For choosing the eigen vectors, we take the largest eigen values as they indicate the maximum variance between the different features. Hence, we can state that the Principal components capture and encode the variation between faces in a lower dimension than the one spanned by vectors in the dimension of the total number of pixels in the image. Higher the variance, better it is to represent the images in that dimension with minimal loss of information. A larger eigen value represents higher variation, and hence, we choose those eigen vectors that are associated with the larger eigen values initially.

[15]:
```
tot = np.sum(eig_vals)
su = 0
```

```
ans = -1
for j in range(190):
    su += eig_vals[j]
    if su/tot>0.95:
        ans = j+1
        print("The largest {} eigen vectors encode >95% variability of the␣
 ↪dataset".format(ans))
        break
```

```
The largest 96 eigen vectors encode >95% variability of the dataset
```

[16]:
```
# I can call this the critical number of eigen vectors
critic_eig = ans
```

Hence, we can choose 96 largest eigen values and their corresponding eigen vectors to represent greater than 95 variability of the given dataset of faces.

[17]:
```
# plot the top 5 eigen faces
fig, axes = plt.subplots(nrows=5, ncols=1, figsize=(8, 8), sharex="all",␣
 ↪sharey="all")
for i in range(5):
    img = eigen_faces[:, i]
    plot_face = img.reshape(image_size[0], image_size[1])

    # display the neutral image
    axes[i].imshow(plot_face, cmap="gray")
    axes[i].axis('off')
    axes[i].set_title("Eigen Face: {}".format(i+1))

plt.tight_layout()
fig.canvas.draw()
```

Eigen Face: 1

Eigen Face: 2

Eigen Face: 3

Eigen Face: 4

Eigen Face: 5

1.0.4 Question b) - Reconstruction of a neutral individual's face using different number of PC's

[18]:
```
def reconstruct_image(orig, pcs):
    """
    A function to reconstruct the original image with the given PCs
    Arguments
    orig - Original image to perform reconstruction on
    pcs  - The principal components to use for the reconstruction

    Returns
    recon - The reconstruction of the orig from pcs
    mse   - The Mean Squared Error of the recon with orig
    """
    # perform mean subtraction
    # 31566 * 1
    mean_sub = orig - mean_img

    # calculate the weights
    # l * 31566 * 31566 * 1 = l * 1
    w = np.matmul(np.transpose(pcs), mean_sub)

    # 31566 * l * l * 1 = 31566 * 1
    recon = np.matmul(pcs, w) + mean_img
    mse = np.mean(np.square(orig - recon))
    return recon, mse
```

For projection and reconstruction of a new image, from section 2.2 of the paper, we find the weights of the mean subtracted original image as follows:-

$$w_k = u_k^T(\Gamma - \Psi)$$

These weights form the vector Ω given by

$$\Omega^T = [w_1 w_2 ... w_{M'}]$$

where M' is the number of PCs that we use. Finally the reconstruction is given by :-

$$\Phi_f = \Sigma_{i=1}^{M'} w_k u_k + \Psi$$

[19]:
```
# random index to pick image from
rand_ind = random.randint(0, 189)
original_img = dataset[:, rand_ind]
original_img = original_img[:, np.newaxis]
```

[20]:
```
mses = []
recs = []
```

```
# Reconstruct the image for all the PCs
for j in range(190):
    pcs = eigen_faces[:, :(j+1)]
    rec, mse = reconstruct_image(original_img, pcs)
    recs.append(rec)
    mses.append(mse)
```

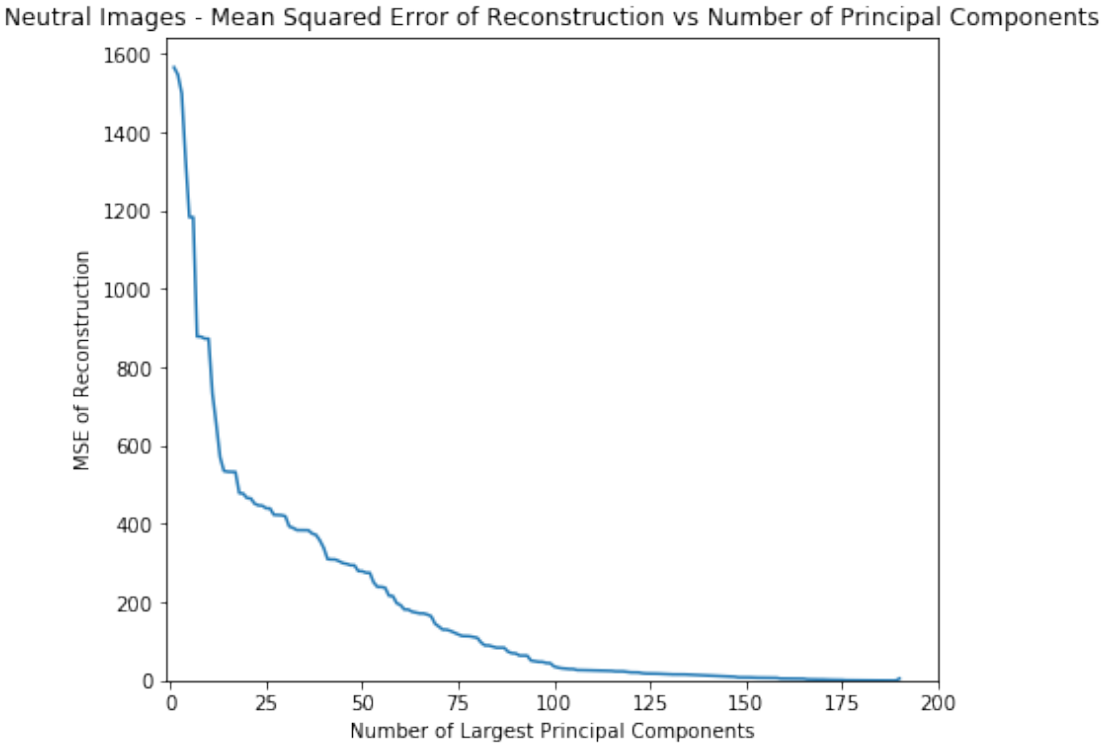
[21]:
```
# Print the minimum and maximum MSE obtained
print(min(mses), max(mses))
```

```
4.886378871904064e-25 1566.234607083892
```

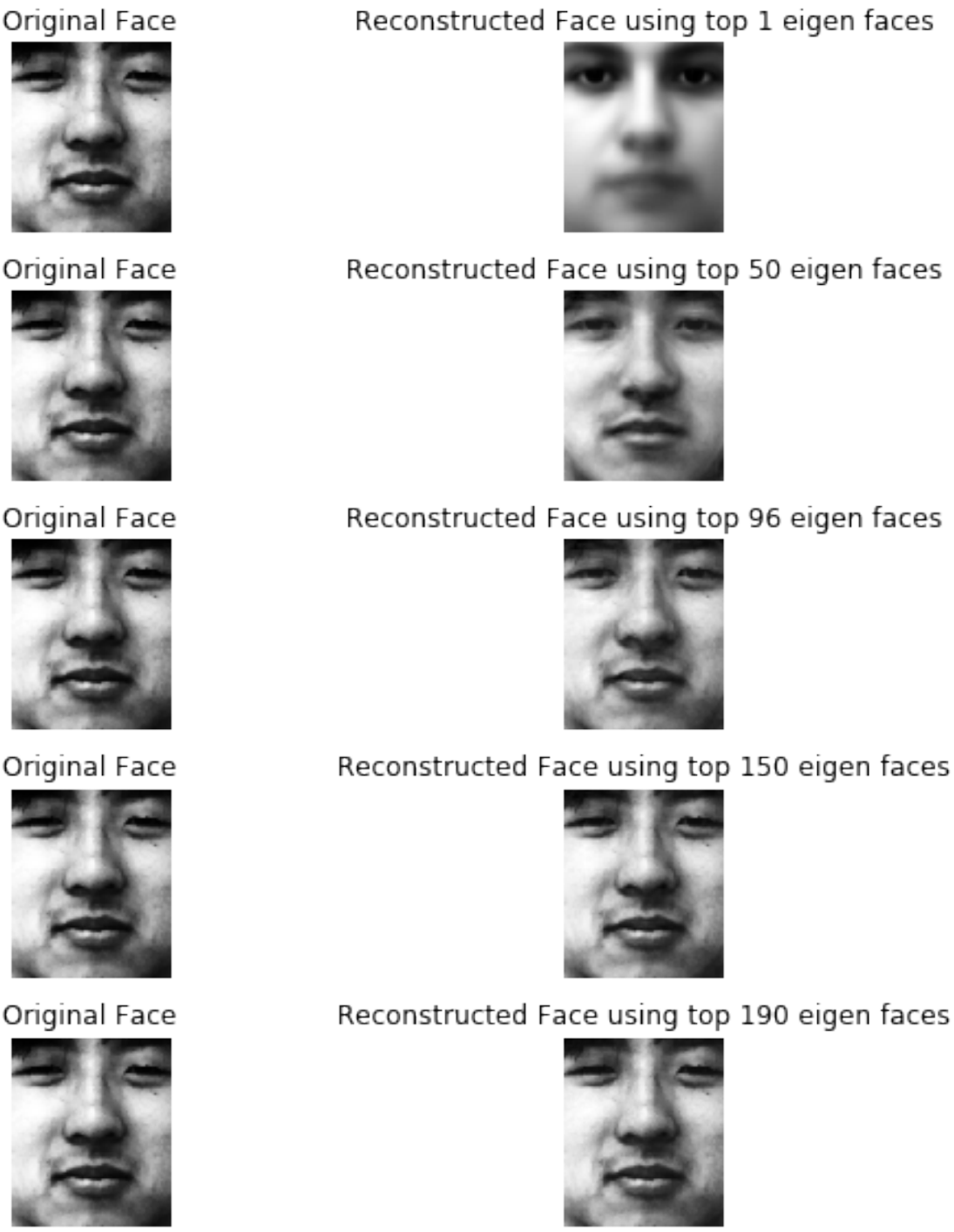
[22]:
```
# print the MSE obtained for the eigen vector associated with the critical␣
↪eigen value
print(mses[critic_eig - 1])
```

```
47.85609385885828
```

[23]:
```
# plot the MSE of reconstruction vs number of largest principal components used
num_eigs = [i+1 for i in range(190)]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.plot(num_eigs, mses)
axes.set_xlabel("Number of Largest Principal Components")
axes.set_ylabel("MSE of Reconstruction")
axes.set_xlim(-1, 200)
axes.set_ylim(0, )
axes.set_title("Neutral Images - Mean Squared Error of Reconstruction vs Number␣
↪of Principal Components")
fig.canvas.draw()
```



[24]:
```
# Plot the reconstructed images for 5 different number of PCs
num_pcs = [1, 50, critic_eig, 150, 190]
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8, 8), sharex="all",␣
 ↪sharey="all")
for i in range(len(num_pcs)):
    r = recs[num_pcs[i] - 1]
    r = r.reshape(image_size[0], image_size[1])

    # display the neutral image
    axes[i][0].imshow(original_img.reshape(image_size[0], image_size[1]),␣
 ↪cmap="gray")
    axes[i][0].axis('off')
    axes[i][0].set_title("Original Face")

    # display the smiling image
    axes[i][1].imshow(r, cmap="gray")
    axes[i][1].axis('off')
    axes[i][1].set_title("Reconstructed Face using top {} eigen faces".
 ↪format(num_pcs[i]))

plt.tight_layout()
fig.canvas.draw()
```



Qualitative Results

I reconstructed a random neutral image from the 1^{st} 190 neutral images using different number of principal components associated with the largest eigen values. I found that the reconstructed

image was not visually close for just 1 principal component. As the number of principal components increases, I found that the reconstructed image starts resembling the original image. The best results seem to be obtained after 96 PCs, which is the critical number of PCs that I calculated.

Quantitave Results

As the number of principal components associated with the largest eigen values used to recreate the neutral image increases, the MSE for my model keeps on decreasing to 0. This is to be expected as this random neutral image that we picked was used in the computation of all the 190 eigen faces, and hence, this neutral image can be spanned by all the 190 principal components. The highest MSE was 1566.2346 for just 1 component, and it lowered to 4.886×10^{-25}. The highest MSE that we obtained was lesser than 255^2, which is the largest possible error that can be obtained between 2 grayscale images. For 96 PCs, the MSE was 47.8561, which is a good reconstruction MSE for just half the total number of principal components. This means that on average, the pixel values between reconstructed image and the original image differ by just a value of 7, which should be imperceptible to the human eye.

190^{th} PC addition

The MSE trend is followed for 189 PCs, except for the final one, where there is a very slight increase in the MSE. This could be due to the fact that the smallest eigen value is very close to 0, leading to numerical inconsistencies in the computation of the weights of the final PC, thus leading to an increase in MSE only for the final added 190^{th} PC.

1.0.5 Question c) - Reconstruction of a smiling individual's PCs using different number of PC's

[25]:
```
# pick the same index, but from the smiling part instead
original_img = smiling_imgs[rand_ind, :, :]
original_img = original_img.reshape(image_size[0] * image_size[1], 1)
```

[26]:
```
sm_mses = []
sm_recs = []

# Reconstruct the image for all the PCs
for j in range(190):
    pcs = eigen_faces[:, :(j+1)]
    rec, mse = reconstruct_image(original_img, pcs)
    sm_recs.append(rec)
    sm_mses.append(mse)
```

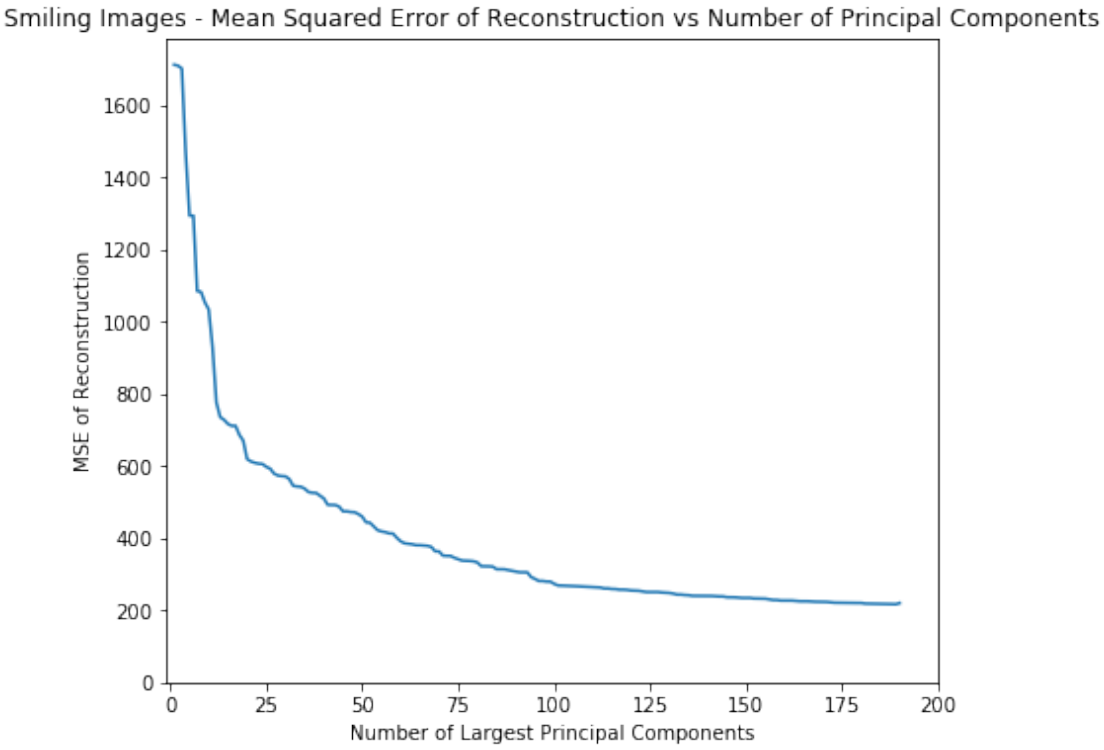
[27]:
```
print(min(sm_mses), max(sm_mses))
```

```
216.84755056183363 1712.0047184654857
```

[28]:
```
print(sm_mses[critic_eig - 1])
```

```
281.2805022934623
```

[29]:
```
# plot the MSE of Reconstruction vs the Number of Largest Principal Components
num_eigs = [i+1 for i in range(190)]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.plot(num_eigs, sm_mses)
axes.set_xlabel("Number of Largest Principal Components")
axes.set_ylabel("MSE of Reconstruction")
axes.set_xlim(-1, 200)
axes.set_ylim(0, )
axes.set_title("Smiling Images - Mean Squared Error of Reconstruction vs Number␣
↪of Principal Components")
fig.canvas.draw()
```



[30]:
```
# Plot the reconstructed images for 5 different number of PCs
num_pcs = [1, 50, critic_eig, 150, 190]
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8, 8), sharex="all",␣
↪sharey="all")
for i in range(len(num_pcs)):
    r = sm_recs[num_pcs[i] - 1]
    r = r.reshape(image_size[0], image_size[1])

    # display the neutral image
    axes[i][0].imshow(original_img.reshape(image_size[0], image_size[1]),␣
↪cmap="gray")
```

```
    axes[i][0].axis('off')
    axes[i][0].set_title("Original Face")

    # display the smiling image
    axes[i][1].imshow(r, cmap="gray")
    axes[i][1].axis('off')
    axes[i][1].set_title("Reconstructed Face using top {} eigen faces".
 ↪format(num_pcs[i]))

plt.tight_layout()
fig.canvas.draw()
```

Original Face

Reconstructed Face using top 1 eigen faces

Original Face

Reconstructed Face using top 50 eigen faces

Original Face

Reconstructed Face using top 96 eigen faces

Original Face

Reconstructed Face using top 150 eigen faces

Original Face

Reconstructed Face using top 190 eigen faces

Qualitative Results

I reconstructed a random smiling image from the 1^{st} 190 smiling images using different number of principal components associated with the largest eigen values. I used the same face for both neutral

and for smiling images in this project for ease of comparison. Visually, the reconstructed images get better as the number of eigen faces that we use increases. But, this is still not as good as the reconstruction that we perform for the neutral images. This is because in this specific example, we find that the smiling image shows teeth, which is not present in any of the neutral images. Hence, we find that the results are qualitatively not as great as we obtained for the reconstruction of neutral images.

Quantitave Results

As the number of principal components associated with the largest eigen values used to recreate the smiling image increases, the MSE for my model keeps on decreasing. But, the MSE of reconstruction does **not** approach 0. The reason for this is that the smiling images are not a part of the process of finding the principal components. Hence, with the given principal components, we cannot span the set of smiling images effectively. For 96 PCs, the MSE was 281.281. Here, we find that 96 PCs do not adequately capture the smiling images, but then, increasing the number of principal components used slightly decreases the MSE. Also, as a sanity check, the highest MSE that we obtained was lesser than 255^2, which is the largest possible error that can be obtained between 2 grayscale images. The highest MSE was 1712.005 for just 1 component, and it lowered to 216.8475

190^{th} PC addition

The MSE trend is followed for 189 PCs, except for the final one, where there is a very slight increase in the MSE. This could be due to the fact that the smallest eigen value is very close to 0, leading to numerical inconsistencies in the computation of the weights of the final PC, thus leading to an increase in MSE only for the final added 190^{th} PC.

Comparison with Neutral Image Reconstruction

This reconstruction is not as good as the neutral image reconstruction. This is because the smiling image was not a part of process of finding the principal components. Hence, with the given principal components, we cannot span the space of smiling images. We can only approximate the smiling image through this process.

1.0.6 Question d) Reconstruction of test set of neutral images from the Principal Components

[31]:
```
# random index to pick image from
rand_neut = random.randint(190, 199)
original_img = neutral_imgs[rand_neut, :, :]
original_img = original_img.reshape(image_size[0] * image_size[1], 1)
```

[32]:
```
te_mses = []
te_recs = []

# Reconstruct the image for all the PCs
for j in range(190):
    pcs = eigen_faces[:, :(j+1)]
    rec, mse = reconstruct_image(original_img, pcs)
    te_recs.append(rec)
```

```
    te_mses.append(mse)
```

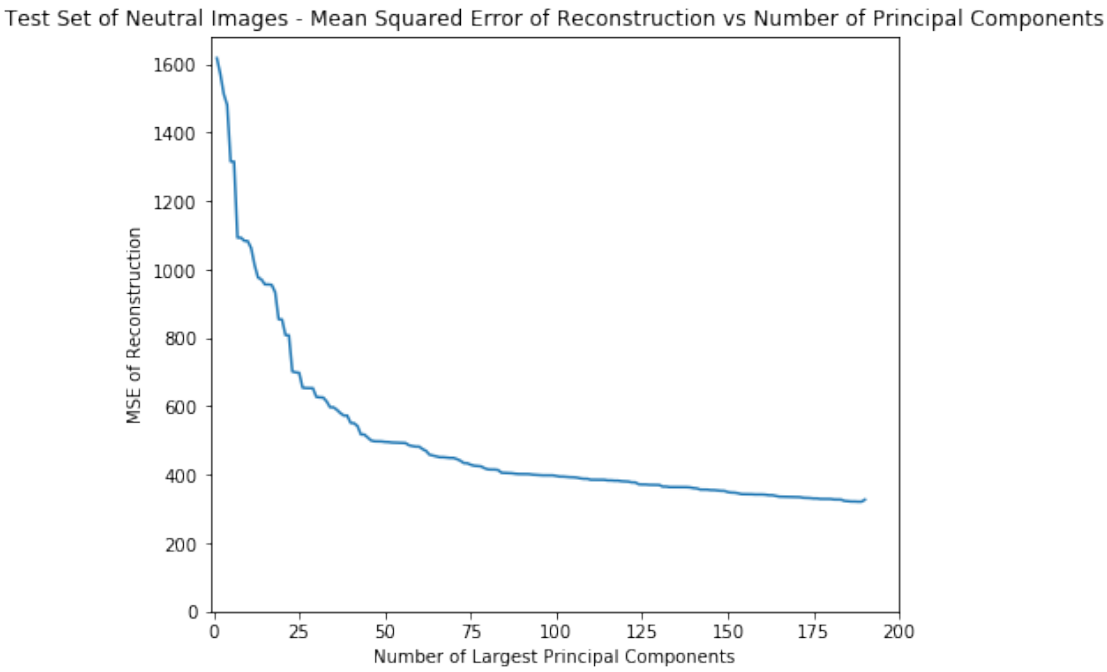
[33]:
```
print(min(te_mses), max(te_mses))
```

```
321.149315915176 1617.694093212688
```

[34]:
```
print(te_mses[critic_eig - 1])
```

```
398.3426233925179
```

[35]:
```
# plot the singular values of the data matrix
num_eigs = [i+1 for i in range(190)]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.plot(num_eigs, te_mses)
axes.set_xlabel("Number of Largest Principal Components")
axes.set_ylabel("MSE of Reconstruction")
axes.set_xlim(-1, 200)
axes.set_ylim(0,)
axes.set_title("Test Set of Neutral Images - Mean Squared Error of␣
 ↪Reconstruction vs \
Number of Principal Components")
fig.canvas.draw()
```



[36]:
```
# Plot the reconstructed images for 5 different number of PCs
num_pcs = [1, 50, critic_eig, 150, 190]
```

```
fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(8, 8), sharex="all",␣
 ↪sharey="all")
for i in range(len(num_pcs)):
    r = te_recs[num_pcs[i] - 1]
    r = r.reshape(image_size[0], image_size[1])

    # display the neutral image
    axes[i][0].imshow(original_img.reshape(image_size[0], image_size[1]),␣
 ↪cmap="gray")
    axes[i][0].axis('off')
    axes[i][0].set_title("Original Face")

    # display the smiling image
    axes[i][1].imshow(r, cmap="gray")
    axes[i][1].axis('off')
    axes[i][1].set_title("Reconstructed Face using top {} eigen faces".
 ↪format(num_pcs[i]))

plt.tight_layout()
fig.canvas.draw()
```

Original Face

Reconstructed Face using top 1 eigen faces

Original Face

Reconstructed Face using top 50 eigen faces

Original Face

Reconstructed Face using top 96 eigen faces

Original Face

Reconstructed Face using top 150 eigen faces

Original Face

Reconstructed Face using top 190 eigen faces

Qualitative Results

I tried reconstruction of a random neutral image using different number of principal components with the largest eigen values I found that the reconstructed image was not close for just 1 principal

component. For more principal components, I found that the reconstructed image starts resembling the original image.

Quantitave Results

As the number of principal components associated with the largest eigen values used to recreate the neutral image increases, the MSE for my model keeps on decreasing. This is to be expected, as the number of Pricipal components increases, they are able to approximately span the original image more and more. The MSE of reconstruction does **not** approach 0. The reason for this is that the given neutral images are not a part of the process of finding the principal components. Hence, with the given principal components, we cannot span the set of 10 other neutral images effectively. For 96 PCs, the MSE was 398.343. Here, we find that 96 PCs captures the neutral image approximately, but then, increasing the number of principal components used slightly decreases the MSE. Also, as a sanity check, the highest MSE that we obtained was lesser than 255^2, which is the largest possible error that can be obtained between 2 grayscale images. The highest MSE was 1617.694 for just 1 component, and it lowered to 321.149

190^{th} PC addition

The MSE trend is followed for 189 PCs, except for the final one, where there is a very slight increase in the MSE. This could be due to the fact that the smallest eigen value is very close to 0, leading to numerical inconsistencies in the computation of the weights of the final PC, thus leading to an increase in MSE only for the final added 190^{th} PC.

Comparison with Neutral Image Reconstruction

This reconstruction is not as good as the neutral images reconstruction from the original set. This is because this neutral image was not a part of process of finding the principal components. Hence, with the given principal components, we cannot span this image completely.

Comparison with Smiling Image Reconstruction

This reconstruction is very similar to the one performed on smiling images, and the same argument applies. The fact that they do not contribute to finding the Principal Components leads to the fact that they cannot efficiently reconstruct the given image. For the MSE comparison, the smiling images start with a higher MSE initially, leading to the conclusion that the smiling images are fundamentally more different than neutral images. This may be because the neutral images from the training set are more different from smiling images than the test set of the neutral images. This is because the smiling images are different in the mouth and the eyes part compared to neutral images.

1.0.7 Question e) Reconstruction of non-human image from the set of PCs

[37]:
```
# read car
car_pth = "../datasets/car.jpg"
car_img = mpimg.imread(car_pth)
car_img = car_img.astype(np.float64)
car_img = car_img.reshape(image_size[0] * image_size[1], 1)
```

[38]:
```
bird_pth = "../datasets/bird_black.jpg"
bird_img = mpimg.imread(bird_pth)
bird_img = bird_img.astype(np.float64)
bird_img = bird_img.reshape(image_size[0] * image_size[1], 1)
```

[39]:
```
# Reconstruct the image for all the PCs
pcs = eigen_faces
car_rec, car_mse = reconstruct_image(car_img, pcs)
bird_rec, bird_mse = reconstruct_image(bird_img, pcs)
```

[40]:
```
print(car_mse, bird_mse)
```

```
5164.572658968666 3744.056504204585
```

[41]:
```
recs = [car_rec, bird_rec]
origs = [car_img, bird_img]
```

[42]:
```
# Plot the reconstructed images for 5 different number of PCs
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(8, 8), sharex="all",␣
 ↪sharey="all")
for i in range(2):
    o = origs[i]
    r = recs[i]
    o = o.reshape(image_size[0], image_size[1])
    r = r.reshape(image_size[0], image_size[1])

    # display the neutral image
    axes[i][0].imshow(o, cmap="gray")
    axes[i][0].axis('off')
    axes[i][0].set_title("Original Image")

    # display the smiling image
    axes[i][1].imshow(r, cmap="gray")
    axes[i][1].axis('off')
    axes[i][1].set_title("Reconstructed Image using top {} eigen faces".
 ↪format(num_pcs[i]))

plt.tight_layout()
fig.canvas.draw()
```

Qualitative Results

Qualitatively, we find that the eigen faces can span only the set of neutral face images effectively, and hence it tries to approximate a new image to the face space spanned by the eigen faces. But, it does not do a good job of the reconstruction as the face image is inherently different from a car or a bird. This means that inherent quality of cars and birds is discarded, and instead an approximation of a face is forced upon their lower dimensional representation.

Quantitative Results

For the case of a non-human face image, we find that the reconstruction is very poor. In the case of the car image, the reconstruction MSE is 5164.5 and for the bird image, it is 3744.05. This

occurs because the projection weights of each eigen face for these images is very low, leading to the dominance of the mean face during the reconstruction.

1.0.8 Question f) Reconstruction of training set neutral image for different rotations

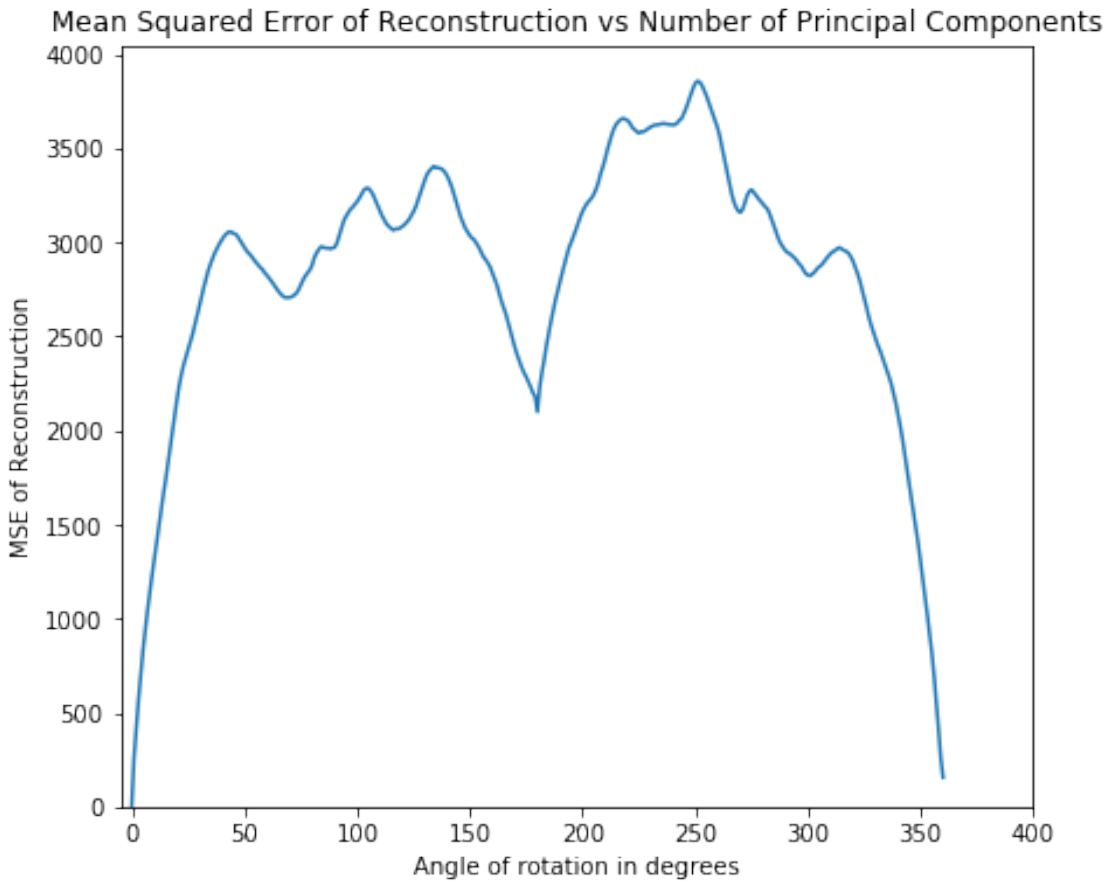
[43]:
```
original_img = dataset[:, rand_ind]
original_img = np.reshape(original_img, (image_size[0], image_size[1]))
```

[53]:
```
mses = []
rots = []
recs = []

pcs = eigen_faces

# Reconstruct the image for all the PCs
for j in range(361):
    rot_image = ndi.rotate(original_img, j, reshape=False)
    rot_image = rot_image.reshape(image_size[0]*image_size[1], 1)
    rots.append(rot_image)
    rec, mse = reconstruct_image(rot_image, pcs)
    recs.append(rec)
    mses.append(mse)
```

[54]:
```
print(min(mses), max(mses))
```

```
5.073264447417301 3855.5505773907694
```

[55]:
```
# plot the singular values of the data matrix
angles = [i for i in range(361)]
fig, axes = plt.subplots(nrows=1, ncols=1, figsize=(7, 6))
axes.plot(angles, mses)
axes.set_xlabel("Angle of rotation in degrees")
axes.set_ylabel("MSE of Reconstruction")
axes.set_xlim(-5, 400)
axes.set_ylim(0, )
axes.set_title("Mean Squared Error of Reconstruction vs Number of Principal␣
 ↪Components")
fig.canvas.draw()
```



[59]:
```
# Plot the reconstructed images for 5 different number of PCs
angles = [0, 2, 5, 45, 90, 120, 180, 270, 300, 360]
fig, axes = plt.subplots(nrows=10, ncols=3, figsize=(12, 12), sharex="all",␣
 ↪sharey="all")
for i in range(len(angles)):
    r = recs[angles[i]]
    r = r.reshape(image_size[0], image_size[1])

    ro = rots[angles[i]]
    ro = ro.reshape(image_size[0], image_size[1])

    # display the neutral image
    axes[i][0].imshow(original_img.reshape(image_size[0], image_size[1]),␣
 ↪cmap="gray")
    axes[i][0].axis('off')
    axes[i][0].set_title("Original Face")

    # display the rotated image
    axes[i][1].imshow(ro, cmap="gray")
```

```
    axes[i][1].axis('off')
    axes[i][1].set_title("Rotated Image Angle: {}".format(angles[i]))

    # display the smiling image
    axes[i][2].imshow(r, cmap="gray")
    axes[i][2].axis('off')
    axes[i][2].set_title("Reconstructed Face with all PCs".format(angles[i]))

plt.tight_layout()
fig.canvas.draw()
```

[67]:
```
# Compute the MSE between the original and 360 rotated image to explain why it␣
 ↪is not 0 again
mse_rot360 = np.mean(np.square(original_img - rots[360].reshape(image_size[0],␣
 ↪image_size[1])))
print("The MSE between the original image and the 360* rotated image is {0:0.
 ↪3f}".format(mse_rot360))
```

```
print("The difference in MSE of reconstruction between the original image and␣
 ↪the 360* rotated \
image is {0:0.3f}".format(mses[360] - mses[0]))
```

```
The MSE between the original image and the 360* rotated image is 165.657
The difference in MSE of reconstruction between the original image and the 360*
rotated image is 150.882
```

Qualitative Results

Qualitatively, we see that as we increase the angle that we rotate the original image by from 0, the reconstructed images are increasingly worse. For the 180* rotation, we find that the reconstruction is slighly better than for it's surrounding angles. This might be because the inverted face has a nose lined up perfectly with the original nose, thus decreasing the MSE for that particular case. As we increase from 180*, we find that reconstructed images again get progressively worse, and then again start resembling the original image as we increase the angle of rotation to almost 360*. When the angle is 360*, we find that the reconstruction is again almost as good as the unrotated image

Quantitative Results

As we increase the angle of rotation from 0, the MSE increases from it's value of almost 0 for the original image to a high value. Then, the MSE decreases till 180*. The MSE increases again till about 260* as seen in the graph. Finally, the MSE decreases till we complete the full rotation of 360*.

MSE of Reconstruction for 0* $\neq$ MSE of Reconstruction for 360*- Reason

Since rotating any image by 360* should give the same unrotated image, the image should ideally be the same, and hence the reconstruction should also be the same. The reason that this is not the case is because the library *scipy.ndimage.rotate*() rotates the image by 360*, leading to numerical approximations. We find that the MSE of reconstruction for the 360* rotated image is higher than that of the unrotated image by 150.882. But, the MSE between the original image and the original 360* rotated image is 165.657. This means that the original image rotated that we used for reconstruction itself was different, leading to different MSE for the reconstructed image. This problem can be solved by only rotating images modulo 360*

1.0.9 Conclusion

Thus, I have implemented PCA on face images in the form of Eigenfaces in this project. Given a dataset, I computed the PCs and performed various experiments on it. I implemented and reasoned about the performance of the reconstruction of images within the training set of neutral images. I studied how the reconstruction is impacted for neutral images outside the training set, as well as for images from a smiling dataset. For non-human images, I analyzed the performance of the given algorithm. Finally, I reasoned about the performance of the reconstruction of images that are rotated by varying degrees.

Project Completed By:-

Name: Anirudh Swaminathan

PID: A53316083

Email ID: aswamina@ucsd.edu