ECE269 PROJECT REPORT

FALL 2019

Jiaming Lai
Department of Electrical and Computer Engineering
University of California San Diego
La Jolla, CA 92037
jil136@ucsd.edu

December 3, 2019

1 Introduction

Following the "Eigenface" technique proposed by Matthew, et al. (1995) [1][2], we derive eigenfaces (obtained via PCA) from face image dataset. We use these eigenfaces (also known as principal components, PCs) to reconstruct different kinds of images and evaluate the result via computing MSE. In section 2, we will cover a simplified description of how the PCs are calculated and reconstruction is done. Combining with the singular values plot, we will give our justification of those PCs. Section 3-5 are about reconstruction result using images from 190 individuals neutral expression image set, from 190 individuals smiling expression image set and from the other 10 individuals neutral expression image set. In section 6, we will reconstruct non-human images using all the PCs and evaluate the result. Section 7 is to reconstruct rotated images and our comment of the result.

2 Singular values plot and justification of PCs

2.1 Simplified description of PC computation and reconstruction

As said in the Discussion 8, we construct covariance matrix C from training image dataset:

$$C = AA^T \tag{1}$$

In order to compute in an efficient way, we compute the eigenvector v_i and its corresponding eigenvalues μ_i of matrix A^TA. In our case, there should be 190 eigenvectors v_i and eigenvalues μ_i, because we use first 190 individuals' neutral expression images to construct the covariance matrix C. We then translate these 190 eigenvectors v_i to eigenvectors for C:

$$u_i = \frac{Av_i}{||Av_i||_2},\ where\ i = 1, 2, \ldots, 190 \tag{2}$$

Given an image Γ, the reconstruction is as following:

$$\hat{\Gamma} = \sum_{i=1}^{N} \omega_i u_i + \Psi \tag{3}$$

where $\omega_i = u_i^T(\Gamma - \Psi)$ and $N \leq 190$ is number of PCs we want to choose. In section 2.3, we will use the notation above to make more justification and explaination.

2.2 Singular values plot

Singular values σ_i are obtained as following:

$$\sigma_i = \sqrt{\mu_i},\ where\ i = 1, 2, \ldots, 190 \tag{4}$$

Figure 1 is the singular values plot from data matrix.

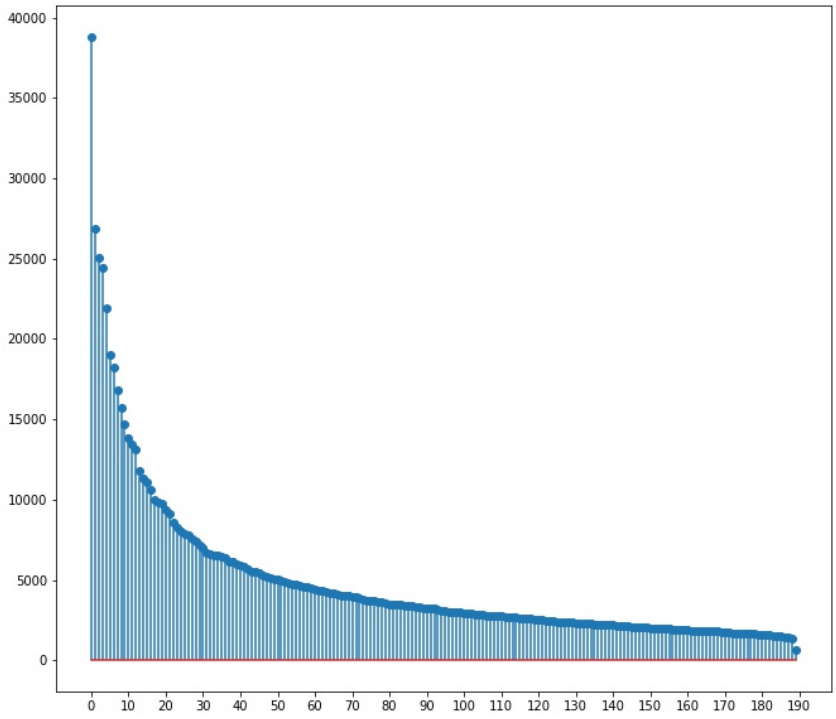

Figure 1: Singular values plot

2.3 Justification and explaination

Suppose we want to choose N PCs, which are also known as eigenfaces, then these PCs are actually eigenvectors of C, $u_1, u_2, \ldots, u_N$, that corresponds to top eigenvalues $\mu_1, \mu_2, \ldots, \mu_N$. Suppose $B = [u_1, u_2, \ldots, u_N]$, then because u_i are orthogomal, matrix BB^T is actually a projection matrix. Given a image Γ, the reconstruction is actually project the raw image from $R(A)$ to $R(BB^T)$, while $R(BB^T)$ is known as face space, having lower dimension than $R(A)$. When we compute MSE, we are actually computing the norm distance between the raw image Γ and the projected image $\hat{\Gamma}$. The number of PCs we choose is depended on how large MSE we can stand. In pratice, there should be a threshod of MSE, we could always choose less eigenvectors as PCs given the MSE is larger than the threshod.

The technique we use here is to reduce dimension. When we project the raw data onto a subspace (in our case, the subspace is the face space), we could derive hyperplanes to classify different clusters. This technique is powerful not only because it reduce the dimension and make computation efficiently, but also, by reducing dimension, we perform denoising on the data.

3 Reconstruction result of problem (b)

3.1 MSE figures

We randomly select three images from 190 individuals neutral expression image set and reconstruct each of the images using using different number of PCs. We compute the mean squared error (MSE), see figure 2

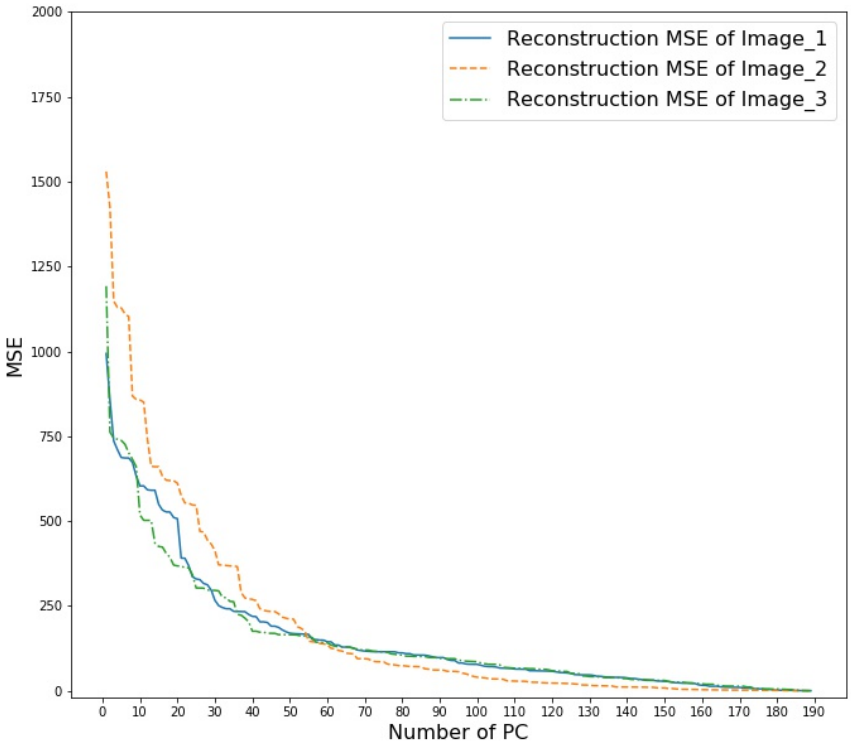

Figure 2: MSE versus number of principal components
of 190 individuals neutral expression images

3.2 Comment and explaination

As we can see in figure 2, when the number of PCs increases, the MSE of three randomly selected images decrease and finally all converge to nearly zero. It is obvious because those PCs are derived from the first 190 individuals neutral expression images. When we select one raw image from this 190 images set and use more and more PCs to reconstruct it, the MSE would, off course, decrease to nearly zero.

4 Reconstruction result of problem (c)

4.1 MSE figures

The same as section 3, we randomly select three images from 190 individuals smiling expression image set and reconstruct each of the images using using different number of PCs. We compute the mean squared error (MSE), see figure 3

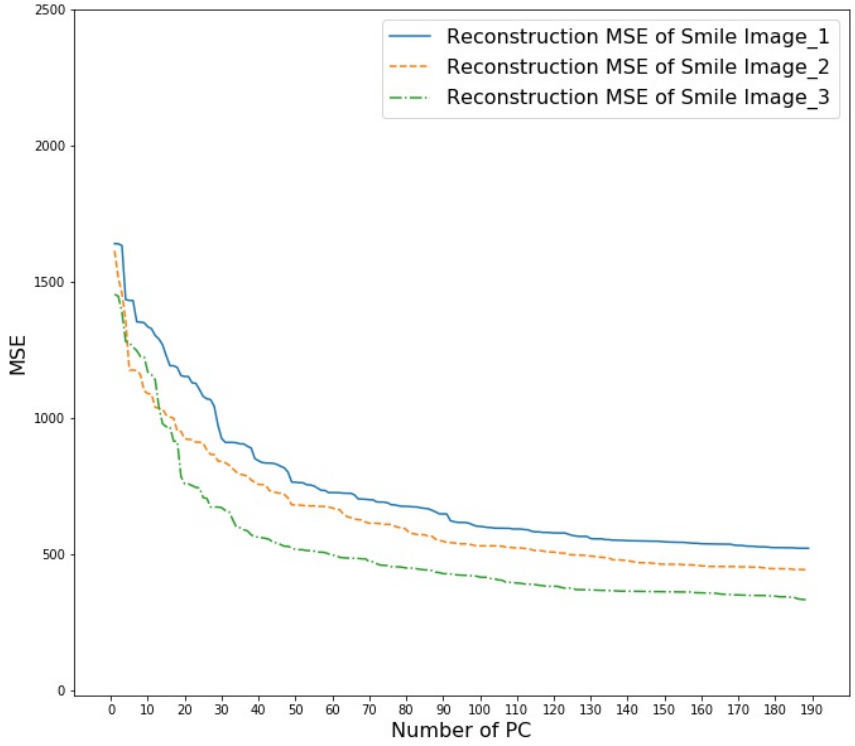

Figure 3: MSE versus number of principal components
of 190 individuals smiling expression image

4.2 Comment and explaination

The MSE of reconstruction of three randomly selected images have the same tendency. When the number of PCs increases, the MSE decreases. But the MSE would not decrease to zero. When PCs' number is large enough, the MSE converge to a bottleneck value. The bottlenelt value is about 500.

Because those three images are selected from 190 individuals smiling expression image set, so they are not included in our training dataset. The PCs we obtained only capture the features related to our training data. There are common features among smiling faces and neutral faces. This explains why the MSE would decrease. But there should be some special features of smiling faces, which results in the bottleneck value of MSE.

5 Reconstruction result of problem (d)

5.1 MSE figures

The same as section 3-4, we randomly select three images from the other 10 individuals neutral expression image set and reconstruct each of the images using using different number of PCs. We compute the mean squared error (MSE), see figure 4

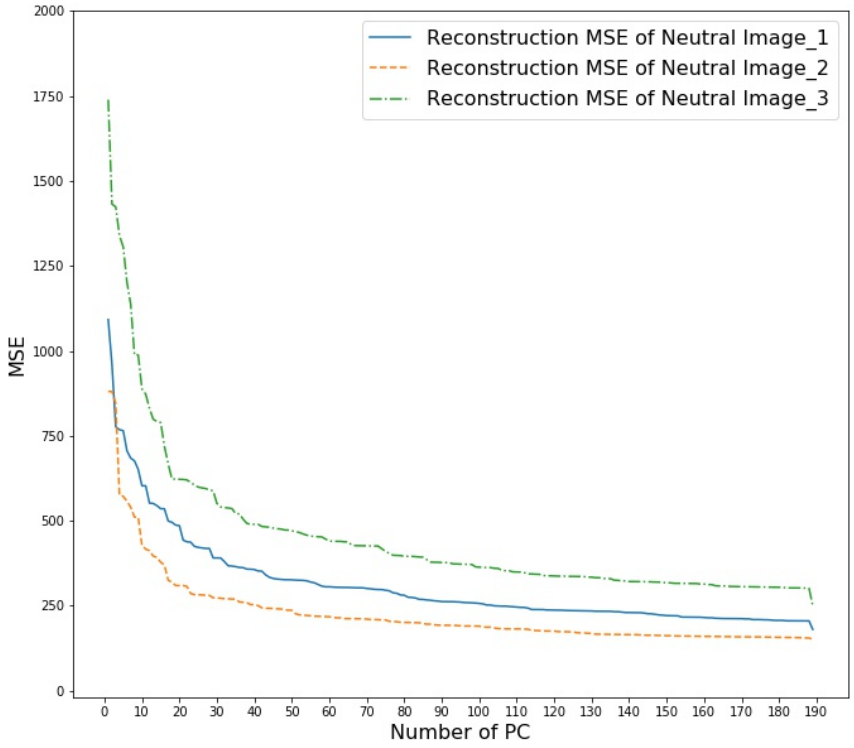

Figure 4: MSE versus number of principal components
of the other 10 individuals neutral expression image

5.2 Comment and explaination

The same as the MSE line of figure 4, 1.when the number of PCs increases, the MSE decreases. 2.When the number of PCs is large enough, the MSE would reach its bottleneck value. The bottleneck value is about 250. 3.As we can see, the bottleneck value is obviously lower than the bottleneck value of figure 3.

Because we are using neutral expression images for reconstruction in this section and those images are not included in the training dataset, so when reconstruction, the projected face would be outside the boundary of cluster of projected faces of training dataset. This results in the the bottleneck value. However, the projected face is nearer to the cluster, comparing with the projected face of smiling image. This is why the the bottleneck value of this section is lower than the bottleneck value of figure 3.

6 Reconstruction result of non-human image

6.1 Reconstruction result

We use an apple image to perform reconstruction. The reconstruction result is shown in figure 5.

(a) Original image (b) Reconstruction using all PCs (c) Reconstruction plot in 'jet' mode

Figure 5: Reconstruction result of apple image using all PCs

6.2 Comment and explaination

Figure 5(b) display in 'gray' colormap and Figure 5(c) display in 'jet' colormap. We are using all PCs generated from neutral expression image data. So when reconstruction, it will force the result to be like a face, just as we can see in figure 5(b). However, if we take a look at figure 5(c), the blue area corresponds to the 'apple' area in original image. The yellow and red area corresponds to the background in original image. So the reconstruction actually save much information of the original image, especially those edges. This means that the PCs we generated use much feature from edges and fringes.

7 Reconstruction result of rotated image

7.1 Reconstruction result

In this section, we randomly select one face image from 190 individuals neutral expression images, rotate it with different degrees, from -90 to 90. And then, we reconstruct the rotated images using all PCs. Figure 6 shows some examples of rotated images, as well as the original image.

(a) Rotate with -45 degree (b) Original image (c) Rotate with 45 degree

Figure 6: Selected image and rotated images

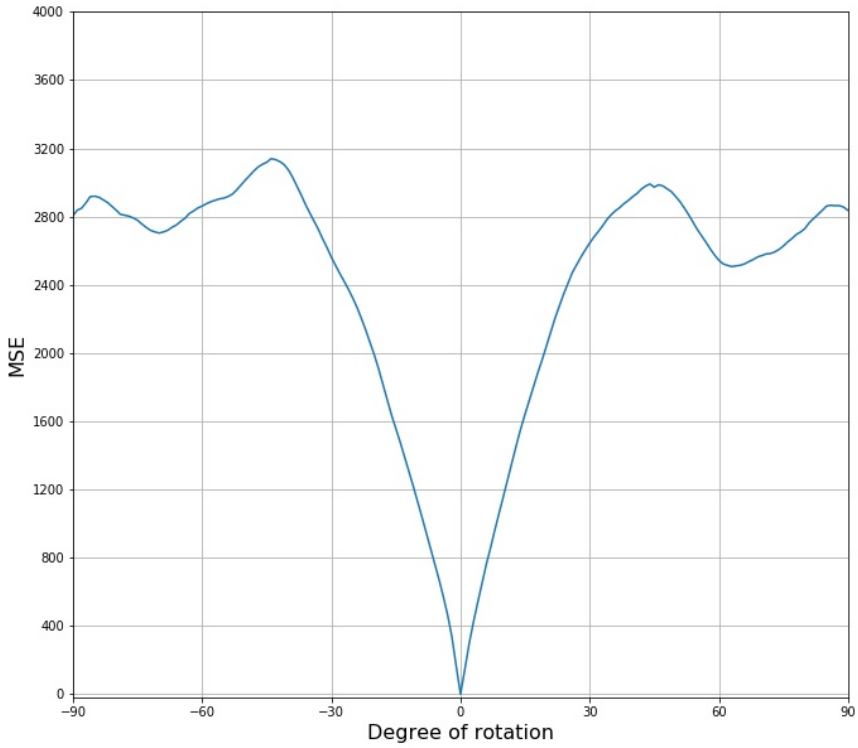

Figure 7: MSE versus different rotate degrees

7.2 Comment and explaination

As shown in figure 7, when rotation degree increases, MSE would increase. When the degree reaches about 40 or -40, MSE value steps into fluctuation. It is not surprising because the rotated image is actually a new face image. In pratice, when we are doing face recognition, we should consider the effect of roration. Because just as we see in figure 7, even a small rotation degree would result in large MSE.

References

[1] Matthew A Turk and Alex P Pentland. Face recognition using eigenfaces. In *Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 586–591. IEEE, 1991.

[2] Matthew Turk and Alex Pentland. Eigenfaces for recognition. volume 3, pages 71–86. MIT Press, 1991.

Appendix

The following is the Matlab code.

eigenface.py

```
# Some general setup
import os
import random
import numpy as np
```

```
import matplotlib.pyplot as plt
from data_utils import load_image

# This is a bit of magic to make matplotlib figures appear inline in the notebook
# rather than in a new window.
%matplotlib inline
plt.rcParams['figure.figsize'] = (11.0, 10.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'

# for auto-reloading extenrnal modules
%load_ext autoreload
%autoreload 2

# Load path of dataset
dataset_path = 'dataset/'

# Cleaning up variables to prevent loading data multiple times (which may cause memory issue)
try:
    del X_neutral, X_smile
    del original_size
    print('Clear previously loaded data.')
except:
    pass

# Function from data_utils.py
X_neutral, X_smile, original_size = load_image(dataset_path)

# Check and print out the size of the training data
print('Training data of neutral: ', X_neutral.shape)
print('Training data of smiling: ', X_smile.shape)
print('Original size of image: ', original_size)
# print(X_neutral[1,:])

# Visualize the first individual pictures from the dataset.
plt.subplot(1,2,1)
plt.imshow(X_neutral[0,:].reshape(original_size))
plt.axis('off')
plt.title('Neutral Expression')
plt.subplot(1,2,2)
plt.imshow(X_smile[0,:].reshape(original_size))
plt.axis('off')
plt.title('Smiling Facial Expression')

plt.show()
plt.savefig('image/image_show.jpg')

# Compute the mean face and normalize every images in training data
# For better computation performance, we will use broadcasting in numpy. If you don't know it, please refer to the following link:
# http://cs231n.github.io/python-numpy-tutorial/#numpy-broadcasting

# Compute the mean face
mean_neutral = np.mean(X_neutral, axis=0)
mean_smile = np.mean(X_smile, axis=0)
# Visualize the mean face
plt.subplot(1,2,1)
plt.imshow(mean_neutral.reshape(original_size))
plt.axis('off')
plt.title('Mean Face of Neutral Expression')
plt.subplot(1,2,2)
plt.imshow(mean_smile.reshape(original_size))
plt.axis('off')
plt.title('Mean Face of Smiling Facial Expression')
```

```
plt.show()
plt.savefig('image/meanface_show.jpg')

# Normalize the training data
X_neutral_nor = X_neutral - mean_neutral # shape of (200, 31266)
X_smile_nor = X_smile - mean_smile # shape of (200, 31266)
# Randomly select some images for visualization
idxs = np.random.choice(200, 8, replace=False)
print('Randomly selected normalized images from neutral expression for
    visualization')
for i,idx in enumerate(idxs):
    plt.subplot(2, 4, i+1)
    plt.imshow(X_neutral_nor[idx,:].reshape(original_size))
    plt.axis('off')
plt.show()

print('Randomly selected normalized images from smile expression for visualization
    ')
for i,idx in enumerate(idxs):
    plt.subplot(2, 4, i+1)
    plt.imshow(X_smile_nor[idx,:].reshape(original_size))
    plt.axis('off')
plt.show()

# Compute eigenvectors and eigenvalues. The column v[:,i] is the eigenvector and
    each eigenvector is unitary.
evalue_neutral, evector_neutral = np.linalg.eig(np.dot(X_neutral_nor[range(0,190)
    ,:],X_neutral_nor[range(0,190),:].T))
evalue_smile, evector_smile = np.linalg.eig(np.dot(X_smile_nor[range(0,190),:],
    X_smile_nor[range(0,190),:].T))

# As usual, check and print out the size of the matrix.
print('Eigenvalues of neutral training data: ', evalue_neutral.shape)
print('Eigenvectors of neutral training data: ', evector_neutral.shape)
print('Eigenvalues of smile training data: ', evalue_smile.shape)
print('Eigenvectors of smile training data: ', evector_smile.shape)

# Sort eigenvalues and corresponding eigenvectors
idx = evalue_neutral.argsort()[::-1]
evalue_neutral = evalue_neutral[idx]
evector_neutral = evector_neutral[:,idx]

idx = evalue_smile.argsort()[::-1]
evalue_smile = evalue_smile[idx]
evector_smile = evector_smile[:,idx]

plt.stem(np.sqrt(evalue_neutral),use_line_collection=True)
my_x_ticks = np.arange(0, 191, 10)
plt.xticks(my_x_ticks)
plt.savefig('image/eigenvalues_neutral.jpg')
plt.show()

# plt.stem(evalue_smile,use_line_collection=True)
# plt.savefig('image/eigenvalues_smile.jpg')
# plt.show()

# #print(evalue_neutral)

# Compute eigenfaces
eigenface_neutral = np.dot(X_neutral_nor[range(0,190),:].T,evector_neutral)
eigenface_smile = np.dot(X_smile_nor[range(0,190),:].T,evector_smile)

# We should normalize the eigenfaces, making every eigenfaces to be unitary
eigenface_neutral = np.divide(eigenface_neutral , np.linalg.norm(eigenface_neutral
    , axis = 0))
```

```
eigenface_smile = np.divide(eigenface_smile , np.linalg.norm(eigenface_smile , axis = 0))

# Check and print the size of each eigenface. The column is the eigenface and each eigenface is unitary.
print('Single eigenface size: ', eigenface_neutral[:,0:1].shape)
# print(eigenface_neutral[:,0:1])
# PC = eigenface_neutral[:,0:20+1].T
# print(PC.shape)

def MSE(ground_truth, reconst):
    """ This function is for compute the MSE """
    mse = ((ground_truth - reconst) ** 2).mean(axis = 0)
    return mse

def Reconstruct_MSE(num_PC, select_img, efaces, evalues):
    """ Reconstruct image and plot MSE vs different numbers of PC """
    PC = efaces[:,0:num_PC+1].T
    W =  np.dot(PC, select_img)
    reconst = np.dot(PC.T, W)
    # Compute the MSE
    mse = MSE(select_img, reconst)
    return mse

# Now we begin to reconstruct the training images using different numbers of PC. Compute the MSE and plot the curve vs number of PC.
# Randomly select 3 different images and reconstruct.

num_image_select = 3
idx = np.random.choice(range(0,190), num_image_select, replace=False)
select_img = X_neutral_nor[idx,:].T
print(idx)

# In this cell, I write in a naive way. It is not efficient but only for easy writing. It's better to try matrix way.
# Define 3 list to store MSE result
mse_1 = []
mse_2 = []
mse_3 = []
for num_PC in range(1,190):
    mse = Reconstruct_MSE(num_PC, select_img, eigenface_neutral, evalue_neutral)
    mse_1.append(mse[0])
    mse_2.append(mse[1])
    mse_3.append(mse[2])

plt.plot(range(1,190), mse_1, linestyle='-', label="Reconstruction MSE of Image_1")
plt.plot(range(1,190), mse_2, linestyle='--', label="Reconstruction MSE of Image_2")
plt.plot(range(1,190), mse_3, linestyle='-.', label="Reconstruction MSE of Image_3")
plt.ylim(-20,2000)
plt.legend(loc="upper right", fontsize=16)
plt.ylabel("MSE", fontsize=16)
plt.xlabel("Number of PC", fontsize=16)
my_x_ticks = np.arange(0, 191, 10)
plt.xticks(my_x_ticks)
plt.savefig('image/mse_neutral.jpg')

# The same work as above but on smile training data
num_image_select = 3
idx = np.random.choice(range(0,190), num_image_select, replace=False)
select_img = X_smile_nor[idx,:].T

# Define 3 list to store MSE result
```

```
mse_1 = []
mse_2 = []
mse_3 = []
for num_PC in range(1,190):
    mse = Reconstruct_MSE(num_PC, select_img, eigenface_neutral, evalue_neutral)
    mse_1.append(mse[0])
    mse_2.append(mse[1])
    mse_3.append(mse[2])

plt.plot(range(1,190), mse_1, linestyle='-', label="Reconstruction MSE of Smile Image_1")
plt.plot(range(1,190), mse_2, linestyle='--', label="Reconstruction MSE of Smile Image_2")
plt.plot(range(1,190), mse_3, linestyle='-.', label="Reconstruction MSE of Smile Image_3")
plt.ylim(-20,2500)
plt.xlim(-10,200)
plt.legend(loc="upper right", fontsize=16)
plt.ylabel("MSE", fontsize=16)
plt.xlabel("Number of PC", fontsize=16)
my_x_ticks = np.arange(0, 191, 10)
plt.xticks(my_x_ticks)
plt.savefig('image/mse_smile.jpg')

# Reconstrcut three of the other 10 individuals neutral expression image
num_image_select = 3
idx = np.random.choice(range(191,200), num_image_select, replace=False)
select_img = X_neutral_nor[idx,:].T
print(idx)

mse_1 = []
mse_2 = []
mse_3 = []
for num_PC in range(1,190):
    mse = Reconstruct_MSE(num_PC, select_img, eigenface_neutral, evalue_neutral)
    mse_1.append(mse[0])
    mse_2.append(mse[1])
    mse_3.append(mse[2])

plt.plot(range(1,190), mse_1, linestyle='-', label="Reconstruction MSE of Neutral Image_1")
plt.plot(range(1,190), mse_2, linestyle='--', label="Reconstruction MSE of Neutral Image_2")
plt.plot(range(1,190), mse_3, linestyle='-.', label="Reconstruction MSE of Neutral Image_3")
plt.ylim(-20,2000)
plt.legend(loc="upper right", fontsize=16)
plt.ylabel("MSE", fontsize=16)
plt.xlabel("Number of PC", fontsize=16)
my_x_ticks = np.arange(0, 191, 10)
plt.xticks(my_x_ticks)
plt.savefig('image/mse_other_neutral.jpg')

# Read an apple image and try to reconstruct it
img_apple = plt.imread(dataset_path + 'apple1_gray.jpg')
# Show the apple image
plt.imshow(img_apple)
plt.axis('off')
plt.show()
# Show the cropped apple image
img_apple_crop = img_apple[5:198,5:167] #(5, 5, 166, 197)
plt.imshow(img_apple_crop)
plt.axis('off')
plt.savefig('image/apple_crop.jpg')
plt.show()
```

```
#print(img_apple_crop.shape)
W =  np.dot(eigenface_neutral.T, img_apple_crop.flatten()-mean_neutral)
reconst = np.dot(eigenface_neutral, W) + mean_neutral
plt.imshow(reconst.reshape(original_size))
plt.axis('off')
plt.savefig('image/apple_reconstruct.jpg')
plt.show()
plt.imshow(reconst.reshape(original_size),cmap='jet')
plt.axis('off')
plt.savefig('image/apple_reconstruct_jet.jpg')
plt.show()

from PIL import Image
# Rotate one of 190 individualsneutral expression image with different degrees and
    try to reconstruct it using all PCs
# num_image_select = 1
# idx = np.random.choice(range(0,190), num_image_select, replace=False)
idx = 73 # Select one of the image
filepath = os.path.join(dataset_path, 'Part_%d' % (1, ))
select_img = Image.open(filepath + '/%da.jpg' % (idx, ))
plt.imshow(select_img)
plt.axis('off')
plt.savefig('image/select_image.jpg')

rotated = select_img.rotate(-45)
plt.imshow(rotated)
plt.axis('off')
plt.savefig('image/select_image_rotate-45.jpg')
plt.show()
rotated = select_img.rotate(45)
plt.imshow(rotated)
plt.axis('off')
plt.savefig('image/select_image_rotate45.jpg')
plt.show()

mse_rotate = []
for degree in range(-90,91):
    rotate_img = select_img.rotate(degree)
    W =  np.dot(eigenface_neutral.T, np.array(rotate_img).flatten()-mean_neutral)
    reconst = np.dot(eigenface_neutral, W)
    mse = MSE(np.array(rotate_img).flatten()-mean_neutral, reconst)
    mse_rotate.append(mse)

plt.plot(range(-90,91), mse_rotate, linestyle='-')
plt.ylim(-20,4000)
plt.xlim((-90,90))
my_x_ticks = np.arange(-90, 91, 30)
plt.xticks(my_x_ticks)
my_y_ticks = np.arange(0, 4001, 400)
plt.yticks(my_y_ticks)
plt.ylabel("MSE", fontsize=16)
plt.xlabel("Degree of rotation", fontsize=16)
plt.grid(True)
plt.savefig('image/mse_rotate.jpg')
```

data_utils.py

```
# As usual, some setup code
import numpy as np
import os
import matplotlib.pyplot as plt

def load_image(ROOT):
    """ Load the image dataset from disk """
    xn = []
```

```
    xs = []
    for b in range(1,3):
        path = os.path.join(ROOT, 'Part_%d' % (b, ))
        X, Y, original_size = load_part(path,b)
        xn.append(X)
        xs.append(Y)
    del X, Y
    XN = np.concatenate(xn)
    XS = np.concatenate(xs)
    return XN, XS, original_size

def load_part(filepath,num):
    """ load neutral and smiling image seperately """
    X = []
    Y = []
    original_size = []
    for i in range((num-1)*100+1,num*100+1):
        # Read image of neutral
        img_1 = plt.imread(filepath + '/%da.jpg' % (i, ))
        original_size = np.shape(img_1)
        X.append(np.array(img_1).flatten())

        # Read image of smiling
        img_2 = plt.imread(filepath + '/%db.jpg' % (i, ))
        Y.append(np.array(img_2).flatten())
    return X, Y, original_size
```