

Face Recognition with Eigenfaces

by

Haonan Wu

N18859539

New York University

Dec, 2017

E.D Wong

Copyright 2019 Haonan Wu

Table of Contents

1	Introduction	1
2	Environment Description	2
3	Algorithmic Description	3
4	Result	5
	Appendix A. Source Code	16

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 folder '*train_data*', '*test_data*' and python file '*eigenfaces.py*' in the same folder and simply run

```
python eigenfaces.py
```

Chapter 3

Algorithmic Description

step 1 Let $X = \{x_1, x_2, \dots, x_n\}$ 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 8×8 .

step 4 Compute the eigenvalues λ_i and eigenvectors v_i of

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

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, I use $k = 5$.

step 6 The 5 principal components of the test set x are then given by:

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

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

step 1 For each test image x . Subtract mean face from it.

step 2 Compute the projection of it.

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

step 3 Reconstruct it from eigenfaces.

$$x' = W\Omega$$

step 4 Compute the Manhattan distance between x and x' . We use the threshold 2271222 to identify non-face and face.

step 5 Finding the nearest neighbor between the projected training images and the projected test image, using Manhattan distance. We use the threshold 13988 to identify unknown face and other faces.

Chapter 4

Result



The mean face.

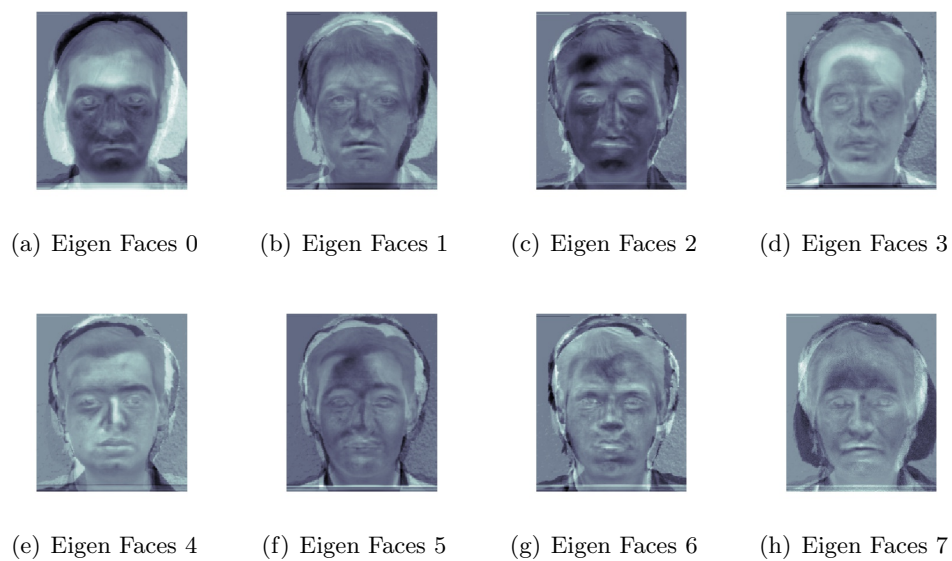


Figure 4.1: Eigen Faces

The eigen space.

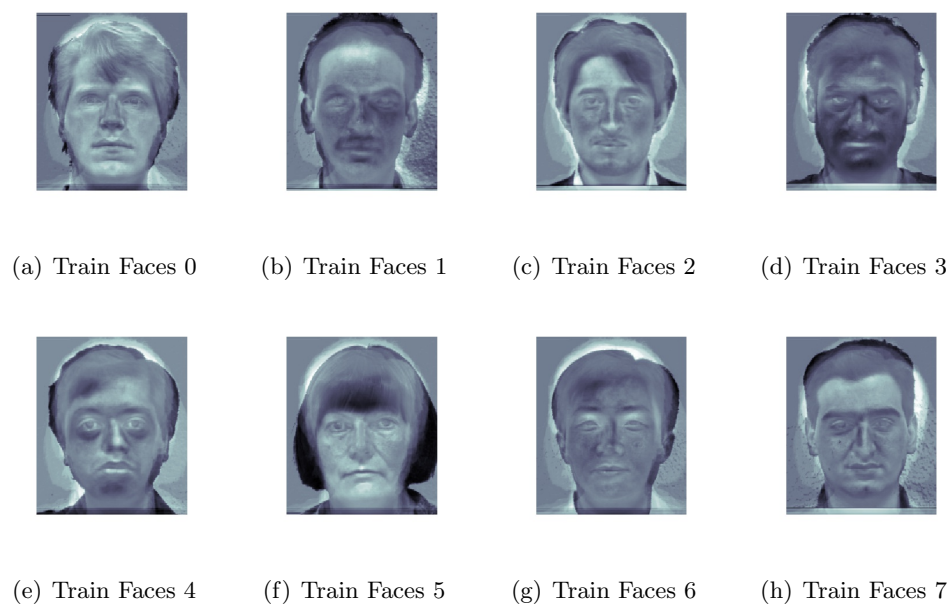


Figure 4.2: Train Face

Train image subtract mean face.

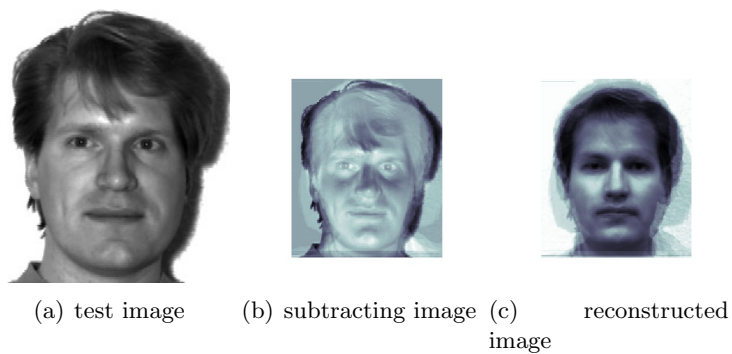


Figure 4.3: Label 1

The distance from the projected test image to its nearest neighbor is 14002.141338. And this distance is larger than threshold2, so it is unknown face.

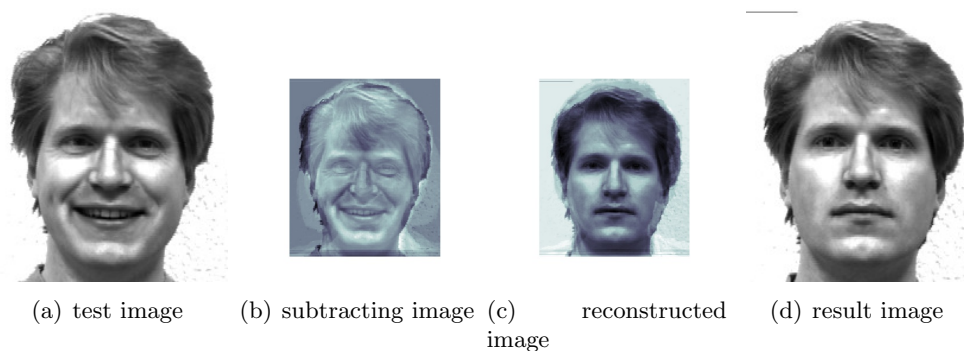


Figure 4.4: Label 1

The distance from the projected test image to its nearest neighbor is 7044.465923.

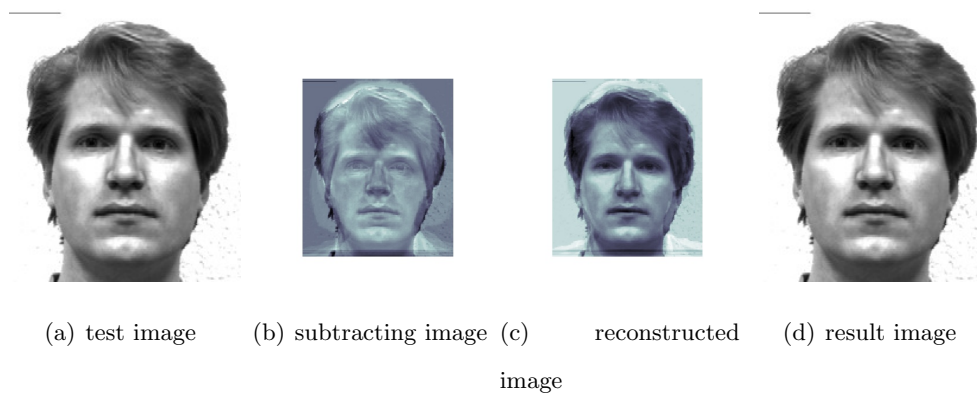


Figure 4.5: Label 1

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

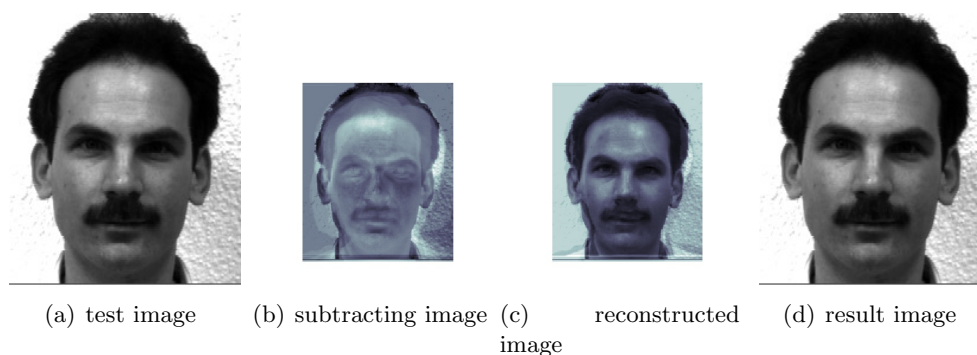


Figure 4.6: Label 2

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.

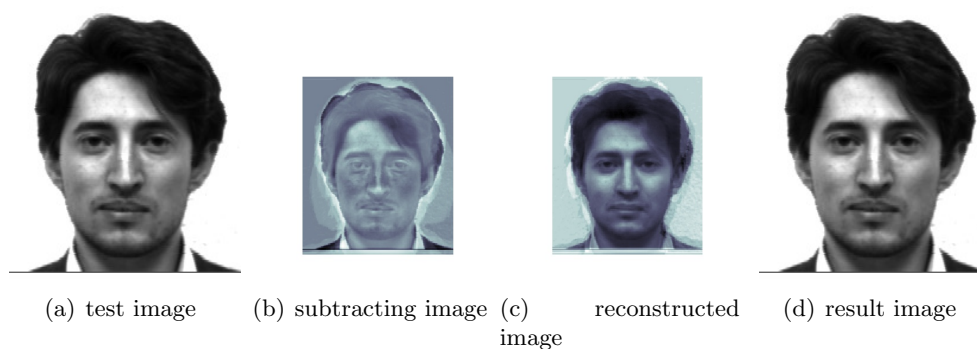


Figure 4.7: Label 3

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.

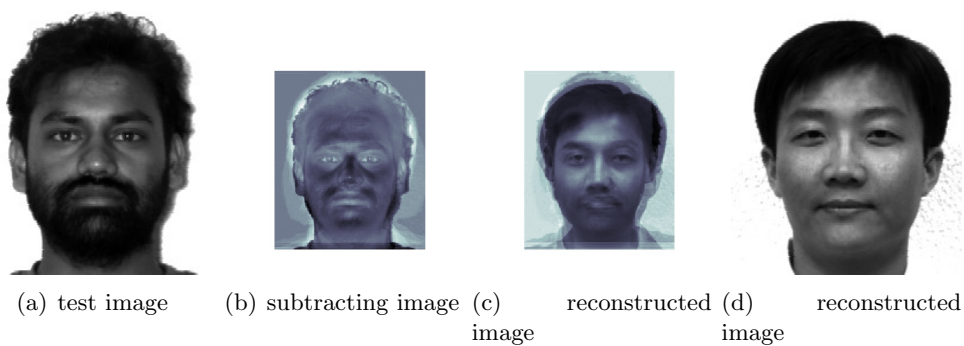


Figure 4.8: Label 7

The distance from the projected test image to its nearest neighbor is 12035.314184.

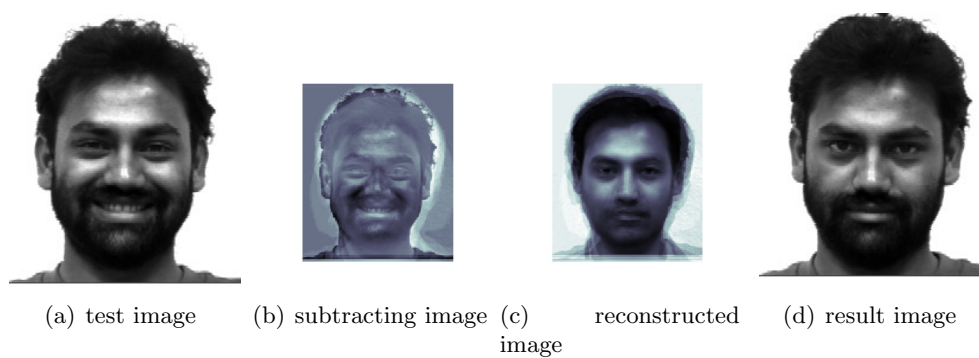


Figure 4.9: Label 7

The distance from the projected test image to its nearest neighbor is 13140.704185.

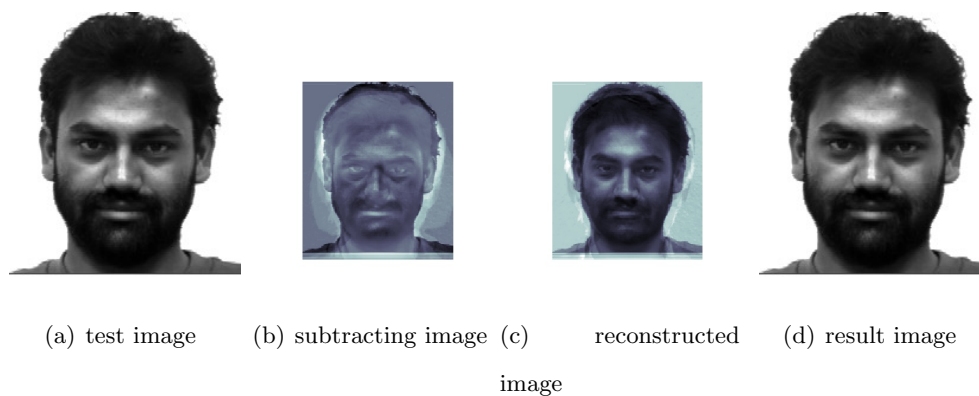


Figure 4.10: Label 7

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

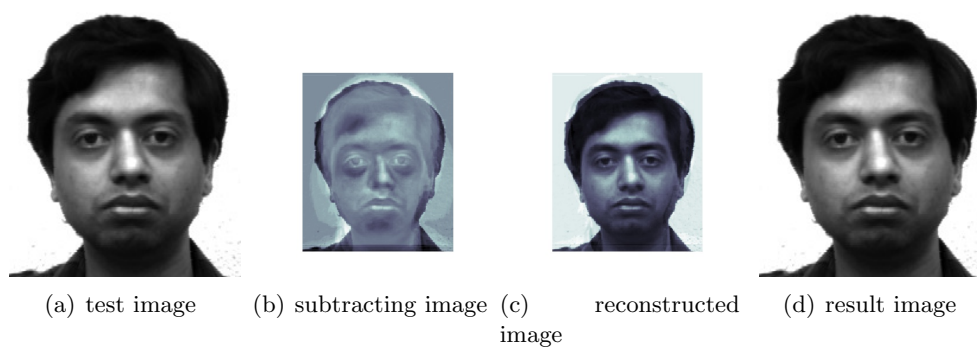


Figure 4.11: Label 10

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.

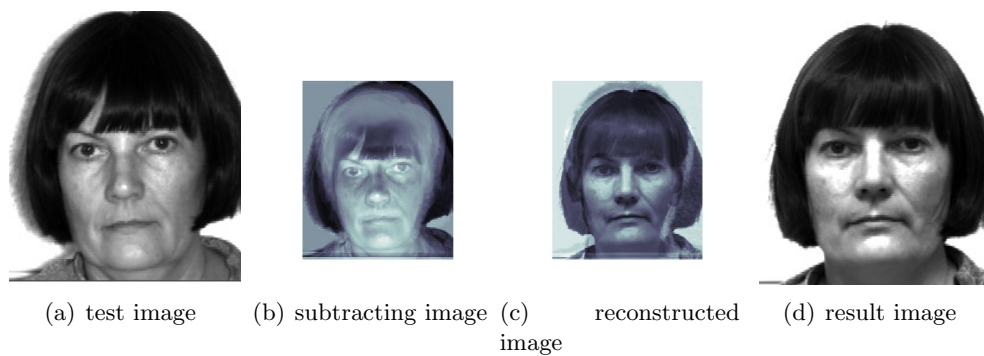


Figure 4.12: Label 11

The distance from the projected test image to its nearest neighbor is 12094.222523.



Figure 4.13: Label 11

The distance from the projected test image to its nearest neighbor is 2692.597414.

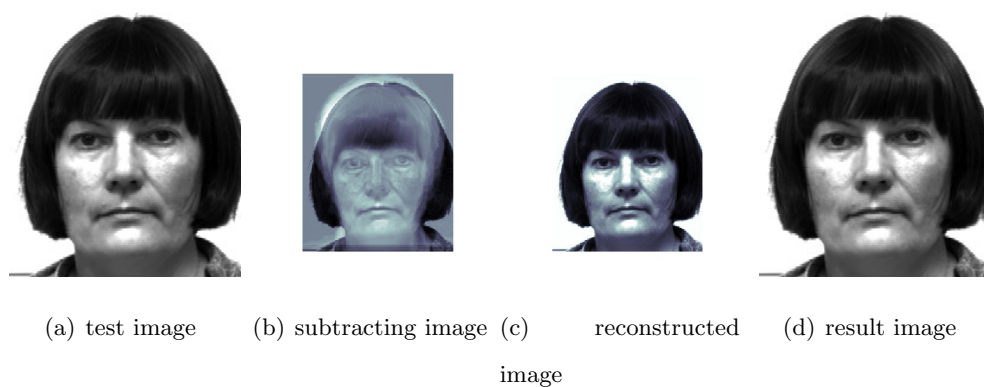


Figure 4.14: Label 11

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.

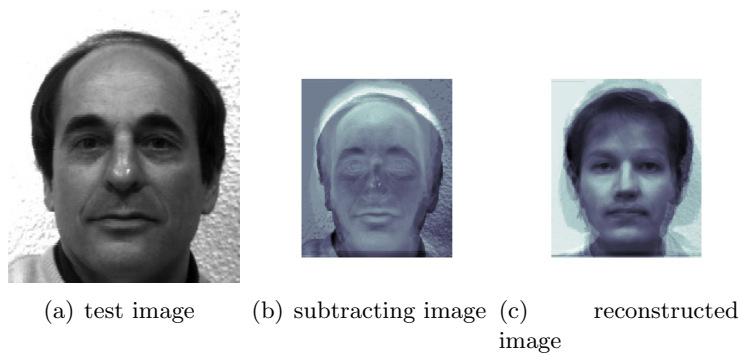


Figure 4.15: Label 12

The distance from the projected test image to its nearest neighbor is 13988.065146.
And this distance is larger than threshold2, so it is unknown face.

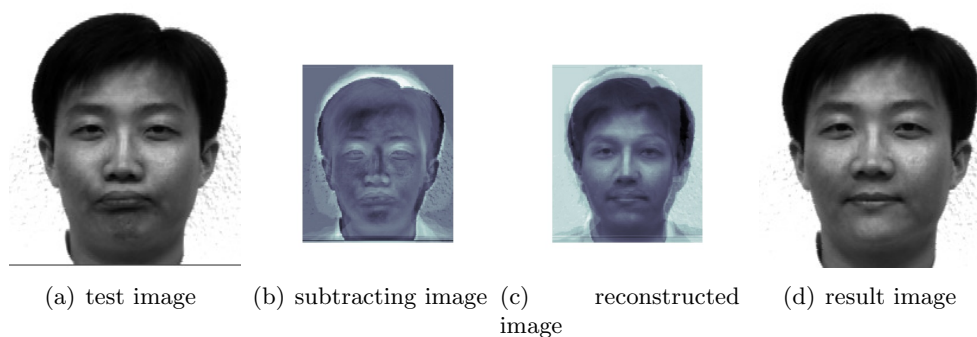


Figure 4.16: Label 14

The distance from the projected test image to its nearest neighbor is 2296.664520.
The distance from the projected test image to its nearest neighbor is 3218.492827.

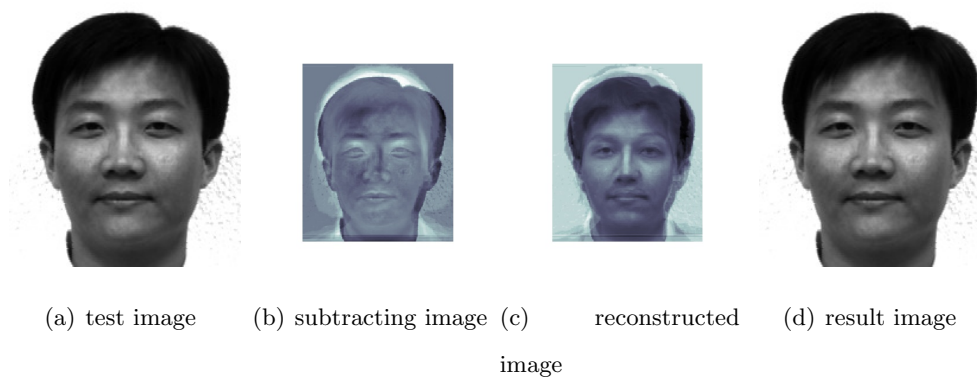


Figure 4.18: Label 14

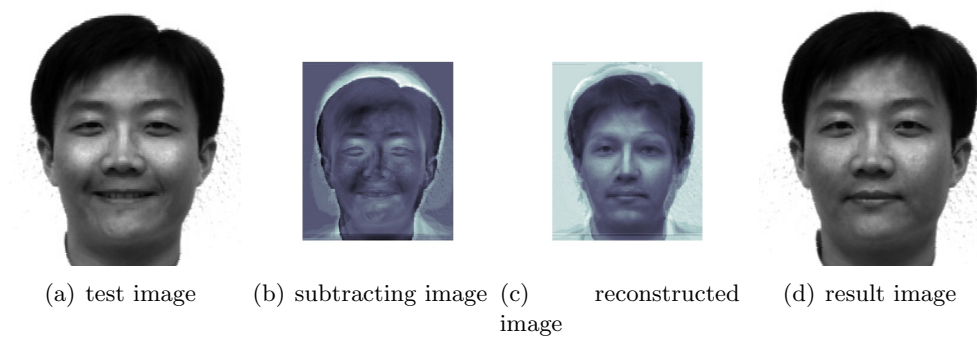


Figure 4.17: Label 14

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.

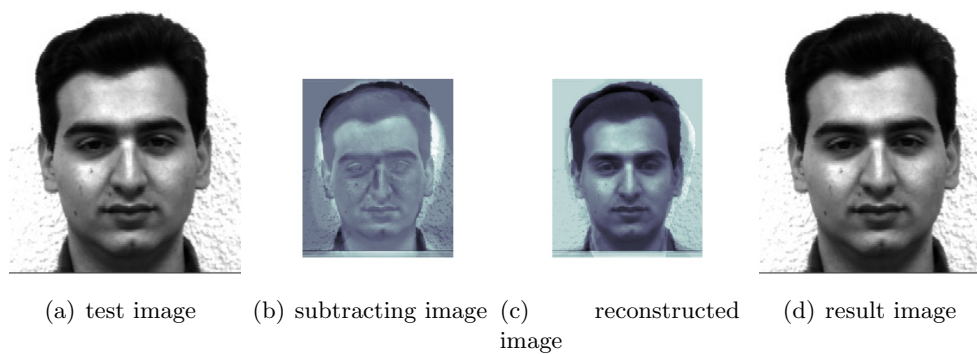


Figure 4.19: Label 15

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.

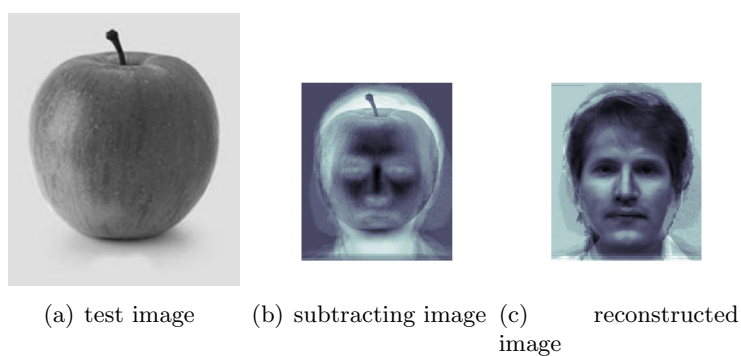


Figure 4.20: Non-face

This image is detected as non-face.

Appendix A

Source Code

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

class Eigenfaces(object):

    # number of labels
    faces_count = 8

    # directory path to the AT&T faces
    faces_dir = '.'

    train_faces_count = 1
    test_faces_count = 1
    face_train_ids = [1, 2, 3, 7, 10, 11, 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 '> Initializing started'

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

    L = np.empty(shape=(self.mn, self.l), 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')

            #print '> reading file: ' + path_to_img
            img = cv2.imread(path_to_img, 0)
            img_col = np.array(img, dtype='float64').flatten()
            L[:, cur_img] = img_col[:]
            cur_img += 1

    # get the mean of all images / over the rows of L
    self.mean_img_col = np.mean(L, axis=1)

    """
    # output the mean image
    fig, ax1 = plt.subplots(ncols=1, nrows=1, figsize=(4, 4))
    ax1.set_axis_off()

    tmp = np.reshape(self.mean_img_col, (self.n, self.m))
    ax1.imshow(tmp, cmap="bone")

```

```

fig.savefig("image/mean.jpg")
"""

# subtract from all training images
for j in xrange(0, self.l):
    L[:, j] -= self.mean_img_col[:, j]

"""

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")
"""

"""
Because L's columns represent the image vector,
we set  $C = L^T L$ 
"""

C = np.matrix(L.transpose()) * np.matrix(L)
C /= L.shape[1]

"""
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_count = 0
    evalues_sum = sum(self.evalues[:])
    evalues_radio = 0.0
    for evalue in self.evalues:
        evalues_count += 1
        evalues_radio += evalue / evalues_sum

        if evalues_radio >= self.threshold:
            break
    """
    evalues_count = 5
    # 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 = L * self.evectors
    norms = np.linalg.norm(self.evectors, axis=0)
    self.evectors = self.evectors / 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 = self.evectors.transpose() * L

```

```

        print '> Initializing ended'

    """
    Classify an image to one of the eigenfaces.
    """
    def classify(self, path_to_img):
        img = cv2.imread(path_to_img, 0)
        img_col = np.array(img, dtype='float64').flatten()
        img_col -= self.mean_img_col
        img_col = np.reshape(img_col, (self.mn, 1))

                                                #
                                                from row vector to col vector

        S = self.evectors.transpose() * img_col
        """
        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
                                     )]

    """
    Evaluate the model using the 4 test faces left
    from every different face in the AT&T set.
    """
    def validate(self):
        print '> Evaluating faces started'

```

```

results_file = os.path.join('results', 'results.txt')

# filename for writing the
# evaluating results in

f = open(results_file, 'w')

# the actual file

test_count = self.test_faces_count * self.faces_count

# number of
# all AT&T test images/faces

test_correct = 0
for face_id in self.face_train_ids:
    for test_id in xrange(1, self.test_faces_count+1):
        # if (test_id in self.training_ids[face_id-1]) == False:
        # we skip the
        # image if it is part
        # of the training set

        path_to_img = os.path.join(self.faces_dir,
                                    's' + str(face_id), str(test_id) + '.jpg')

        # relative
        # path

        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

"""
Evaluate the model for the small celebrity data set.
Returning the top 5 matches within the AT&T set.
Images should have the same size (92,112) and are
located in the celebrity_dir folder.
"""

def evaluate(self, celebrity_dir='.'):
    print '> Evaluating test data set matches started'
    # go through all the celebrity images in the folder
    flag = -1.0
    for img_name in os.listdir(celebrity_dir):
        path_to_img = os.path.join(celebrity_dir, img_name)
        name_noext = os.path.splitext(img_name)[0]
        """
        # read as a grayscale image
        # flatten the image
        # subtract the mean column
        # from row vector to col vector
        """
        img = cv2.imread(path_to_img, 0)
        img_col = np.array(img, dtype='float64').flatten()
        img_col -= self.mean_img_col
        img_col = np.reshape(img_col, (self.mn, 1))
        """
        # projecting the normalized probe onto the
        # Eigenspace, to find out the weights
        """
        S = self.evectors.transpose() * img_col

        reconstruction = self.evectors * S

```

```

"""
# output the reconstruction image
# and the origin image subtracts the mean image
print(reconsturction.shape)
print(self.mean_img_col.shape)

fig, ax1 = plt.subplots(ncols=1, nrows=1, figsize=(4, 4))
ax1.set_axis_off()
tmp = reconsturction+np.reshape(self.mean_img_col, (self.mn, 1))
tmp = np.reshape(tmp, (self.n, self.m))
#tmp = np.reshape(img_col, (self.n, self.m))
ax1.imshow(tmp, cmap="bone")
fig.savefig("image/"+name_noext+".jpg")
"""

diff = reconsturction - img_col
dis0 = np.linalg.norm(diff, ord=1)

if flag < 0:
    flag = dis0

if dis0 >= flag:
    #print(dis0)
    result_dir = 'results' #os.path.join('results', name_noext)
    # os.makedirs(result_dir)
    result_file = os.path.join(result_dir, 'results_no_face_' +
                                name_noext + '.txt')

    f = open(result_file, 'w')
    f.write('Not a face')
    continue

# finding the min ||W_j - S||
diff = self.W - S
norms = np.linalg.norm(diff, axis=0, ord=1)
mean_dis = np.mean(norms)
top_ids = [np.argmin(norms)]#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')

f = open(result_file, 'w')

                                # open the results file
                                for writing

for top_id in top_ids:
    if norms[top_id] < 13988:
        face_id = (top_id / self.train_faces_count) + 1
                                #
                                getting the
                                face_id of one of
                                the closest
                                matches
        subface_id = self.training_ids[face_id-1][top_id % self.
                                train_faces_count]
                                #
                                getting the exact
                                subimage from the
                                face
        face_idx = self.face_train_ids[face_id-1]
        path_to_img = os.path.join(self.faces_dir,
                                    's' + str(face_idx), str(subface_id) + '.jpg')

                                # relative
                                path to

```

```

the top5
face

shutil.copyfile(path_to_img,

# copy the top
face from source
os.path.join(result_dir, 'results_' + name_noext
+ '.jpg')
)
# to
destination

f.write('id: %3d, score: %.6f\n' % (top_id, norms[top_id
])) # write
the id and its
score to the
results file

else:
    f.write("Unkown Face")
    #print(name_noext, norms[top_id], mean_dis)
f.close()

print '> Evaluating test data set matches ended'

if __name__ == "__main__":
    TRAIN_DATA_DIR = ".\\train_data"
    TEST_DATA_DIR = ".\\test_data"

    if not os.path.exists('results'):
        os.makedirs('results')
    else:
        shutil.rmtree('results')
        os.makedirs('results')

```

```
efaces = Eigenfaces(str(TRAIN_DATA_DIR), _threshold=0.7)
#efaces.validate()

# if we have third argument (celebrity folder)
efaces.evaluate(str(TEST_DATA_DIR))
```