|  |
| --- |
|  |
| **PCA Project** |
| ECE 269 |

|  |
| --- |
| Yening Dong  PID: A53314848 |

**Preface**

In the PCA Project, I use the Python Opencv to solve all the problems, so I have to state certain features of Opencv.

Opencv takes all variable as row vectors, so I organize the face images in training set as a row vector rather than a column vector in any matrix.

Therefore, my code and MSE results may be different from the majority.

**Contents**

[**Question (a)** 3](#_Toc26055906)

[**Question (b)** 4](#_Toc26055907)

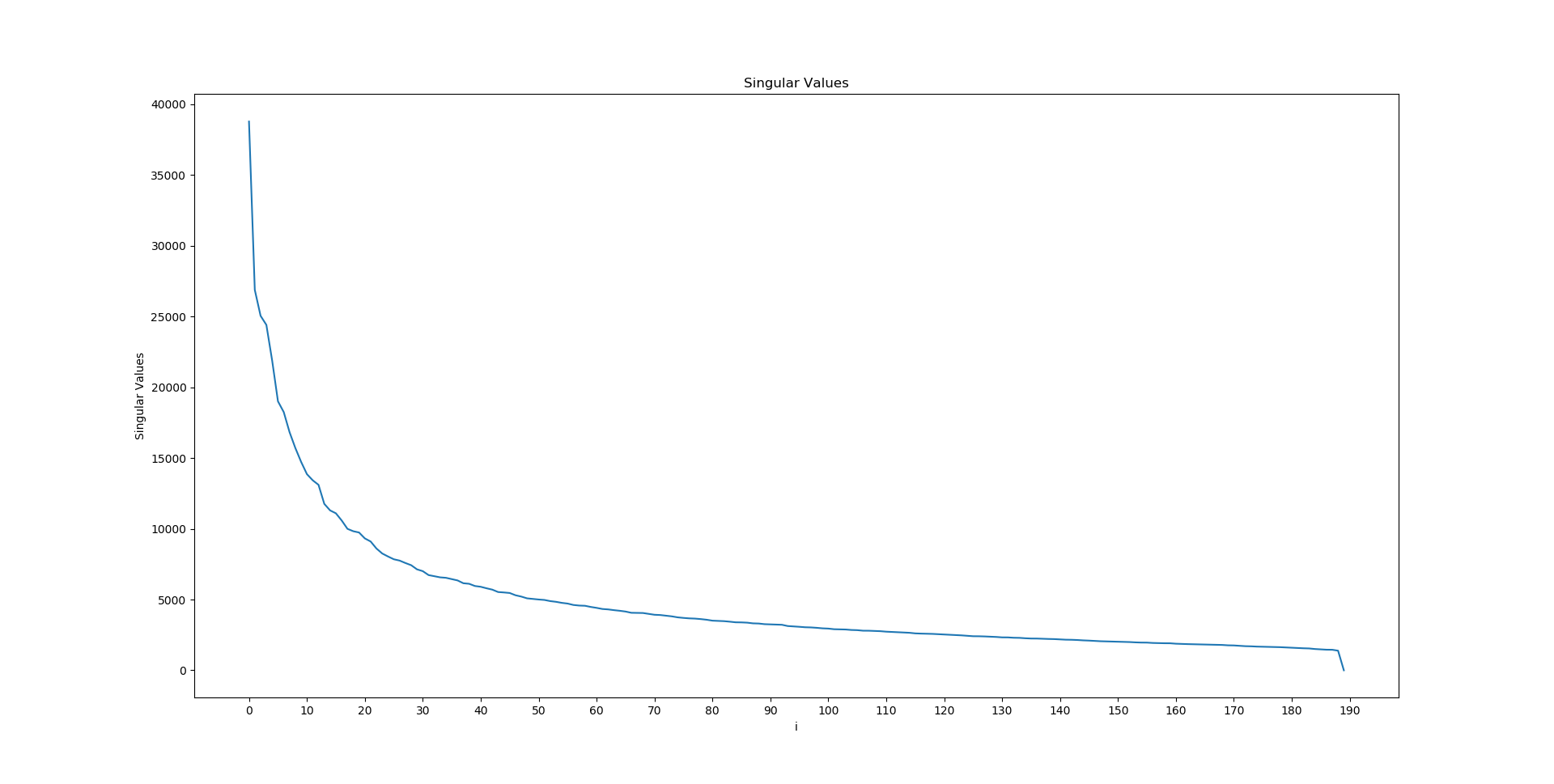
[**Question (c)** 6](#_Toc26055908)

[**Question (d)** 8](#_Toc26055909)

[**Question (e**) 10](#_Toc26055910)

[**Question (f)** 12](#_Toc26055911)

[**Code** 15](#_Toc26055912)

**Question (a)**

**Comment:**

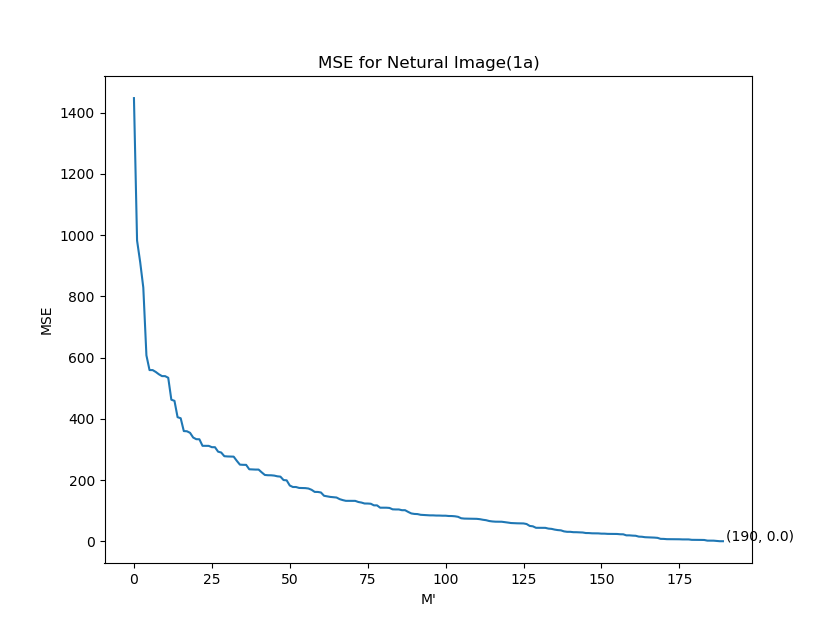
I choose the eigenvectors corresponding to the first 150 eigenvalues. From the singular values graph, we can see the singular values decrease dramatically at the beginning with high gradients, but the gradient seems to be steady after 150th singular value, and the singular values are corresponding to each eigenvalues, their relation is

so the eigenvalues corresponding to the last 40 singular values have less influence on the following process. Above all, the eigenvectors corresponding to the first 150 eigenvalues seem to have the biggest influence on the following process.

**Question (b)**

Face Images(1a) Reconstructed Face Image(1a)



**Comment:**

In the images:

1. From the images, we can see the reconstructed one is much same as the input one, the result satisfies the theory expectation. The face image(1a) is in the training set, so the 1a vector must be in the linear combination of eigenvectors, and we can restore 1a by eigenvectors.

In the MSE graph:

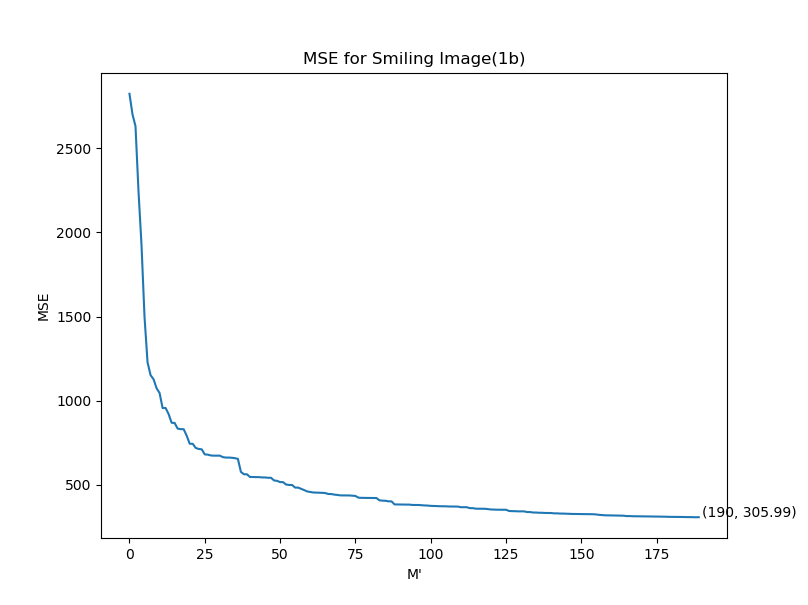
1. The MSE decreases as the number of M’ increases, and the gradients become smaller, especially after 150thM’ which satisfies the conclusion I get from SVD graph in question (a).
2. At last, the MSE is close to the 0, so we are supposed to reconstruct the same image when we use all 190 eigenvectors.

Above all, more M’ means less MSE, but the influence becomes smaller when M’ increases. And question (b) belongs to the first possibility mentioned in the paper: near face space and near a face class.

**Question (c)**

Face Image(1b) Reconstructed Face Image



**Comment：**

In the Images：

1. The reconstructed face image looks like the input, but it is a little blurred, compared with the result of question (b).
2. Because the training set is full of neutral expression, the smiling expression in reconstructed face image isn’t obvious.

In the MSE graph:

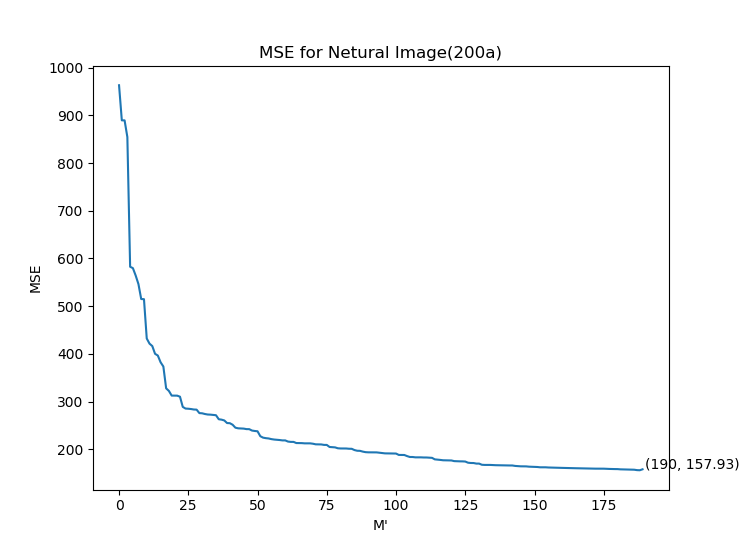
1. The same conclusion as question (b): The MSE decreases as the number of M’ increases, and the gradients become smaller, especially after 150thM’ which satisfies the conclusion I get from SVD graph in question (a). Furthermore, MSE in question (c) is always higher than this in question (b), in terms of the same number of M’.
2. At last, the MSE is close to 305 rather than 0, which is different from question (b). It’s because that the input image is smiling expression, but all eigenfaces from the training set are netural expressions.

Above all, more M’ means less MSE, but the influence becomes smaller when M’ increases. And question (c) belongs to the third possibility mentioned in the paper: distant from face space and near a face class.

**Question (d)**

Face Image (200 a) Reconstructed Face Image



**Comment：**

In the Images：

1. The reconstructed face image looks like the input, but certain details like moles disappear, so the result of question (d) is better than that of question (c), but worse than that of question (b).
2. Because the training set is full of neutral expression, the netural expression retores well, although the input image isn’t in the training set.

In the MSE graph:

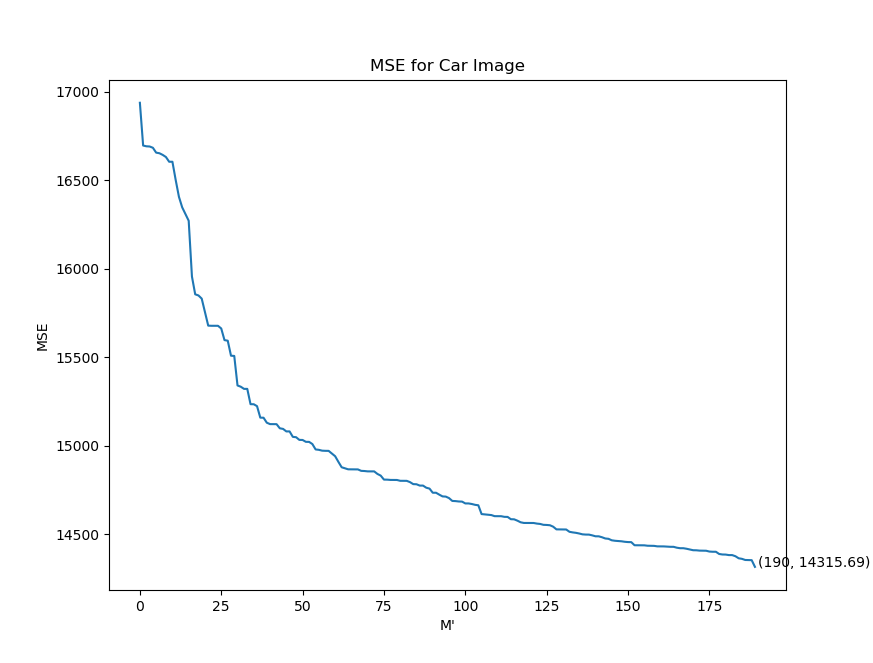
1. The same conclusion as question (b): The MSE decreases as the number of M’ increases, and the gradients become smaller, especially after 150thM’ which satisfies the conclusion I get from SVD graph in question (a). Furthermore, MSE in question (c) is always higher than this in question (b) and lower than that in question (c) in terms of the same number of M’.

MSE(b) < MSE(d) < MSE(c)

1. When M’=190, the MSE is close to 158 rather than 0, which is different from question (b).

It’s because that the input image is out of the training set, we cann’t retore all information of it by eigenfaces, although the input image is netural expression.

Above all, more M’ means less MSE, but the influence becomes smaller when M’ increases. And question (d) belongs to the second possibility mentioned in the paper: near face space but not near a known face class.

**Question (e**)



Input Image

Reconstructed Image

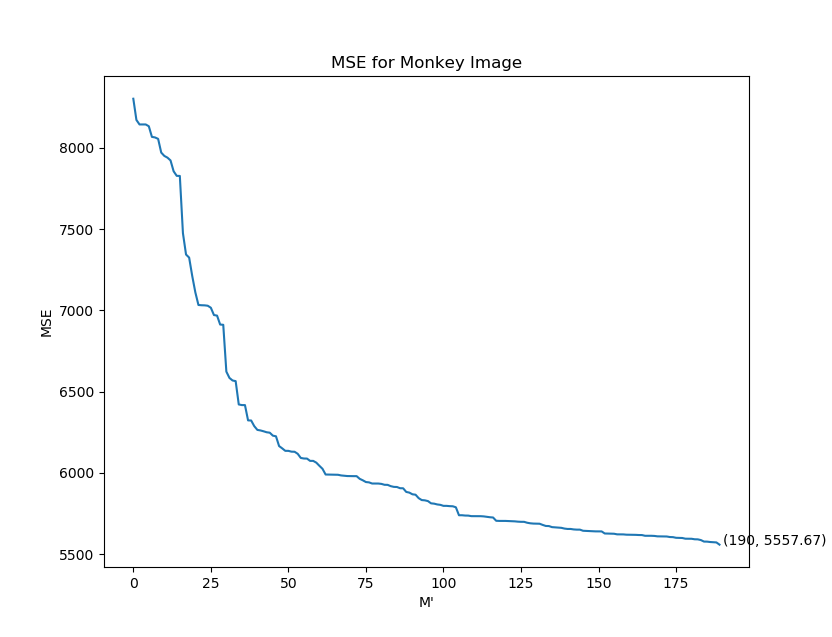


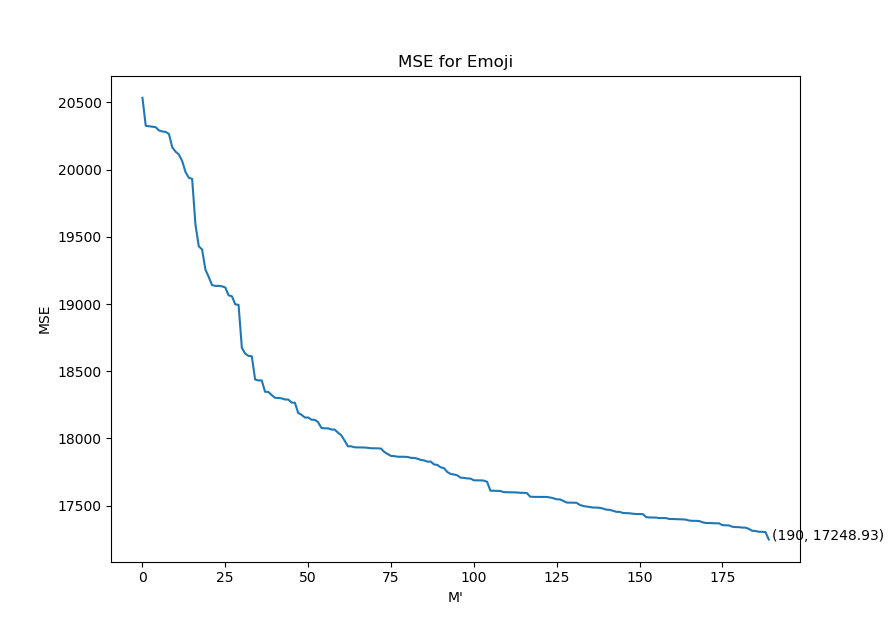
Reconstructed Image

Reconstructed Image

Input Image

Input Image





**Comment:**

In question (e), I choose three different kinds of images: car image, monkey face image and emoji image.

In the images:

1. Whatever car, monkey face or emoji, none of them is human expression, so the eigenfaces have no effect on reconstructing these input images. The reconstructed images don’t look like the corresponding input image at all.
2. In the mid of the reconstructed car image, there is a bright area, it’s the location of the car in input image. Because there is no car image in training set, we cann’t reconstruct its feature, but we can show its location.

In the MSE graphs:

1. The MSE in question (e) are bigger than the MSE in question (b,c,d), because the input images aren’t human faces.
2. Compared with the three MSE graphs in question (e), we can see

Firstly, emoji isn’t a real item and there are little information we get from the simple emoji, so it’s hard to reconstruct the emoji, using the eigenfaces which have no connection to emoji.

Secondly, car is a real item which contains enough information, but there is no car image in the training set, so we can’t reconstruct the car by eigenfaces, although we can get the all information from the input image.

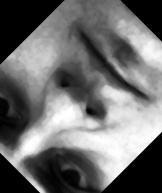
Thirdly, monkey face has the lowest MSE among three items, because monkey face still is a face consisted of eyes, nose, mouth and so on. However, its MSE is higher than the human face obviously, for there still are many differences between human and monkey.

**Question (f)**

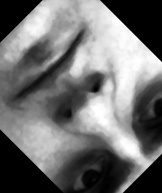
In question (f), I choose five different degrees: 45°, 90°, 135°, 180° and 225°, the result of each one show in the following.

Face Image(1a) reconstructed image rotated 45° reconstructed image

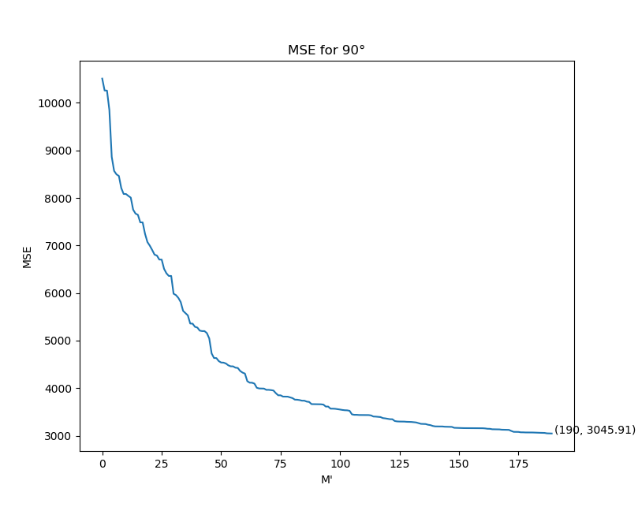
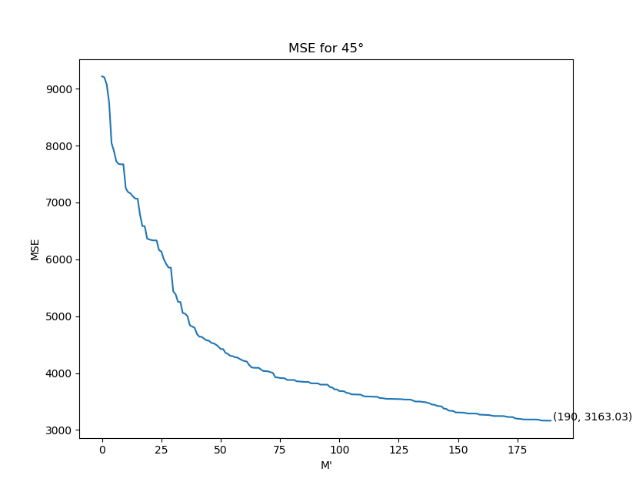
  

rotated 90° reconstructed image rotated 135° reconstructed image

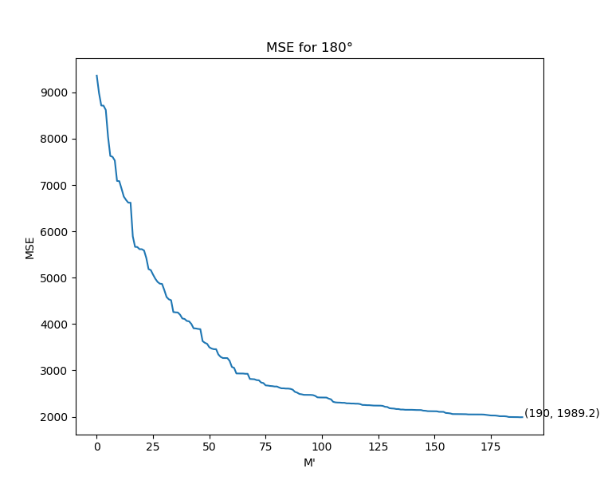
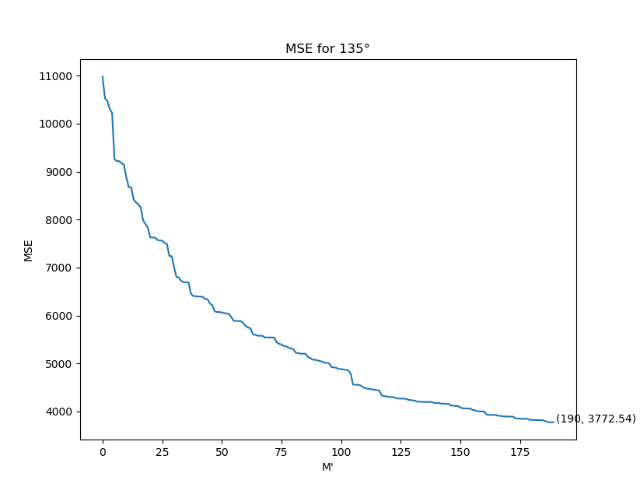
 

rotated 180° reconstructed image rotated 225° reconstructed image

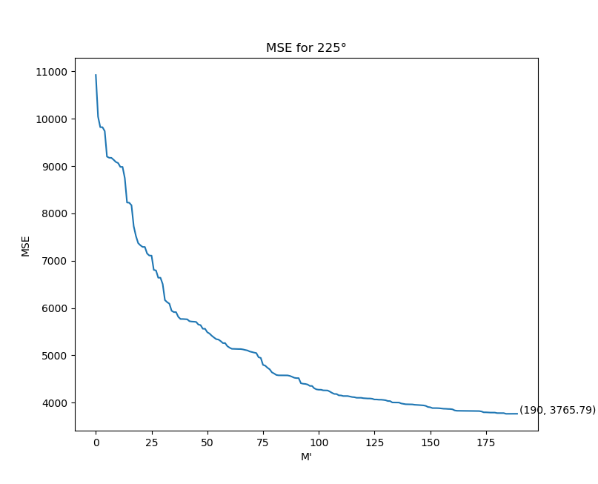
The following are the MSE graphs corresponding to the reconstructed iamges



MSE graph of 45° MSE graph of 90°



MSE graph of 135° MSE graph of 180°



MSE graph of 225°

**Comment:**

In question (f), I choose five different degrees, and the reconstructed images and MSE vary,

In the images:

1. Whatever a degree is, the reconstructed image doesn’t look like the input one, as long as the image has been rotated. The inforamtion of images will be changed after rotateing, but the training set still is same, so the eigenfaces have less effect on reconstructing these input images. As a result, the reconstructed images don’t look like the corresponding input images.
2. After rotating and croping, four corners of certain images are all black, and the four corners of corresponding reconstructed images seem to be more dark.

In the MSE graphs:

1. The MSE in question (f) are bigger than the MSE in question (b,c,d), but less than the MSE in question (e), because rotated face images are at least kind of human faces rather than other items to some extent.
2. In the question (f)’s MSE graphs, the MSE of image rotated 180° has the lowest value, because the image is full of the inversed face without any black blank.

**Code**

# The following codes are modified from learn opencv  
# The code uses the face images as row vector rather than column vector  
from \_\_future\_\_ import print\_function  
import os  
import sys  
import cv2  
import numpy as np  
from sklearn.metrics import mean\_squared\_error as mse  
import matplotlib.pyplot as plt  
import pandas as pd  
  
# Read FaceImages from the directory  
def readImages(path):  
 print("Reading FaceImages from " + path, end="...")  
 # Create array of array of FaceImages.  
 images\_set = []  
 # List all files in the directory and read points from text files one by one  
 # os.listdir(), return a list including all files and documents in the specific files  
 for filePath in sorted(os.listdir(path)):  
  
 # Add to array of FaceImages  
 imagePath = os.path.join(path, filePath)  
 im = cv2.imread(imagePath,flags=cv2.IMREAD\_GRAYSCALE)  
 # imread 的output is (192,163,3)的np.array, dtype = uint8  
  
 if im is None:  
 print("image:{} not read properly".format(imagePath))  
 else:  
 # Convert image to floating point  
 # I have no idea about this operation, but all the sample code on the Internet have the operation  
 im = np.float32(im)  
 # Add image to list  
 images\_set.append(im)  
  
 print(str(sum\_faceImages) + " files read.")  
 return images\_set  
  
  
# Create original matrix from a list of FaceImages  
# cv2.PCACompute is design defaultly for row vector  
def createDataMatrix(images):  
 # The Input is not truely original face FaceImages  
 # these face FaceImages are dealt, when we read them (function: 'readImages')  
 print("Creating data matrix", end=" ... ")  
 '''   
 Allocate space for all FaceImages in one data matrix.  
 The size of the data matrix is ( w \* h \* 3, numImages )  
 w = width of an image in the dataset.  
 h = height of an image in the dataset.  
 3 is for the 3 color channels.  
 '''  
 # In the project, all face FaceImages are grey FaceImages with only 1 color channel  
 # sz[2] always be 1

# len() is calculating the number of face FaceImages in the training set  
 numImages = len(images)  
 # Create a matrix, according to 1st face image,  
 # because shapesize of all face FaceImages in the training set are same, so only use 1st face  
 sz = images[0].shape  
  
 # create a matrix to describe all face FaceImages in the training set  
 # numImages is the number of rows (vector), each face FaceImages will be described by a row vector  
 # sz[0]\*sz[1]\*sz[2] is the dimension of the row vector  
 matrix\_face = np.zeros((numImages, sz[0] \* sz[1]), dtype=np.float32)  
 for i in range(0, numImages):  
 # flatten() is numpy.ndarray.flatten, which convert arrays into one-dimension array  
 # use row to make up one-dimension numpy array  
 # The function convert each face image into one-dimension row vector (array)  
 image = images[i].flatten()  
 # fill these one-dimension row vector into the matrix created before.  
 matrix\_face[i, :] = image  
  
 print("DONE")  
 return matrix\_face  
  
# Add the weighted eigenFaces to the mean (row vector)  
def createNewFace(mean,dirname,num\_eigenfaces,eigenVectors):  
  
 print("Loading FaceImages from " + dirname, end="...")  
 # Start with the mean image(come from cv2.PCACompute)  
 output = mean  
  
 new\_faceimage = cv2.imread(dirname,flags=cv2.IMREAD\_GRAYSCALE)  
   
# rotated image  
 # height, width = new\_faceimage.shape[:2]  
 # center = (width / 2, height / 2)  
 # rot\_mat = cv2.getRotationMatrix2D(center, 225, 1)  
 # new\_faceimage = cv2.warpAffine(new\_faceimage, rot\_mat, (new\_faceimage.shape[1], new\_faceimage.shape[0]))  
  
 # # resieze shape of input image  
 # new\_faceimage = cv2.resize(new\_faceimage,(193,162))  
  
 new\_faceimage = np.float32(new\_faceimage)  
 new\_faceimage = new\_faceimage.flatten()  
 # print('size of new\_faceimage is', np.shape(new\_faceimage))  
  
 for i in range(0, num\_eigenfaces):  
 weight = (new\_faceimage-mean).dot(eigenVectors[i].T)  
  
 # new faceimages (row vector) with weighted  
 output = np.add(output, eigenVectors[i] \* weight)  
  
 print("DONE")  
 return output

def CalculateMSE(mean,dirname,num\_eigenfaces,eigenVectors):  
  
 print("Calculating MSE from" + dirname, end="...")  
 new\_faceimage = cv2.imread(dirname,flags=cv2.IMREAD\_GRAYSCALE)  
  
 # # for resize no human face image  
 # new\_faceimage = cv2.resize(new\_faceimage,(193,162))  
   
# rotate image  
 # height, width = new\_faceimage.shape[:2]  
 # center = (width / 2, height / 2)  
 # rot\_mat = cv2.getRotationMatrix2D(center, 225, 1)  
 # new\_faceimage = cv2.warpAffine(new\_faceimage, rot\_mat, (new\_faceimage.shape[1], new\_faceimage.shape[0]))  
  
 new\_faceimage = np.float32(new\_faceimage)  
 new\_faceimage = new\_faceimage.flatten()  
 new\_faceimage\_reshape = new\_faceimage.reshape((1, 31266))  
  
 mean\_SE = dict()  
 for i in range(1,num\_eigenfaces+1):  
  
 output = mean  
  
 for j in range(0, i):  
 weight = (new\_faceimage - mean).dot(eigenVectors[j].T)  
  
 output = np.add(output, eigenVectors[j] \* weight)  
  
 mean\_SE1=mse(new\_faceimage\_reshape,output)  
 # print(MSE1)  
 mean\_SE[i-1]=mean\_SE1  
  
 print("DONE")  
 print('The number of MSE is', len(mean\_SE))  
  
 mean\_SE = pd.DataFrame.from\_dict(mean\_SE, orient='index', columns=['values'])  
 plt.plot(mean\_SE)  
 plt.show()  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
  
# Number of EigenFaces  
 num\_eigenfaces = 190  
  
# Directory containing FaceImages  
 dirName = "Netural\_Images"  
  
# Read FaceImages  
 images = readImages(dirName)  
  
# Size of FaceImages, in order to resize all the following row vector into (1,32166)  
 sz = images[0].shape  
  
# Create original faceimages matrix for PCA, each faceimage is a row vector  
 matrix\_faces = createDataMatrix(images)  
 print('Shape of matrix\_faces is ', np.shape(matrix\_faces))  
 matrix\_faces\_T = matrix\_faces.T  
  
 # get the covariance matirx and mean  
 covar, mean = cv2.calcCovarMatrix(matrix\_faces, mean=None, flags=cv2.COVAR\_SCALE | cv2.COVAR\_ROWS| cv2.COVAR\_SCRAMBLED)  
 # mean\_x is same as mean\_1, cv2.COVAR\_SCALE is equal to 1/M, because the default is 1  
  
 print("shape of mean is ", np.shape(mean))  
 print("shape of covar is ", np.shape(covar))  
  
# get eigenvalues and eigenvectors  
 print("Calculating PCA ", end="...")  
 eVal, eVec = cv2.eigen(covar, True)[1:]  
 svd = np.sqrt(eVal)  
  
# to make some operations on eVec，1. （ui）T \* （A）T，2. normalize，  
# OpenCV uses row operations rather than column operations, so we must transpose the matrixs  
 eVec = cv2.gemm(eVec, matrix\_faces - mean, 1, None, 0)  
 eVec = np.apply\_along\_axis(lambda n: cv2.normalize(n,n).flat, 1, eVec)  
 print('DONE')  
  
 plt.plot(svd)  
 plt.show()  
  
# Create window for displaying Mean Face  
 cv2.namedWindow("AverageFace", cv2.WINDOW\_AUTOSIZE)  
 cv2.imshow("AverageFace", mean.reshape(sz))  
  
# load new faceimages, and use eigenvector to build new faceimage with weight  
 new\_faceimage = createNewFace(mean, '1a.jpg', num\_eigenfaces, eVec)  
 print('shape of new\_faceimage',np.shape(new\_faceimage))  
  
# show the new\_faceimage with weight  
 cv2.namedWindow('new\_faceimage', cv2.WINDOW\_AUTOSIZE)  
 cv2.imshow('new\_faceimage',new\_faceimage.reshape(sz))  
 cv2.imwrite(r'C:\Users\83414\Desktop\ECE 269\Project\225\_f.jpg',new\_faceimage.reshape(sz)\*255)  
  
 CalculateMSE(mean, '1a.jpg', num\_eigenfaces, eVec)  
  
  
 cv2.waitKey(0)

cv2.destroyAllWindows()