

**EE4208 Intelligent System Design**

**Assignment 1: Face Recognition**

**Submitted by:**

**Ng Yi Rong (U1820156H)**

**Wang Jue (U1820975L)**

**Zheng Yicheng (U1822468G)**

School of Electrical & Electronics Engineering

Academic Year 2020/2021

**Table of Contents**

[Chapter 1: Introduction 3](#_Toc68866720)

[Chapter 2: Literature Review 3](#_Toc68866721)

[Chapter 3: Data Collection 7](#_Toc68866722)

[Chapter 4: Face Detection 8](#_Toc68866723)

[Chapter 5: Face Training 9](#_Toc68866724)

[Chapter 6: Real-Time Face Recognition 10](#_Toc68866725)

[Chapter 7: Results and Discussion 11](#_Toc68866726)

[References 12](#_Toc68866727)

[Appendices 13](#_Toc68866728)

[Appendix A: Source Code for Eigenface Approach 13](#_Toc68866729)

[Appendix B: Source Code for Illumination Normalization 14](#_Toc68866730)

[Appendix C: Source Code for Capturing Faces 15](#_Toc68866731)

[Appendix D: Source Code for Loading Face Images. 17](#_Toc68866732)

[Appendix E: Source Code for Training Face Recognizer 18](#_Toc68866733)

[Appendix F: Source Code for Real-Time Face Recognition 19](#_Toc68866734)

[Appendix G: Source Code to Run Face Recognition Program 22](#_Toc68866735)

# Chapter 1: Introduction

The main objective of this project is to design a real-time face recognition system using state-of-the-art face detection and face recognizer algorithms. The scope of this project covers literature review, data collection, face detection, face training, real-time face recognition, and results and discussion.

The resources needed in this project are mainly software applications. Python programming language is used with several other packages such as Numpy [1], Scikit-Learn [2], and OpenCV [3] packages.

# Chapter 2: Literature Review

The traditional method to perform face recognition is by the eigenface approach [4] based on Principal Component Analysis (PCA). The pipeline of eigenface approach is shown in Figure 1.

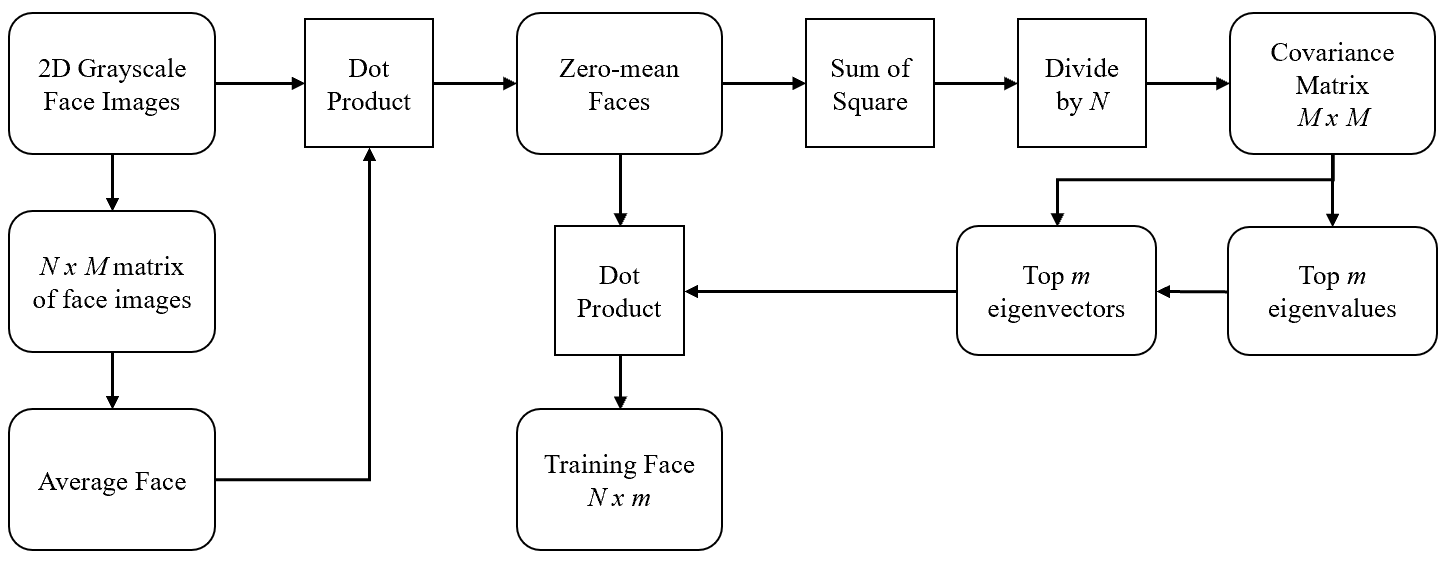


Figure 1: Pipeline of Eigenface Approach.

Firstly, all the 2D grayscale images are flattened into 1D and combined into a single matrix. The average face of the whole face dataset is computed by taking the sum of each pixel location and dividing it by the number of face images. The average face is computed as follows:

The purpose of average face is to subtract from each face image so that illumination is removed. The zero-mean face is computed as follows:

Next, covariance matrix is computed by taking the square of zero-mean face images and dividing by the total number of face samples. The covariance matrix is computed as follows:

This covariance matrix is used to calculate the eigenvalues and eigenvectors. Only the top *m* eigenvalues are important features for face recognition where *m* << total number of features. Dot product of zero-mean faces and *m* eigenvectors is computed to project zero-mean faces into eigenspace. Subsequently, these new faces in eigenspace are used as training faces for training the classifier. The source code for eigenface approach can be found in [Appendix A](#_Appendix_A:_Source). The training faces are computed as follows:

|  |  |
| --- | --- |
|  | , where *v* is the eigenvector |

Although the eigenface approach is fast, it has poor recognition accuracy due to lighting change. Face averaging is not an optimal solution to remove illumination. Another method to remove illumination is by RGB/BGR to luma-chroma conversion as shown in Figure 2. This method converts 2D RGB face images into luma-chroma channels. As shown in Figure 3, this method can reduce background illumination effect. After removing the background illumination effect, the face images will be converted to grayscale for eigenface approach.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| RGB Face | Luma-Chroma Face | Grayscale Face |

Figure 2: Illumination Removal by Luma-Chroma Conversion Method.

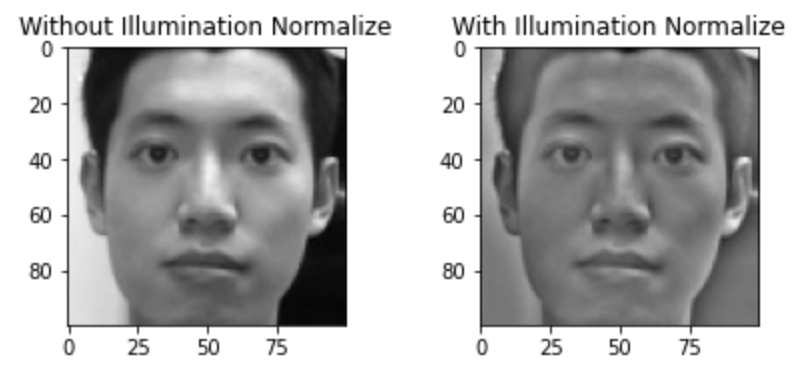


Figure 3: Background Illumination Effect.

By comparing the two processed images in Figure 3, the image processed by luma-chroma conversion method has better illumination effect reduction than the image processed by average face. The source code for illumination normalization can be found in [Appendix B](#_Appendix_B:_Source).

After illumination effect reduction, average subtraction and PCA, all the training face images will be used to train a SVM [5] classifier supported by scikit learn package [2]. Our experimental results have shown that SVM classifier cannot detect a face with mask. Therefore, we proposed another approach which is based on Local Pattern Binary Histogram (LBPH) Face Recognizer [6]. The LBPH face recognition process is illustrated in Figure 4.

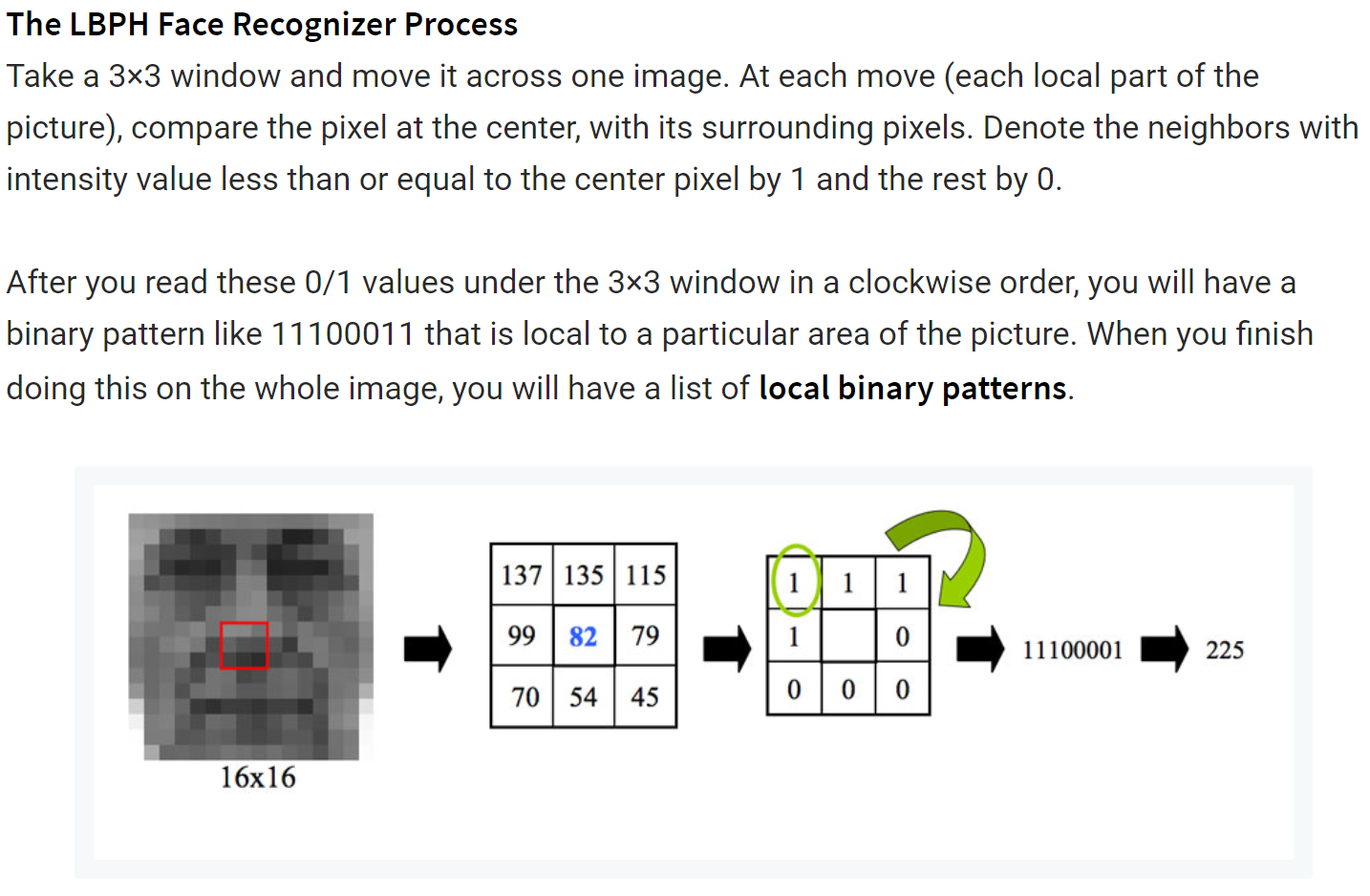


Figure 4: Illustration of LBPH Face Recognition Process. Extracted from [7].

Binary patterns were converted to decimal numbers. The number of counts for each decimal value (pixel) was computed in a histogram. One histogram for each face image in the training data. If there are 100 face images, LBPH will extract 100 histograms after training and store them for face recognition stage. This algorithm also keeps track of which histogram belongs to which person. During recognition stage, a new histogram is generated for new face image. The new histogram is compared with the historgrams obtained during training and return the best match.

# Chapter 3: Data Collection

The training dataset contains 100 face samples from 20 different person. Each person has 5 face images of different facial expressions. Each face image is resized to dimensions. Sample face images of one person is shown in Table 1.

Table 1: Sample Faces of One Person. Each face image is 100x100.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| No Glasses, No Smile | No Glasses, Smile | No Glasses, Show Teeth | Wear Glasses, No Smile | Wear Glasses, Show Teeth |
|  |  |  |  |  |

# Chapter 4: Face Detection

The face images obtained in Table 1 were captured via laptop’s built-in webcam. Region of Interest (ROI) of face was detected using the pre-trained cascade classifier from OpenCV called Haar Classifier [8]. There are several parameters such as scale factor, minimum neighbours, and minimum window size to be tuned so that the classifier can successfully detect a human face.

When a human face is presented in front of the webcam, the image is converted into grayscale because the face classifier can only work on grayscale data. If a face is detected in the image using *haarcascade\_frontalface\_dafault* detector, a rectangle is drawn to mark it as a ROI. This ROI is captured and resized to 100 x 100 dimensions which will be used for face training.

Additionally, eye classifier was used to detect eyes in faces. This is necessary to detect a person wearing face mask. Face mask covers most of the important face features such as nose and mouth. As a result, face detector alone will not work.

After the face ROI is marked, the eye classifier will scan through the face ROI to detect eyes. However, we noticed that the eye classifier cannot detect eyes when a person is wearing glasses with reflective lens. Therefore, we used eyeglasses classifier, *haarcascade\_eye\_tree\_eyeglasses,* detector instead which has better detection performance. Sample ROI face image is shown in Figure 5. The source code for face capturing can be found in [Appendix C](#_Appendix_C:_Source).

|  |
| --- |
|  |
| Figure 5: Sample ROI Face Image. |

# Chapter 5: Face Training

All 100 faces were used as training data to train the face recognizer. These faces were extracted from database via *get\_faces\_labels()* function (refer to [Appendix D](#_Appendix_D:_Source)). All extracted faces were represented in an array format of shape (100, 100, 100), where each face is a 2D array of pixel values. Each face was labelled with an integer value ranges from 0 to 19 and stored in an array. The person name and its corresponding label are stored in a dictionary as shown in Figure 6.

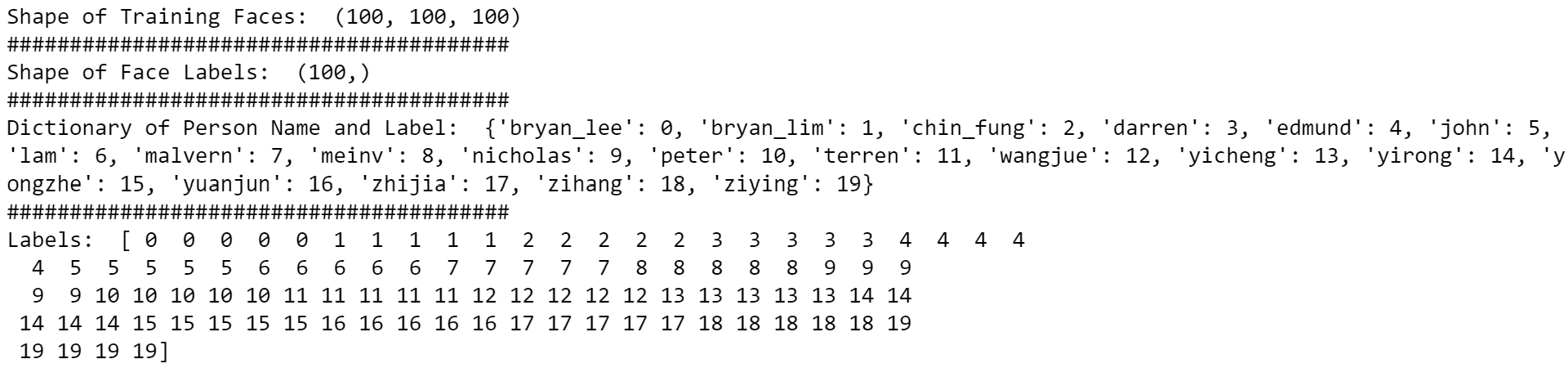


Figure 6: Outputs of *get\_faces\_labels()* function.

The face recognizer used in this project is Local Binary Pattern Histogram [6]. Training data was passed to LBPH Face Recognizer for training and the trained model was saved for real-time face recognition. The source code for face training can be found in [Appendix E](#_Appendix_E:_Source).

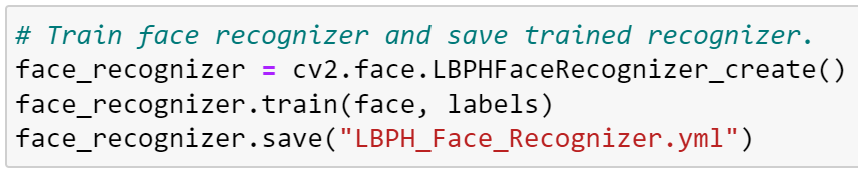


Figure 7: Code Segment to Train Face Recognizer.

# Chapter 6: Real-Time Face Recognition

Real-time face recognition was performed using the trained LBPH face recognizer and Haar Classifiers. The webcam will capture a person face and scan for faces and eyes using *haarcascade\_frontalface\_dafault* and *haarcascade\_eye\_tree\_eyeglasses* detectors. If a person not wearing face mask, face and eyes will be detected. The detected region also known as ROI will be resized to 100 x 100 and passed to the LBPH face recognizer for prediction. If a person is wearing face mask, face detector will not be able to detect face. Hence, eye detector is used to detect eyes which are important feature in a person face.

The prediction outputs the predicted label and its confidence value. Confidence value determines how different is the new face to the ground truth. High confidence value means the new face is less similar to the ground truth, and vice versa. A threshold of 120 was set for the confidence value. If the predicted new face has a confidence value less than 120, it returns the person identity. Otherwise, it returns ‘unknown face’. The source code for real-time face recognition can be found in [Appendix F](#_Appendix_F:_Source) and [Appendix G](#_Appendix_G:_Source).

# Chapter 7: Results and Discussion

We have experimented several approaches for face recognition as shown in Table 2. SVM classifier’s accuracy is low and sensitive to brightness difference. Although luma-chroma conversion approach was used, the recognition accuracy of SVM is poor.

We adopted a second approach which used VGGFace CNN descriptors [9] to fit on our training face images. Then a new face was compared with the training faces using cosine similarity. A matched face returns a cosine similarity close to 1, otherwise close to 0. This approach has good recognition accuracy but there is a lot of delay in real-time recognition.

The third approach which is also our proposed approach can perform real-time face recognition almost instantly. Furthermore, this approach can recognize a person with or without face mask.

Table 2: List of Approaches Experimented.

|  |  |
| --- | --- |
| Approach 1 | Eigenface + SVM |
| Approach 2 | VGGFace Feature Extraction + Cosine Similarity |
| Approach 3 | LBPH Face Recognizer |

# References

[1] C. R. Harris *et al.*, "Array programming with NumPy," *Nature,* vol. 585, no. 7825, pp. 357-362, 2020/09/01 2020, doi: 10.1038/s41586-020-2649-2.

[2] L. Buitinck *et al.*, "API design for machine learning software: experiences from the scikit-learn project," *arXiv preprint arXiv:1309.0238,* 2013.

[3] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools,* 2000.

[4] M. Turk and A. Pentland, "Eigenfaces for recognition," *Journal of cognitive neuroscience,* vol. 3, no. 1, pp. 71-86, 1991.

[5] V. Vapnik, I. Guyon, and T. Hastie, "Support vector machines," *Mach. Learn,* vol. 20, no. 3, pp. 273-297, 1995.

[6] S. Liao, X. Zhu, Z. Lei, L. Zhang, and S. Z. Li, "Learning multi-scale block local binary patterns for face recognition," in *International Conference on Biometrics*, 2007: Springer, pp. 828-837.

[7] R. Raja. "Face recognition using OpenCV and Python: A beginner's guide." SuperDataScience Team. https://www.superdatascience.com/blogs/opencv-face-recognition (accessed 09 April, 2021).

[8] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, 2001, vol. 1: IEEE, pp. I-I.

[9] O. M. Parkhi, A. Vedaldi, and A. Zisserman, "Deep Face Recognition," 2015.

# Appendices

## Appendix A: Source Code for Eigenface Approach

def normalise\_face(image, n=100, m=100):

# Find average face based on face dataset, return shape (100, 10000)

avg\_face = np.mean(image, axis=0)

print("avg face (1,10000): ", avg\_face.shape)

# Compute zero mean faces, return shape (50, 10000)

zeromean\_face = image - avg\_face

print("zero mean face (n,10000): ", zeromean\_face.shape)

# Compute covariance matrix, return shape (10000, 10000)

covariance = np.dot(zeromean\_face.T, zeromean\_face) / n

print("covariance matrix shape (10000, 10000): ", covariance.shape)

total\_features = image.shape[1] # 10000 features

print("Calculaing top eigenvalues and corresponding eigenvectors...")

# Compute eigenval and eigenvec, return shape (1, m) and (10000, m)

eigenvalues, eigenvectors = linalg.eigh(covariance, eigvals=(total\_features-m,total\_features-1))

print("eigenvalues shape (m,): ", eigenvalues.shape)

print("eigenvectors shape (10000, m): ", eigenvectors.shape)

print("Eigens Computation Done!")

# Compute eigenfaces, return shape (m, 10000)

eigenfaces = eigenvectors.T

print("eigenface shape (m,10000)", eigenfaces.shape)

# Project zero mean faces into eigen space for training, return shape (50, m)

face\_train = np.dot(zeromean\_face, eigenvectors)

print("face\_train (n,m): ", face\_train.shape)

return avg\_face, zeromean\_face, eigenvalues, eigenvectors, eigenfaces, face\_train, covariance

## Appendix B: Source Code for Illumination Normalization

def illumination\_normalize(array1d):

image\_bgr=[]

image\_ycrcb=[]

image\_gray=[]

for i in array1d:

image\_reshape = i.reshape(100,100)

bgr\_image = cv2.cvtColor(image\_reshape, cv2.COLOR\_GRAY2BGR) # Convert gray to bgr

ycrcb\_image = cv2.cvtColor(bgr\_image, cv2.COLOR\_BGR2YCrCb) # Convert bgr to ycrcb

# separate channels

y, cr, cb = cv2.split(ycrcb\_image)

# get background which paper says (gaussian blur using standard deviation 5 pixel for 300x300 size image)

# account for size of input vs 300

sigma = int(5 \* 100 / 100)

#print('sigma: ',sigma)

gaussian = cv2.GaussianBlur(y, (0, 0), sigma, sigma)

# subtract background from Y channel

y = (y - gaussian + 100)

# merge channels back

ycrcb = cv2.merge([y, cr, cb])

#convert to BGR

output\_bgr = cv2.cvtColor(ycrcb, cv2.COLOR\_YCrCb2BGR)

#convert to grayscale

output\_gray = cv2.cvtColor(output\_bgr, cv2.COLOR\_BGR2GRAY) # 2D

output\_gray1d = output\_gray.reshape(-1) # 1D

image\_gray.append(output\_gray1d)

image\_gray\_array = np.array(image\_gray)

image\_bgr.append(bgr\_image)

image\_bgr\_array = np.array(image\_bgr)

image\_ycrcb.append(ycrcb\_image)

image\_ycrcb\_array = np.array(image\_ycrcb)

return image\_gray\_array, image\_bgr\_array, image\_ycrcb\_array

## Appendix C: Source Code for Capturing Faces

import os, sys

import cv2

from PIL import Image

import numpy as np

import pickle

import time

import matplotlib.pyplot as plt

%matplotlib inline

# Take face shots via webcam to create face database

cam = cv2.VideoCapture(0)

cam.set(3, 1280) # set video width

cam.set(4, 960) # set video height

face\_detector = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml')

eyes\_detector = cv2.CascadeClassifier('haarcascade\_eye.xml')

eyes\_glasses\_detector = cv2.CascadeClassifier('haarcascade\_eye\_tree\_eyeglasses.xml')

# For each person, enter one numeric face id

user\_name = input('\n Enter name and press <enter> ==> ')

print("\n Look at camera and press <spacebar> to take picture")

# Initialize individual sampling face count

count = 1

while(True):

ret, img = cam.read()

img = cv2.flip(img, 1) # flip video image vertically

gray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

faces = face\_detector.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5, minSize=(5,5))

for (x,y,w,h) in faces:

cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0), 2)

roi\_gray = gray[y:y+h, x:x+w]

roi\_color = img[y:y+h, x:x+w]

resized\_color = cv2.resize(img[y:y+h,x:x+w], dsize=(224,224), interpolation=cv2.INTER\_AREA)

eyes = eyes\_glasses\_detector.detectMultiScale(roi\_gray, scaleFactor=1.2, minNeighbors=5, minSize=(5,5));

for (ex, ey, ew, eh) in eyes:

cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

#cv2.rectangle(roi\_gray, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

resized\_gray = cv2.resize(gray[y:y+h,x:x+w], dsize=(100,100), interpolation=cv2.INTER\_AREA)

k = cv2.waitKey(1) & 0xff # Press 'ESC' for exiting video

if k == 27:

break

elif k == 32: #if spacebar is pressed

# Save the captured image into the datasets folder

cv2.imwrite("ROI\_Faces/User." + str(user\_name).lower() + "." + str(count) + ".jpg", roi\_color)

cv2.imwrite("Grayscale\_Faces/User." + str(user\_name).lower() + '.'+ str(count) + ".jpg", resized\_gray)

cv2.imwrite("Color\_Faces/User." + str(user\_name).lower() + '.' + str(count) + ".jpg", resized\_color)

print("Shot {} is taken!".format(count))

count += 1

cv2.imshow('image', img)

cam.release()

cv2.destroyAllWindows()

## Appendix D: Source Code for Loading Face Images.

def get\_faces\_labels(resized\_images\_path='Resized\_Faces'):

file\_path = os.listdir(resized\_images\_path)

faces = []

face\_labels = []

current\_id=0

label\_ids={}

for file in file\_path:

if file.endswith("png") or file.endswith("jpg"):

path=os.path.join(resized\_images\_path, file)

label=os.path.basename(path).split(".")[1]

#print(label,path)

if not label in label\_ids:

#label\_ids[label] = os.path.basename(path).split(".")[2]

label\_ids[label]=current\_id

current\_id+=1

id\_=label\_ids[label]

pil\_image=Image.open(path)#grayscale

image\_array=np.array(pil\_image,'uint8')

#print(image\_array)

faces.append(image\_array)

face\_labels.append(id\_)

faces\_array = np.array(faces)

labels\_array = np.array(face\_labels)

return faces\_array, labels\_array, label\_ids

## Appendix E: Source Code for Training Face Recognizer

# Train face recognizer and save trained recognizer.

face\_recognizer = cv2.face.LBPHFaceRecognizer\_create()

face\_recognizer.train(face, labels)

face\_recognizer.save("LBPH\_Face\_Recognizer.yml")

## Appendix F: Source Code for Real-Time Face Recognition

#17 Real-Time Detection using webcam after training the face recognizer

def lbph\_face\_recognition():

faceCascade = cv2.CascadeClassifier('haarcascade\_frontalface\_default.xml') # load classifier

eyeCascade = cv2.CascadeClassifier('haarcascade\_eye.xml')

eye\_glassesCascade = cv2.CascadeClassifier('haarcascade\_eye\_tree\_eyeglasses.xml')

smileCascade = cv2.CascadeClassifier('haarcascade\_smile.xml')

#faceCascade = cv2.CascadeClassifier('lbpcascade\_frontalface.xml') # load classifier

cap = cv2.VideoCapture(0)

#face\_recognizer = cv2.face.EigenFaceRecognizer\_create()

face\_recognizer=cv2.face.LBPHFaceRecognizer\_create()

face\_recognizer.read("LBPH\_Face\_Recognizer.yml")

#Load face labels

# with open('face\_eye\_smile\_labels.pickle', 'rb') as f:

# y\_train = np.array(pickle.load(f))

names = ['Bryan\_Lee', 'Bryan\_Lim', 'Chin\_Fung', 'Darren', 'Edmund', 'John', 'Lam',

'Malvern', 'meinv', 'Nicholas', 'Peter', 'Ter\_Ren', 'Wang\_Jue', 'Yi\_Cheng',

'Yi\_Rong', 'Yong\_Zhe', 'Yuan\_Jun', 'Zhi\_jia', 'Zi\_Hang', 'Zi\_Ying']

cap.set(3,640) # set Width

cap.set(4,480) # set Height

while True:

\_, frame = cap.read()

frame = cv2.flip(frame, 1)

gray = cv2.cvtColor(frame, cv2.COLOR\_BGR2GRAY)

# detect faces

faces = faceCascade.detectMultiScale(

gray,

scaleFactor=1.2,

minNeighbors=5,

minSize=(5, 5)

)

found=0

for (x,y,w,h) in faces:

cv2.rectangle(frame,(x,y),(x+w,y+h),(255,0,0),2)

roi\_color = frame[y:y+h, x:x+w]

roi\_gray = gray[y:y+h, x:x+w]

# roi\_gray = cv2.resize(roi\_gray, (100,100))

eyes = eye\_glassesCascade.detectMultiScale(roi\_gray, scaleFactor=1.2, minNeighbors=5, minSize=(5,5))

for (ex, ey, ew, eh) in eyes:

cv2.rectangle(roi\_color, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

# smile = smileCascade.detectMultiScale(roi\_gray, scaleFactor=1.2, minNeighbors=25, minSize=(120,120))

# for (xx, yy, ww, hh) in smile:

# cv2.rectangle(roi\_color, (xx, yy), (xx + ww, yy + hh), (0, 0, 255), 2)

gray\_face = cv2.resize((gray[y:y+h,x:x+w]),(100,100))

label, conf = face\_recognizer.predict(gray\_face)

found=1

if (found==1):

if conf<=120:

person = names[label]

else:

person = "Unknown"

text = str(label) + person + ":" + str(round(conf,3))

cv2.rectangle(frame, (x, y), (x + w, y + h), (255, 0, 0), 2)

cv2.putText(frame, text, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (255, 0, 0), 2)

elif (found==0):

eyes = eye\_glassesCascade.detectMultiScale(gray, scaleFactor=1.2, minNeighbors=5, minSize=(5,5))

for (ex, ey, ew, eh) in eyes:

roi\_gray = gray[ey:ey+eh, ex:ex+ew]

cv2.rectangle(frame, (ex, ey), (ex + ew, ey + eh), (0, 255, 0), 2)

# smile = smileCascade.detectMultiScale(roi\_gray, scaleFactor=1.2, minNeighbors=25, minSize=(120,120))

# for (xx, yy, ww, hh) in smile:

# cv2.rectangle(roi\_color, (xx, yy), (xx + ww, yy + hh), (0, 0, 255), 2)

#gray\_eyes = cv2.resize((gray[y:y+h,x:x+w]),(100,100))

label, conf = face\_recognizer.predict(gray)

if conf<=120:

person = names[label]

else:

person = "Unknown"

text = str(label) + person + ":" + str(round(conf,3))

cv2.rectangle(frame, (ex, ey), (ex + ew, ey + eh), (255, 0, 0), 2)

cv2.putText(frame, text, (ex, ey - 10), cv2.FONT\_HERSHEY\_SIMPLEX, 0.9, (255, 0, 0), 2)

cv2.imshow('frame',frame)

k = cv2.waitKey(1) & 0xff

if k == 27: # press 'ESC' to quit

break

cap.release()

cv2.destroyAllWindows()

## Appendix G: Source Code to Run Face Recognition Program

# Load labelled face datasets

face, labels, name\_label\_dict = get\_faces\_labels()

print("Shape of Training Faces: ",face.shape)

print("########################################")

print("Shape of Face Labels: ",labels.shape)

print("########################################")

print("Dictionary of Person Name and Label: ",name\_label\_dict)

print("########################################")

print("Labels: ",labels)

# Run Real-Time Face Recognition

lbph\_face\_recognition()