전남대학교
CHONNAM NATIONAL UNIVERSITY

SCHOOL OF ELECTRONICS & COMPUTER ENGINEERING

COMPUTER VISION

PROGRAMMING #1

EIGEN FACE RECOGNITION

Student Name	:	**Do Nhu Tai** 다이 도느
Student ID	:	176680
Email	:	donhutai@gmail.com
Submission Date	:	2018. 04. 21
Professor	:	**Lee, Chilwoo**

Chonnam National Univeristry

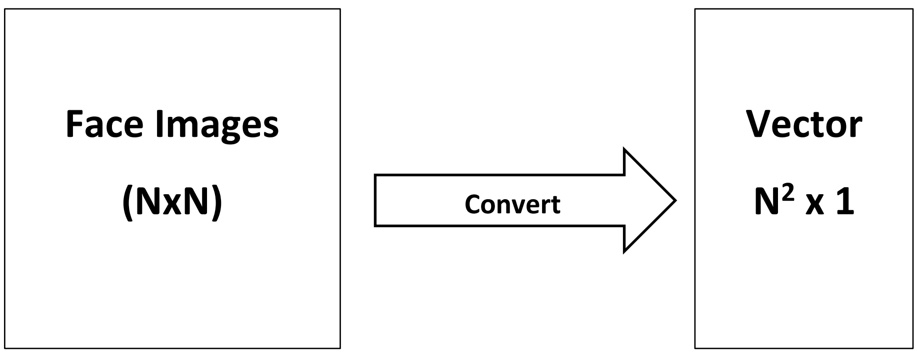
COMPUTER VISION – PROGRAMMING 1

Tai Do Nhu (다이 도느), 176680, donhutai@gmail.com

Table of Contents

1. Eigen Face Recognition

1.1. Main Idea



The problem that arises when identifying is that the dimension is too large (N^2). How do we find space with less dimension?
The goal of the method is to "reduce the number of dimensions" of a vector set so that it remains "the most important information". Feature extraction (keep the attribute "new") rather than feature selection (retain the original property k).

$$X = a_1v_1 + a_2v_2 + \dots + a_Mv_M \rightarrow Y = b_{1u1} + b_{2u2} + \dots + b_ku_k$$

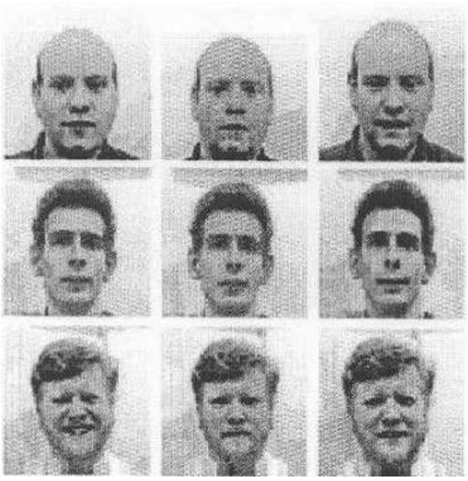
The PCA method will try to find the linear transformation T satisfying: y = T.x such that the mean squared error (MSE) is the smallest.

The PCA method involves the eigenvalues and eigenvectors of the covariance matrix C of the sample set X. We then only retain the K vector separately for the largest K prime to do the basis for this new space.

2. Step by Step

2.1. Tính toán các Eigenfaces

Step 1: Use face images I_1, I_2, ... I_M (face set training) with face right and all photos must be the same size.



Step 2: Reproduces all I_i images into vector Γ_i

Example: For simplicity we assume that there are only 4 images in the training set (3x3 size). We can calculate:

$$\Gamma = \begin{bmatrix} 225 \\ 229 \\ 48 \\ 251 \\ 33 \\ 238 \\ 0 \\ 255 \\ 217 \end{bmatrix} \quad \Gamma = \begin{bmatrix} 10 \\ 219 \\ 24 \\ 255 \\ 18 \\ 247 \\ 17 \\ 255 \\ 2 \end{bmatrix} \quad \Gamma = \begin{bmatrix} 196 \\ 35 \\ 234 \\ 232 \\ 59 \\ 244 \\ 243 \\ 57 \\ 226 \end{bmatrix} \quad \Gamma = \begin{bmatrix} 255 \\ 223 \\ 224 \\ 255 \\ 0 \\ 255 \\ 249 \\ 255 \\ 235 \end{bmatrix}$$

Step 3: Calculate the average face vector by the formula:

$$\boxed{\Psi = \frac{1}{M}\sum_{i=1}^{M}\Gamma_i}$$

Specifically we have:

$$\Psi = \begin{bmatrix} 171.50 \\ 176.50 \\ 135.5 \\ 248.25 \\ 27.50 \\ 246.00 \\ 127.25 \\ 205.50 \\ 170.00 \end{bmatrix}$$

Step 4: Minus nfor the average face vector. Specifically we have:

$$\boxed{\Phi_i = \Gamma_i - \Psi}$$

$$\bar{x}^1 = \begin{bmatrix} 53.50 \\ 52.50 \\ -84.50 \\ 2.75 \\ 5.50 \\ -8.00 \\ 127.25 \\ 49.50 \\ 47.00 \end{bmatrix} \quad \bar{x}^2 = \begin{bmatrix} -161.50 \\ 42.50 \\ -108.50 \\ 6.75 \\ -9.50 \\ 1.00 \\ -110.25 \\ 49.50 \\ -168.00 \end{bmatrix} \quad \bar{x}^3 = \begin{bmatrix} 24.50 \\ -141.50 \\ 101.50 \\ -16.25 \\ 31.50 \\ -2.00 \\ 115.75 \\ -148.50 \\ 56.00 \end{bmatrix} \quad \bar{x}^4 = \begin{bmatrix} 83.50 \\ 46.50 \\ 91.50 \\ 6.75 \\ -27.50 \\ 9.00 \\ 121.75 \\ 9.49.50 \\ 65.00 \end{bmatrix}$$

Step 5: Calculate covariance C with N^2 x N^2

$$\boxed{C = \frac{1}{M} \sum_{n=1}^{M} \Phi_n \Phi_n^T = AA^T}$$

Where A with N^2 x M:

$$\boxed{A = [\Phi_1 \ \Phi_2 \cdots \Phi_M]}$$

$$A = \begin{bmatrix} 53.50 & -161.50 & 24.50 & 83.50 \\ 52.50 & 42.50 & -141.50 & 46.50 \\ -84.50 & -108.50 & 101.50 & 91.50 \\ 2.75 & 6.75 & -16.25 & 6.75 \\ 5.50 & -9.50 & 31.50 & -27.50 \\ -8.00 & 1.00 & -2.00 & 9.00 \\ -127.25 & -110.25 & 115.75 & 121.75 \\ 49.50 & 49.50 & -148.50 & 49.50 \\ 47.00 & -168.00 & 56.00 & 65.00 \end{bmatrix}$$

With C such as:

$$C = \begin{bmatrix} 36517 & -3639 & 23129 & -778 & 304 & 113 & 24000 & -4851 & 36446 \\ -3639 & 26747 & -19155 & 3045 & -5851 & 324 & -22083 & 28017 & -9574 \\ 23129 & -19155 & 37587 & -1997 & 1247 & 1188 & 45603 & -20097 & 25888 \\ -778 & 3045 & -1996 & 363 & -746.5 & 78 & -2153 & 3217 & -1476 \\ 304 & -5851 & 1247 & -747 & 1869 & -364 & 645 & -6237 & 1831 \\ 113 & 324 & 1188 & 78 & -364 & 150 & 1772 & 396 & -71 \\ 24000 & -22083 & 45603 & -2153 & 645.5 & 1772 & 56569 & -22919 & 26937 \\ -4851 & 28017 & -20097 & 3218 & -6237 & 396 & -22919 & 29403 & -11088 \\ 36446 & -9574 & 25888 & -1476 & 1831 & -71 & 26937 & -11088 & 37794 \end{bmatrix}$$

Step 6: Calculates the eigenvector u_i ("private vector") of the $A.A^T$ square matrix (C is N^2xN^2). This matrix is too large and not feasible.

Step 6.1: Consider the matrix $A^T.A$ (note that this matrix is only M x M size)

$$A^T.A = \begin{bmatrix} 33712 & 11301 & -33998 & -115015 \\ 11301 & 82627 & -50914 & -43014 \\ -33998 & -50914 & 70771 & 14141 \\ -11015 & -43014 & 14141 & 39888 \end{bmatrix}$$

Step 6.2: Calculates the eigenvectors of this $A^T.A$ square matrix

$$v_1 = \begin{bmatrix} -0.263 \\ -0.679 \\ 0.586 \\ 0.355 \end{bmatrix} \quad v_2 = \begin{bmatrix} 0.521 \\ -0.437 \\ -0.559 \\ 0.475 \end{bmatrix} \quad v_3 = \begin{bmatrix} -0.640 \\ 0.314 \\ -0.306 \\ 0.631 \end{bmatrix}$$

$$\lambda_1 = 153520 \qquad \lambda_2 = 50696 \qquad \lambda_3 = 22781$$

After calculating the microscopic vectors (size Mx1), it is easy to deduce the individual vectors u_i (size N^2x1) desired by the formula:

$$\boxed{u_i = Av_i} \quad (*)$$

$$u_1 = \begin{bmatrix} 139.5734 \\ -109.108 \\ 187.906 \\ -12.435 \\ 13.699 \\ 3.452 \\ 219.448 \\ -116.108 \\ 157.593 \end{bmatrix} \quad u_2 = \begin{bmatrix} 124.311 \\ 109.995 \\ -9.981 \\ 10.777 \\ -23.655 \\ 0.781 \\ -25.098 \\ 11.715 \\ 97.366 \end{bmatrix} \quad u_3 = \begin{bmatrix} -39.787 \\ 52.380 \\ 46.675 \\ 9.591 \\ -33.493 \\ 11.725 \\ 88.213 \\ 60.533 \\ -58.978 \end{bmatrix}$$

Note the normalization of vectors u_i: $\|u_i\| = 1$, $u_i = \dfrac{u_i}{\|u_i\|}$. After normalization, we obtain the last vectors as follows:

$$u_1 \begin{bmatrix} 0.356 \\ -0.279 \\ 0.480 \\ -0.032 \\ 0.035 \\ 0.009 \\ 0.560 \\ -0.296 \\ 0.402 \end{bmatrix} \quad u_2 \begin{bmatrix} -0.552 \\ -0.489 \\ 0.044 \\ -0.048 \\ 0.105 \\ -0.004 \\ 0.112 \\ 0.492 \\ -0.432 \end{bmatrix} \quad u_3 \begin{bmatrix} -0.264 \\ 0.347 \\ 0.309 \\ 0.064 \\ -0.222 \\ 0.078 \\ 0.585 \\ 0.401 \\ -0.391 \end{bmatrix}$$

Step 6.3: Calculate the best vector M i of A.AT by the formula (*).

Step 7: Only retain the vector K of the above-mentioned M vector (corresponding to the largest K value), of course K << N^2.

Select K by the following formula:

$$\frac{\sum_{i=1}^{K} \lambda_i}{\sum_{i=1}^{N} \lambda_i} > Threshold \quad \text{(e.g., 0.9 or 0.95)}$$

2.2. Represent the available faces (training set) into this vector space

Each face Φ_i in the training set can be re-expressed as a linear combination of the largest individual K vector:

$$\boxed{\sum_{j=1}^{K} w_j u_j}$$

Where

$$\boxed{(w_j = u_j^T \Phi_i)}$$

2.3. Face recognition with EIGENFACES

Standardized

$$\Gamma : \Phi = \Gamma - \Psi$$

Performed Φ as follows:

$$\Omega = \begin{bmatrix} u_1^T.\Phi \\ u_2^T.\Phi \\ u_3^T.\Phi \\ \text{K K} \\ \text{K K} \\ u_K^T.\Phi \end{bmatrix}$$

Find min:

$$\boxed{e_r = \min_l \|\Omega - \Omega^l\|}$$

3. Python Code

3.1. Datasets

yalefaces.py

```
import os
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import cv2
from glob import glob

class YaleFaceDb(object):
    def __init__(self, image_width = 100, image_height = 100, image_dir =
'datasets/yalefaces/centered'):
        self.image_dir = image_dir
        self.image_width = image_width
        self.image_height = image_height
        self.type = type

        self.image_list_person  = { }
        self.image_list_subject = { }
        self.image_list_person_subject  = { }

        self.image_label = None
        self.image_list = None

        self.load()
    # __init__

    def load(self):
        self.image_list_person.clear()
        self.image_list_subject.clear()
        self.image_list_person_subject.clear()

        image_real_dir = os.path.realpath(self.image_dir)
        image_names = glob(os.path.join(self.image_dir,'*.*'))

        image_list = []
        image_label = []
        for image_path in image_names:
            (_, image_name) = os.path.split(image_path)
            names  = image_name.split('.') # perrson.subject(.pgm)
            person  = names[0]
            subject = names[1]

            image = Image.open(image_path)
            image = image.resize((self.image_width,self.image_height),Image.ANTIALIAS)
            image = np.expand_dims(np.asarray(image), axis=3)

            image_label.append([person, subject, image_name])
            image_list.append(image)

            if self.image_list_subject.get(subject)==None:
                self.image_list_subject[subject] = []
            self.image_list_subject[subject].append(image)

            if self.image_list_person.get(person)==None:
                self.image_list_person[person] = []
            self.image_list_person[person].append(image)

            self.image_list_person_subject[image_name] = image
        # for
        self.image_list = np.array(image_list)
        self.image_label = np.array(image_label)
    # load
```

```
    def get_list(self):
        return self.image_list

    def get_label(self):
        return self.image_label

    def get_dataset(self): # (x, y) with y containing [person, subject, image_name]
        return  (self.image_list, self.image_label)

    def get_person(self,person):
        return self.image_list_person.get(person)

    def get_subject(self,subject):
        return self.image_list_subject.get(subject)

    def get_person_subject(self, person, subject):
        return self.image_list_person_subject.get(person + '.' + subject)

    def get_category_person(self):
        return sorted(list(self.image_list_person.keys()))

    def get_category_subject(self):
        return sorted(list(self.image_list_subject.keys()))


    def get_random_train_test(self, percent = 0.8): # (x_train, y_train,x_test,y_test)
        total  = len(self.image_list)
        mask = np.random.random_sample(total)<=percent
        return  ((np.array(self.image_list)[mask[:]], np.array(self.image_label)[mask[:]]), \
                (np.array(self.image_list)[mask[:]==False],
np.array(self.image_label)[mask[:]==False]))

    def plot_image(self, cnt):
        plot_image(self.image_list[cnt, :, :, :], self.image_label[cnt, :])
    # plot_image

    def plot_images(self, tfrom = 0, size = [4,4]):
        plot_images(self.image_list, self.image_label, tfrom, size=[4,4])
    # plot_image

def test_db():
    db = YaleFaceDb()
    images = db.get_list()
    labels = db.get_label()

    plot_images(images, labels)
    plot_image(images[10, :, :, :], labels[10, :])
# test_db

def plot_images(images, labels, start = 0, size = [4,4], wspace=1.5, hspace=1.5):
    r, c = size
    fig, axs = plt.subplots(r, c)
    cnt = 0
    for i in range(r):
        for j in range(c):
            if images.shape[3] == 1:
                axs[i,j].imshow(images[start+cnt, :,:, 0], cmap='gray')
            else:
                axs[i,j].imshow(images[start+cnt, :,:, :])
            axs[i,j].axis('off')
            axs[i,j].set_title('%s'%(labels[cnt, 0]))
            cnt += 1
    plt.subplots_adjust(wspace=wspace, hspace=hspace)
    plt.show()
# sample_images
```

```
def plot_image(image, label):
    plt.imshow(image[:,:,0], cmap = 'gray')
    plt.title('%s - %s'%(label[0], label[1]))
    plt.axis('off')
    plt.show()
# sample_images
```

3.2. Processing

eigenfaces.py

```
import matplotlib.pyplot as plt
import numpy as np

class EigenFaces(object):
    def __init__(self):
        self.mean_face = None
        self.eigen_faces = None

        self.vector_mean_matrix = None
        self.mean_vector = None
        self.eigen_value = None
        self.norm_ui = None

        self.weights = None
        self.size = None
        self.percent = 0.9

    # __init__

    def update(self, x_train, y_train):
        self.x_train = x_train
        self.y_train = y_train
        (self.mean_face, self.eigen_faces), (self.vector_mean_matrix, self.mean_vector,
self.eigen_value, self.norm_ui) = calculate_eigen_faces(self.x_train)


        self.size    = find_size(self.eigen_value, self.percent)
        self.weights = get_all_weight(self.x_train, self.mean_vector, self.norm_ui, self.size)
    # update

    def plot_mean_face(self):
        plot_image(self.mean_face, 'Mean Face')
    # plot_mean_face

    def plot_eigen_faces(self, start = 0, size = [4,4], wspace=1.5, hspace=1.5, fig_size =
(10, 10)):
        plot_images(self.eigen_faces, self.y_train, start, size, wspace, hspace, fig_size)
    # plot_mean_face

    def calc_weight(self, image):
        return get_weight(image.flatten(), self.mean_vector, self.norm_ui, self.size)

    def predict(self, image):
        weight_vector = self.calc_weight(image)
        (closest_face_id, norm_weight_vector) = distance_classify(weight_vector, self.weights)
        label = self.y_train[closest_face_id]
        return (label, closest_face_id, norm_weight_vector)

    def evaluation(self, x_test, y_test):
        cnt_true  = 0
        cnt_total = len(x_test)
        for cnt in range(len(x_test)):
            (predict_label, predict_closest_face_id, predict_norm_weight_vector) =
self.predict(x_test[cnt])
```

```
            truth_label = y_test[cnt]
            if predict_label[0] == truth_label[0]:
                cnt_true = cnt_true + 1
        return (cnt_true, cnt_total)
# def

# EigenFaces

def plot_images(images, labels, start = 0, size = [4,4], wspace=2.5, hspace=2.5, fig_size =
(10, 10)):
    r, c = size
    fig, axs = plt.subplots(r, c)
    plt.figure(figsize=fig_size, dpi=180)
    fig.set_size_inches(fig_size)
    cnt = 0
    for i in range(r):
        for j in range(c):
            if images.shape[3] == 1:
                axs[i,j].imshow(images[start+cnt, :,:, 0], cmap='gray', aspect='auto')
            else:
                axs[i,j].imshow(images[start+cnt, :,:, :], aspect='auto')
            axs[i,j].axis('off')
            if len(labels.shape)==1:
                axs[i,j].set_title('%s'%(labels[cnt]))
            elif len(labels.shape)==2 and len(labels[cnt]) == 1:
                axs[i,j].set_title('%s'%(labels[cnt, 0]))
            elif len(labels.shape)==2 and len(labels[cnt]) >= 2:
                axs[i,j].set_title('%s (%s)'%(labels[cnt, 0], labels[cnt, 1]))
            cnt += 1
    plt.subplots_adjust(wspace=wspace, hspace=hspace)
    plt.show()
# plot_images

def plot_image(image, label):
    if image.shape[2] == 1:
        plt.imshow(image[:,:,0], cmap='gray')
    else:
        plt.imshow(image[:,:,:])
    if type(label)==str:
        plt.title(label)
    if type(label) is np.ndarray:
        if len(label)==1:
            plt.title('%s'%(label[0]))
        elif len(label)>=2:
            plt.title('%s (%s)'%(label[0], label[1]))
    plt.axis('off')
    plt.show()
# plot_image

def gen_images():
    images = np.empty(shape=(3,4,4))
    images[0,:,:] = np.array([[1,2,3,4],[5,6,7,8],[5,6,7,8],[5,6,7,8]])
    images[1,:,:] = np.array([[2,3,4,5],[6,7,8,9],[5,6,7,8],[5,6,7,8]])
    images[2,:,:] = np.array([[0,2,4,6],[3,5,7,9],[5,6,7,8],[5,6,7,8]])
    return images

def convert_matrix_presentation(images):
    vector2d = []
    for image in images:
        vector = image.flatten()
        vector2d.append(vector)
    return np.array(vector2d)

def calculate_eigen_vectors(vector_matrix):
    mean_vector = vector_matrix.mean(axis=0)
    vector_mean_matrix = vector_matrix[:,:] - mean_vector
```

```
    covariance_matrix = np.matmul(vector_mean_matrix,vector_mean_matrix.T) # vector_matrix: [M
x N^2], [N^2 x M]
    u, eigen_value, eigen_vector_vi = np.linalg.svd(covariance_matrix)     # eigen_value: 1 x
M, eigen_vector_vi: M x M
    # vector_mean_matrix.T (N^2 x M) x eigen_vector_vi.T (M x 1) = N^2 x 1
    # M eigen vectors with high values
    eigen_vector_ui = np.matmul(vector_mean_matrix.T, eigen_vector_vi[:,:].T).T
    # normalize eigen vectors
    norms = np.linalg.norm(eigen_vector_ui, axis=1)   # N^2 x 1
    norm_ui = np.divide(eigen_vector_ui.T, norms).T   # 1 x N^2
    return (vector_mean_matrix, mean_vector, eigen_value, norm_ui) # M x N^2, 1 x N^2

def calculate_eigen_faces(images):
    vector_matrix = convert_matrix_presentation(images)
    (vector_mean_matrix, mean_vector, eigen_value, norm_ui) =
calculate_eigen_vectors(vector_matrix)
    eigen_faces = norm_ui.reshape(images.shape)
    mean_images = mean_vector.reshape(images.shape[1], images.shape[1], 1)
    return (mean_images, eigen_faces), (vector_mean_matrix, mean_vector, eigen_value, norm_ui)

def get_weight(face_vector, mean_vector, norm_ui, size):
    theta = face_vector - mean_vector
    return np.matmul(norm_ui[:size], theta)

def find_size(eigen_value, percent = 0.9):
    total = eigen_value.sum()
    for i in range(len(eigen_value)):
        size = i + 1
        cur  = eigen_value[:size].sum()
        if cur/float(total)>=percent:
            return size
    return len(eigen_value)

def get_all_weight(images, mean_vector, norm_ui, size):
    w = [get_weight(images[i,:,:].flatten(), mean_vector, norm_ui, size)for i in
range(images.shape[0])]
    return w

def distance_classify(w, weights):
    diff = weights - w
    norm_weight = np.linalg.norm(diff, axis=1)
    closest_face_id = np.argmin(norm_weight)
    return (closest_face_id, norm_weight)
```

4. Result

Yale Face Dataset with 80% Train and 20% Test and Accuracy 92%.

References

[1], [2] in document folder