Face Recognition with Eigenfaces

by

Haonan Wu

N18859539

New York University

Dec, 2017

E.D Wong

Table of Contents

Chapter 1

Introduction

The Principal Component Analysis (PCA) was independently proposed by Karl Pearson (1901) and Harold Hotelling (1933) to turn a set of possibly correlated variables into a smaller set of uncorrelated variables. The idea is, that a high-dimensional dataset is often described by correlated variables and therefore only a few meaningful dimensions account for most of the information. The PCA method finds the directions with the greatest variance in the data, called principal components.

This project is to implement PCA by Python.

Chapter 2

Environment Description

- Window 10
- Virtual Studio 2015
- Python 2.7
    - (1) numpy
    - (2) matplotlib

Keep the floder '$train\_data$', '$test\_data$' and python file '$eigenfaces.py$' in the same floder and simple run

```
python eigenfaces.py
```

Chapter 3

Algorithmic Description

step 1 Let $X = \{x_1, x_2, \ldots, x_n\}$ be the training set's matrix with $x_i \in \{[0, 255]\}^d$.

d is the size of the image. In this case, $d = 45045$.

step 2 Compute the mean

$$\mu = \frac{1}{n}\sum_{i=1}^{n} x_i$$

.

step 3 Compute the the Covariance Matrix

$$S = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T$$

. The size of S in this case is 9×9.

step 4 Compute the eigenvalues λ_i and eigenvectors v_i of

$$Sv_i = \lambda_i v_i, i = 1, 2, \ldots, n$$

step 5 Order the eigenvectors descending by their eigenvalue. The k principal components are the eigenvectors corresponding to the k largest eigenvalues.

In this case, instead of using the integer k to truncate the eigenvalue, I use the propotion of λ_i in $\sum \lambda_i$ to truncate the eigenvalue. And the threshold is 0.7.

In other word, if the sum of kept eigenvalues is larger than 0.7, we throw the rest one.

step 6 The k principal components of the test set x are then given by:

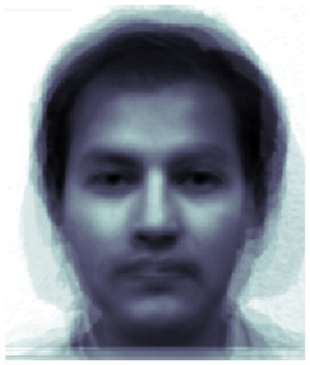
$$y = W^T(x - \mu)$$

where $W = (v_1, v_2, \ldots, v_k)$.

Finding the nearest neighbor between the projected training images and the projected test image.

Chapter 4

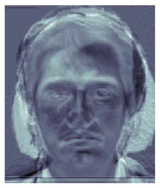
Result



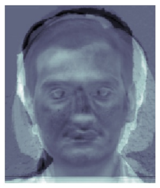
The mean face.

(a) Eigen Faces 0

(b) Eigen Faces 1

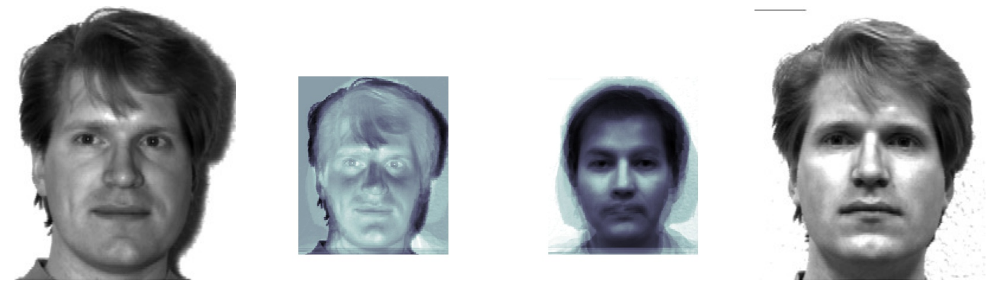


(c) Eigen Faces 2



(d) Eigen Faces 3

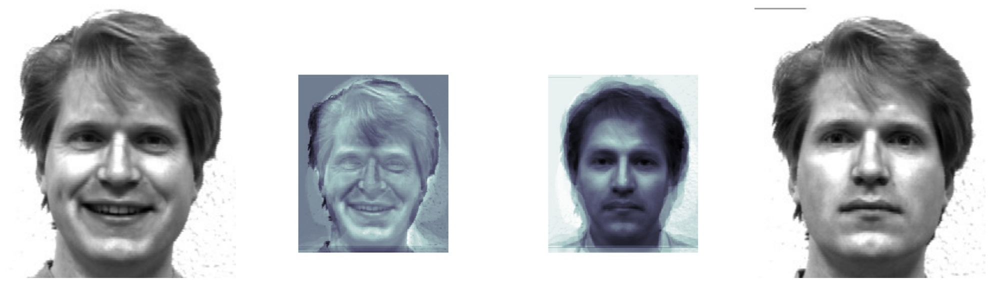
Figure 4.1: Eigen Faces



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.2: Label 1

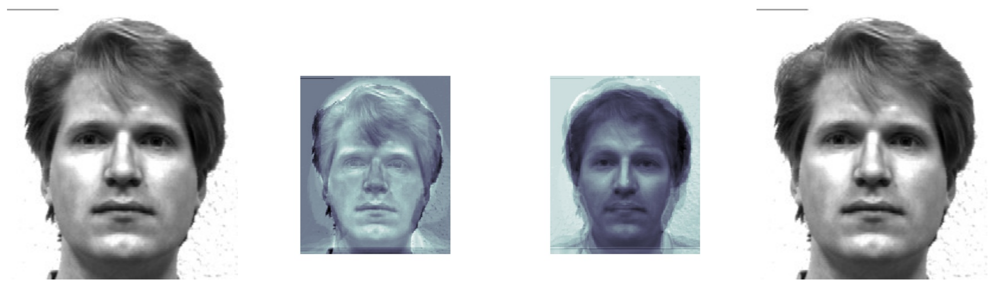
The distance from the projected test image to its nearest neighbor is 6456.452342.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.3: Label 1

The distance from the projected test image to its nearest neighbor is 2991.545346.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.4: Label 1

The distance from the projected test image to its nearest neighbor is 0.0.

For the first face, all three test image are detected correctly. But if the threshold for the k largest eigenvalues is larger than 0.8, the centerlight image will be detected incorrectly.

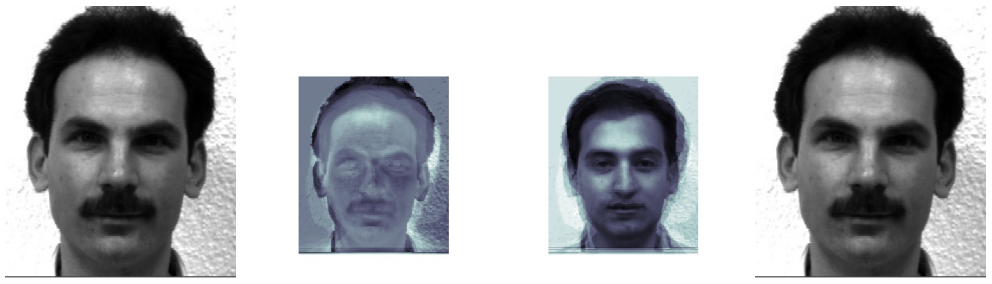

(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.5: Label 2

The distance from the projected test image to its nearest neighbor is 0.0.

For 2nd face, the test image is also the train image, it should be correctly.

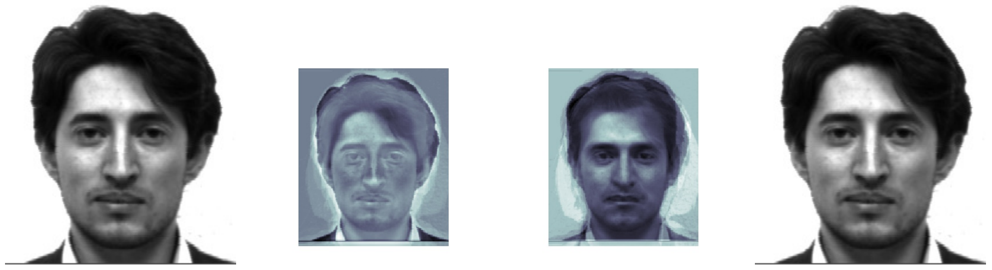
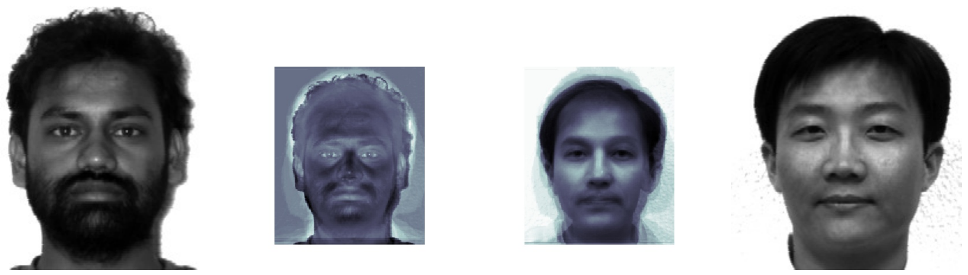

(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.6: Label 3

The distance from the projected test image to its nearest neighbor is 0.0.

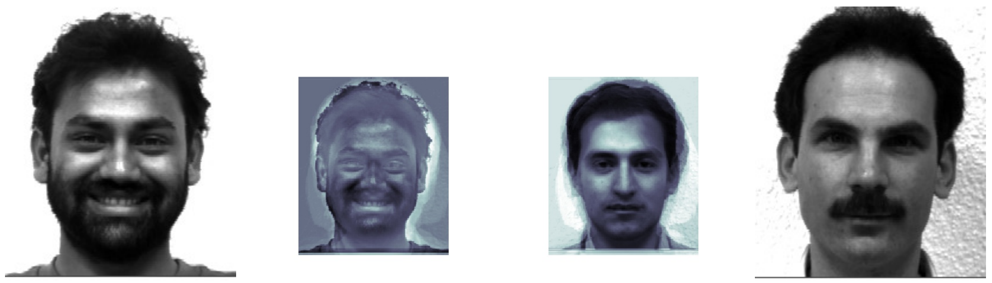
For the 3rd face, the test image is also the train image, it should be correctly.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.7: Label 7

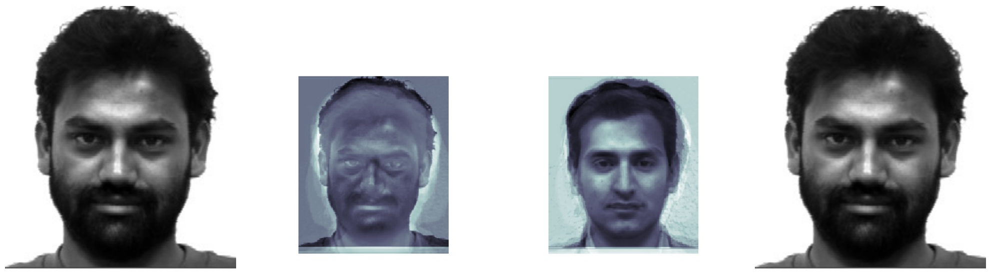
The distance from the projected test image to its nearest neighbor is 4341.019721.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.8: Label 7

The distance from the projected test image to its nearest neighbor is 3922.218310.



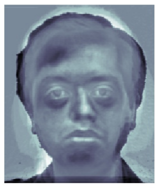
(a) test image (b) subtracting image (c) reconstructed image (d) result image

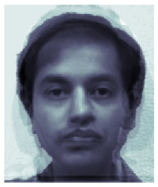
Figure 4.9: Label 7

The distance from the projected test image to its nearest neighbor is 0.0.

For the 7th face, only the training image can be detected. And I have tried every possible threshold for truncating k largest eigenvalues. And I can't predicted correctly.
And I also use the happy image or the centerlight image to train the PCA, and only the training image can be detected, and another two of the images can't be predicted correctly.

(a) test image


(b) subtracting image


(c) reconstructed image

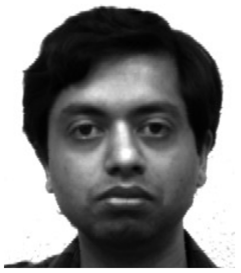
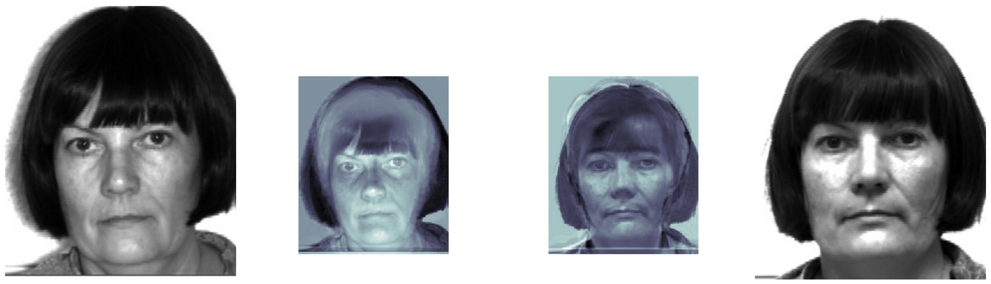

(d) result image

Figure 4.10: Label 10

The distance from the projected test image to its nearest neighbor is 0.0.

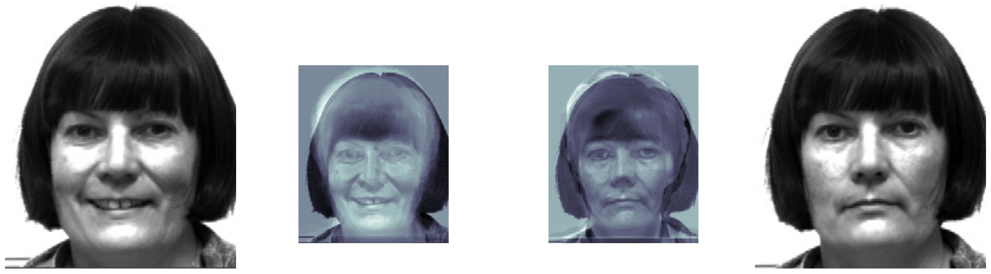
The test image is also the train image, it should be correctly.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.11: Label 11

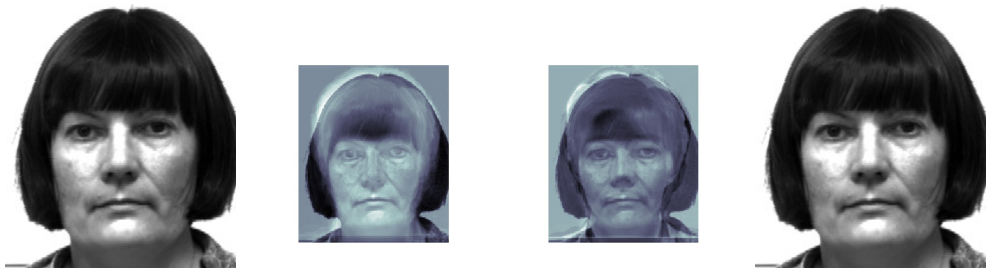
The distance from the projected test image to its nearest neighbor is 7256.988907.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.12: Label 11

The distance from the projected test image to its nearest neighbor is 1109.675944.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.13: Label 11

The distance from the projected test image to its nearest neighbor is 0.0.

For the 11th face, all three test image are detected correctly.

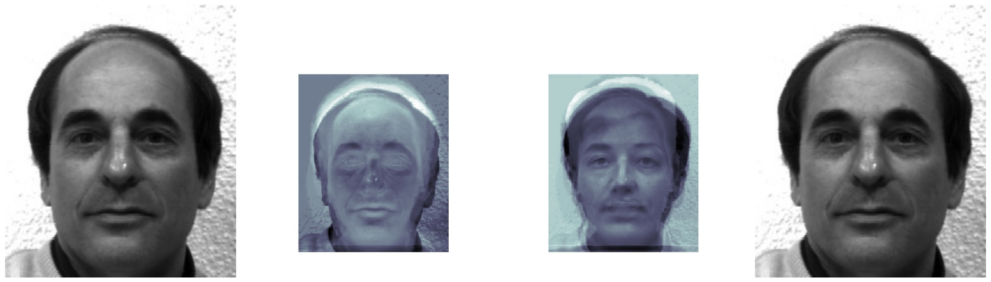

(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.14: Label 12

The distance from the projected test image to its nearest neighbor is 0.0.

The test image is also the train image, it should be correctly.

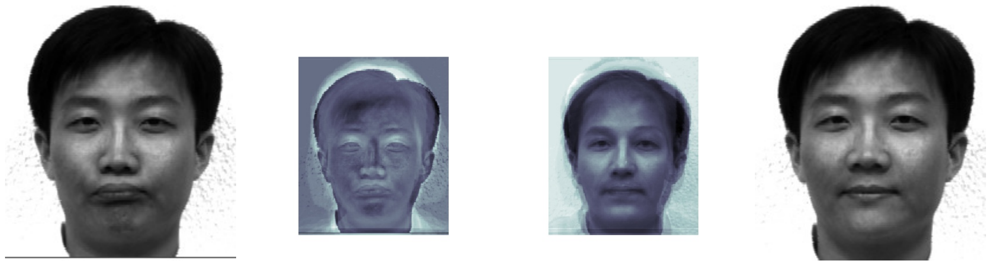

(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.15: Label 14

The distance from the projected test image to its nearest neighbor is 1788.380024.

The distance from the projected test image to its nearest neighbor is 2401.624066.

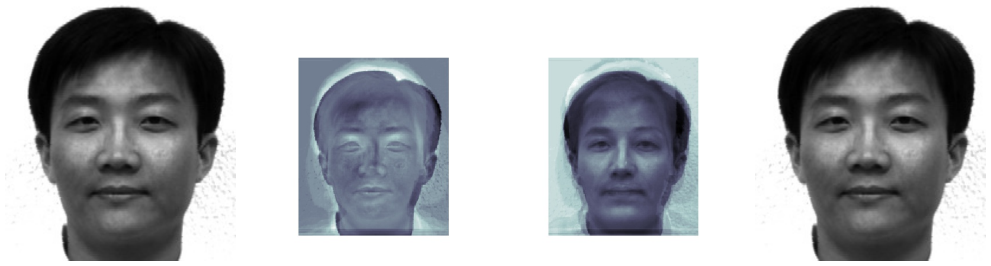
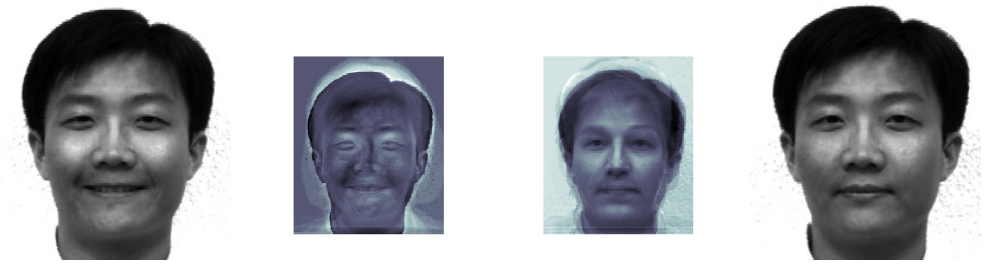

(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.17: Label 14

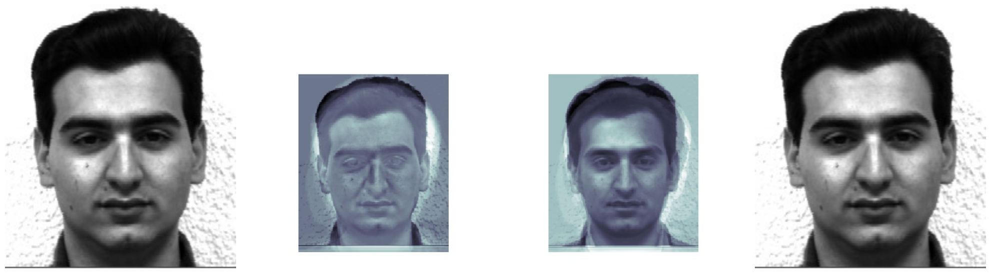


(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.16: Label 14

The distance from the projected test image to its nearest neighbor is 0.0.

For the 14th face, all three test image are detected correctly.

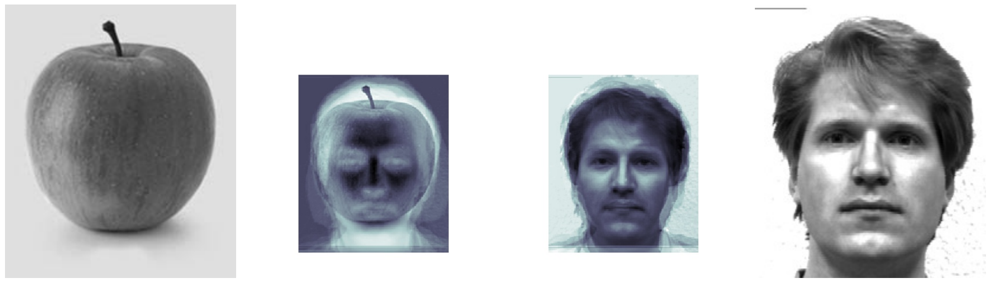


(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.18: Label 15

The distance from the projected test image to its nearest neighbor is 0.0.

For the 15th face, the test image is also the train image, it should be correctly.



(a) test image (b) subtracting image (c) reconstructed image (d) result image

Figure 4.19: Non-face

The distance from the projected test image to its nearest neighbor is 2713.258087.

The test image is an apple, and from the distance, I can't choose any threshold to figure out which is non-face and which is a face. Because there are many test image whose shortest distance to their projected neighbor is bigger than 2713.

Appendix A

Source Code

```
# -*- coding: utf-8 -*-
# @Author: Haonan Wu
# @Date:   2017-12-02 13:36:23
# @Last Modified by:   Haonan Wu
# @Last Modified time: 2017-12-02 14:06:09

import os
import cv2
import sys
import shutil
import random
import numpy as np
import matplotlib.pyplot as plt

"""
Algorithm Reference:
    http://docs.opencv.org/modules/contrib/doc/facerec/facerec_tutorial.html
"""
class Eigenfaces(object):

    # number of labels
    faces_count = 9
    faces_dir = '.'

    train_faces_count = 1
    test_faces_count = 1
```

```
face_train_ids = [1, 2, 3, 7, 10, 11, 12, 14, 15]

# training images count
l = train_faces_count * faces_count
# number of columns of the image
m = 195
# number of rows of the image
n = 231
# length of the column vector
mn = m * n

"""
Initializing the Eigenfaces model.
"""
def __init__(self, _faces_dir = '.', _threshold = 0.85):
    print '> Training started'

    self.faces_dir = _faces_dir
    self.threshold = _threshold
    self.training_ids = []

    L = np.empty(shape=(self.l, self.mn), dtype='float64')
    cur_img = 0

    for face_id in self.face_train_ids:
        training_ids = [1]
        self.training_ids.append(training_ids)
        for training_id in training_ids:
            path_to_img = os.path.join(self.faces_dir,
                    's' + str(face_id), str(training_id) + '.jpg')
            img = cv2.imread(path_to_img, 0)
            img_row = np.array(img, dtype='float64').flatten()
            L[cur_img] = img_row
            cur_img += 1

    """
```

```
get the mean of all images / over the rows of L
subtract from all training images
"""
self.mean_img_row = np.mean(L, axis=0)


# output the mean image
fig, ax1 = plt.subplots(frameon=False)
ax1.set_axis_off()
tmp = np.reshape(self.mean_img_row, (self.n, self.m))
ax1.imshow(tmp, cmap="bone")
fig.savefig("image/mean.jpg", bbox_inches='tight', pad_inches=0)

L -= self.mean_img_row
"""
print(L.shape)
for j in range(len(self.face_train_ids)):
    fig, ax1 = plt.subplots(ncols=1, nrows=1, figsize=(4, 4))
    ax1.set_axis_off()
    tmp = np.reshape(L[j], (self.n, self.m))
    ax1.imshow(tmp, cmap="bone")
    fig.savefig("image/train_"+str(self.face_train_ids[j])+".jpg")
"""
C = np.matrix(L) * np.matrix(L.transpose())
C /= L.shape[0]


"""
Eigenvectors/values of the covariance matrix.
And set them into decreasing order of values.
"""

self.evalues, self.evectors = np.linalg.eig(C)
sort_indices = self.evalues.argsort()[::-1]
self.evalues = self.evalues[sort_indices]
self.evectors = self.evectors[sort_indices]
```

```
"""
include only the first k evectors/values so
that they include approx.
"""
evalues_sum = sum(self.evalues[:])
evalues_count = 0
evalues_radio = 0.0
for evalue in self.evalues:
    evalues_count += 1
    evalues_radio += evalue / evalues_sum

    if evalues_radio >= self.threshold:
        break

# truncate the number of eigenvectors/values to consider
self.evalues = self.evalues[0:evalues_count]
self.evectors = self.evectors[0:evalues_count]

"""
change eigenvectors from rows to columns
left multiply to get the correct evectors
find the norm of each eigenvector
normalize all eigenvectors
"""
self.evectors = self.evectors * L
norms = np.linalg.norm(self.evectors, axis=1)
self.evectors = self.evectors.transpose() / norms
"""
# output the eigen face
print(self.evectors.shape)
for j in range(self.evectors.shape[1]):
    fig, ax1 = plt.subplots(ncols=1, nrows=1, figsize=(4, 4))
    ax1.set_axis_off()
    tmp = np.reshape(self.evectors[:, j], (self.n, self.m))
    ax1.imshow(tmp, cmap="bone")
```

```
            fig.savefig("image/"+str(j)+".jpg")
        """
        self.W = L * self.evectors

        print '> Training ended'

    """
    Classify an image to one of the eigenfaces.
    """
    def classify(self, path_to_img):
        img = cv2.imread(path_to_img, 0)
        img_row = np.array(img, dtype='float64').flatten()
        img_row -= self.mean_img_row



        S =  img_row * self.evectors

        """
        projecting the normalized probe onto the
        Eigenspace, to find out the weights
        """
        diff = self.W - S

                                              # finding the min ||W_j - S||
        norms = np.linalg.norm(diff, axis=0)

        closest_face_id = np.argmin(norms)

                                              # the id [0..240) of the
                                              minerror face to the sample
        return self.face_train_ids[(closest_face_id / self.train_faces_count
                                              )]

    def validate(self):
        print '> Evaluating faces started'
        results_file = os.path.join('results', 'results.txt')
```

```python
        f = open(results_file, 'w')

        test_count = self.test_faces_count * self.faces_count
        test_correct = 0
        for face_id in self.face_train_ids:
            for test_id in xrange(1, self.test_faces_count+1):
                path_to_img = os.path.join(self.faces_dir,
                        's' + str(face_id), str(test_id) + '.jpg')

                result_id = self.classify(path_to_img)
                result = (result_id == face_id)

                if result == True:
                    test_correct += 1
                    f.write('image: %s\nresult: correct, got %2d\n\n' % (
                                                          path_to_img,
                                                          result_id))
                else:
                    f.write('image: %s\nresult: wrong, got %2d\n\n' %
                            (path_to_img, result_id))

        print '> Evaluating faces ended'
        self.accuracy = float(100. * test_correct / test_count)
        print 'Correct: ' + str(self.accuracy) + '%'
        f.write('Correct: %.2f\n' % (self.accuracy))
        f.close()

                                              # closing the file


    def evaluate(self, celebrity_dir='.'):
        print '> Evaluating test data set matches started'
        # go through all the celebrity images in the folder
        for img_name in os.listdir(celebrity_dir):
            path_to_img = os.path.join(celebrity_dir, img_name)
            """
```

```
# read as a grayscale image
# flatten the image
# subract the mean column
# from row vector to col vector
"""
img = cv2.imread(path_to_img, 0)
img_row = np.array(img, dtype='float64').flatten()
img_row -= self.mean_img_row

"""
# projecting the normalized probe onto the
# Eigenspace, to find out the weights
"""
S = img_row * self.evectors

# finding the min ||W_j - S||
diff = self.W - S
norms = np.linalg.norm(diff, axis=1)
top_ids = np.argpartition(norms, 1)[:1]

"""
the image file name without extension
path to the respective results folder
make a results folder for the respective celebrity
the file with the similarity value and id's
"""
name_noext = os.path.splitext(img_name)[0]
result_dir = 'results' #os.path.join('results', name_noext)
# os.makedirs(result_dir)
result_file = os.path.join(result_dir, 'results_' + name_noext +
                                       '.txt')

"""
# output the reconstruction image
# and the origin image subtracts the mean image
print(S.shape)
```

```
        print(self.W.shape)


        fig, ax1 = plt.subplots(ncols=1, nrows=1, figsize=(4, 4))
        ax1.set_axis_off()
        tmp = S * self.evectors.transpose() + self.mean_img_row
        tmp = np.reshape(tmp, (self.n, self.m))
        #tmp = np.reshape(img_row, (self.n, self.m))
        ax1.imshow(tmp, cmap="bone")
        fig.savefig("image/"+name_noext+".jpg")
```