

**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 Universitry**

**COMPUTER VISION – PROGRAMMING 1**

Tai Do Nhu (다이 도느), 176680, [donhutai@gmail.com](mailto:donhutai@gmail.com)

Table of Contents

[Table of Contents 3](#_Toc512118069)

[1. Eigen Face Recognition 3](#_Toc512118070)

[1.1. Main Idea 3](#_Toc512118071)

[2. Step by Step 3](#_Toc512118072)

[2.1. Tính toán các Eigenfaces 3](#_Toc512118073)

[2.2. Represent the available faces (training set) into this vector space 7](#_Toc512118074)

[2.3. Face recognition with EIGENFACES 7](#_Toc512118075)

[References 7](#_Toc512118076)

1. Eigen Face Recognition
   1. Main Idea

**Convert**

**Face Images**

**(NxN)**

**Vector**

**N2 x 1**

The problem that arises when identifying is that the dimension is too large (N2). 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 = a1v1 + a2v2 + … + aMvM 🡪 Y = b1u1 + b2u2 + … + bkuk



Không gian N chiều với hệ cơ sở *v1, v2,…,vn*

Không gian K chiều (K<<N) với hệ cơ sở *u1, u2,…,un*

Ánh xạ tuyến tính ***T*** cần tìm (phép chiếu)

Vector ***x*** ban đầu có ***N*** chiều

Vector ***y*** chỉ còn ***K***

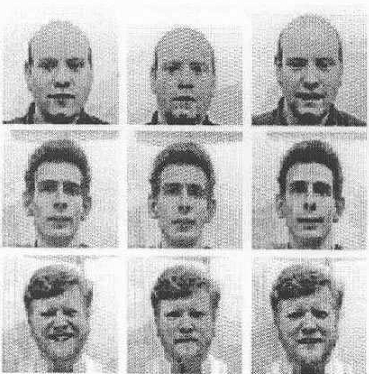


The PCA method will try to find the linear transformation T satisfying: y = T.x such that the mean squared error (MSE) is the smallest.

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.

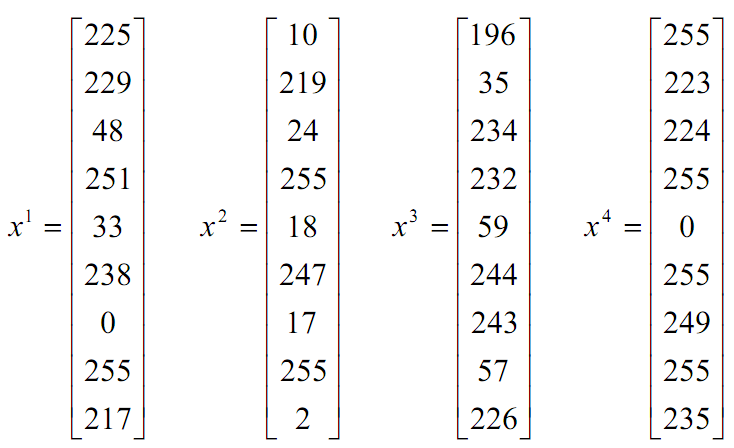
1. Step by Step
   1. Tính toán các Eigenfaces

**Step 1**: Use face images I1, I2, ... IM (face set training) with face right and all photos must be the same size.



**Step 2:** Reproduces all Ii images into vector 

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



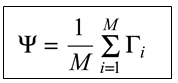
Γ1

Γ2

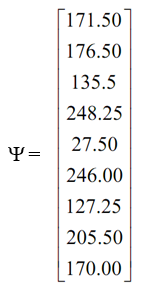
Γ3

Γ4

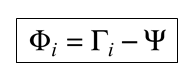
**Step 3**: Calculate the average face vector by the formula:

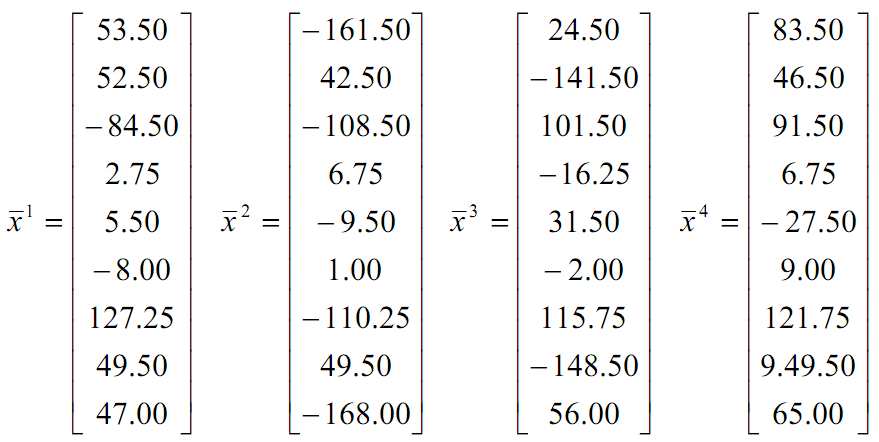


Specifically we have:

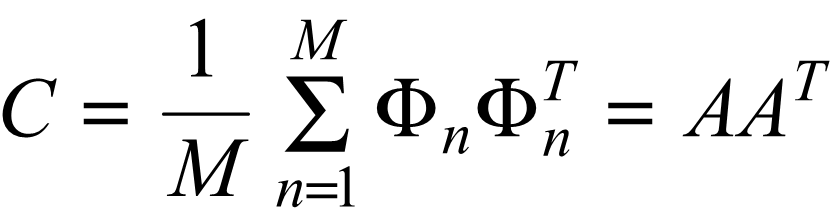


**Step 4**: Minus nfor the average face vector. Specifically we have:



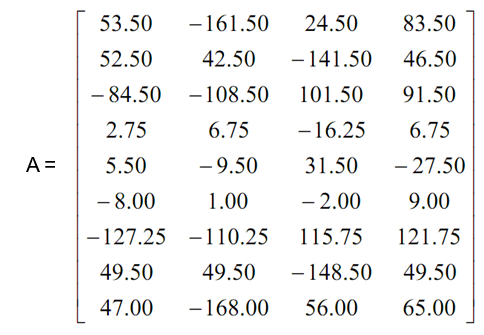


**Step 5**: Calculate covariance C with N2 x N2

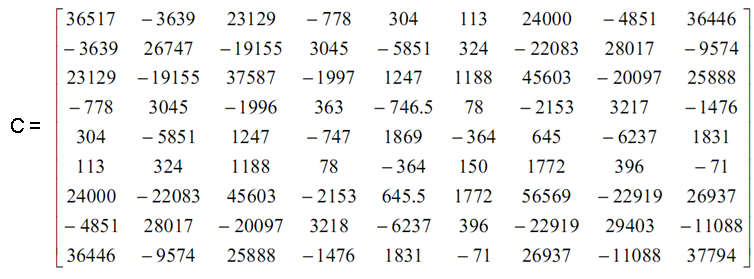


Where A with N2 x M:





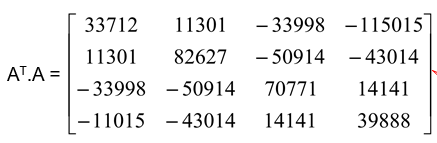
With C such as:



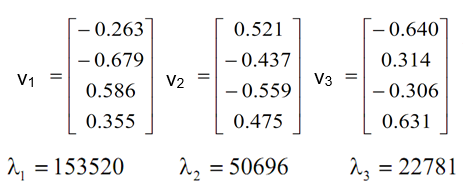
**Step 6**: Calculates the eigenvector ui ("private vector") of the A.AT square matrix (C is N2xN2).

This matrix is too large and not feasible.

**Step 6.1**: Consider the matrix AT.A (note that this matrix is only M x M size)

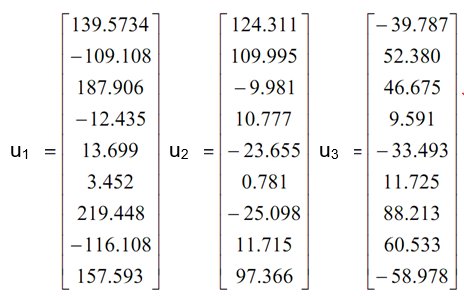


**Step 6.2:** Calculates the eigenvectors of this AT.A square matrix

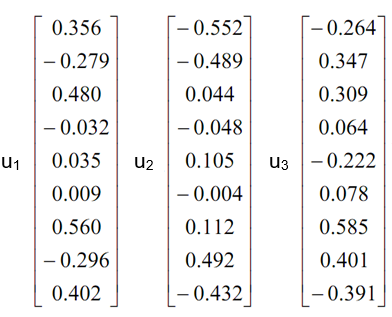


After calculating the microscopic vectors (size Mx1), it is easy to deduce the individual vectors ui (size N2x1) desired by the formula:





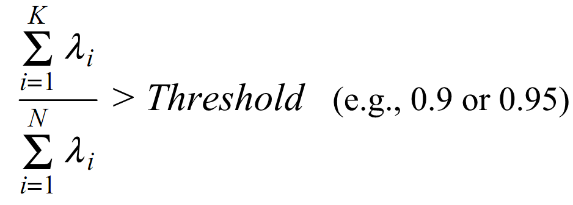
Note the normalization of vectors u­i: , . After normalization, we obtain the last vectors as follows:



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

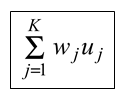
**Step 7**: Only retain the vector K of the above-mentioned M vector (corresponding to the largest K value), of course K << N2.

Select K by the following formula:

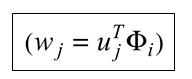


* 1. Represent the available faces (training set) into this vector space

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



Where

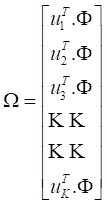


* 1. Face recognition with EIGENFACES

Standardized



Performed as follows:



Find min:



1. Python Code
   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

* 1. 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)

1. Result

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

References

[1], [2] in document folder