NANYANG TECHNOLOGICAL UNIVERSITY

EE4208 Intelligent System Design

Assignment 1: Face Recognition

Submitted by:

Ng Yi Rong (U1820156H) Group: F42

Wang Jue (U1820975L) Group: F41

Zheng Yicheng (U1822468G) Group: F41

School of Electrical & Electronics Engineering

Academic Year 2020/2021

Table of Contents

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.

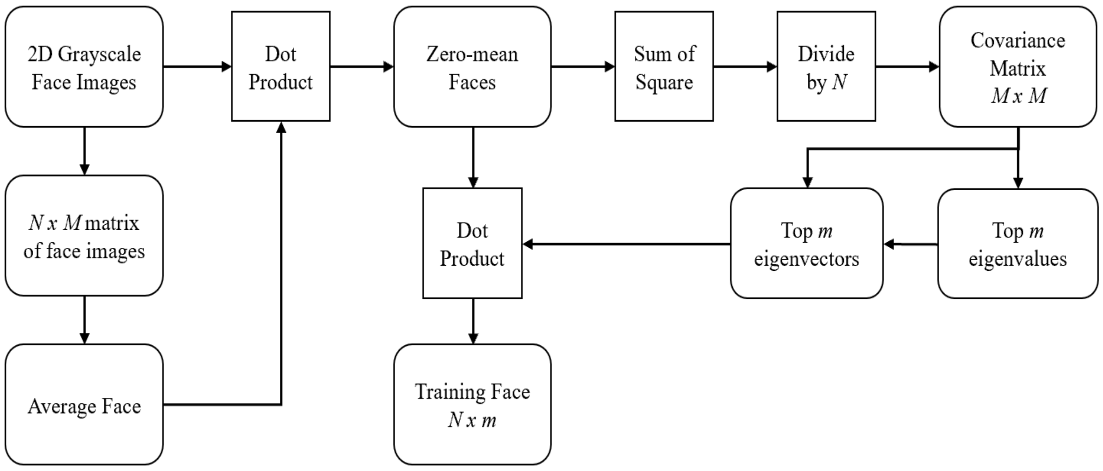


Figure 1: Pipeline of Eigenface Approach.

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:

$$f_{ave} = \frac{1}{n}\sum_{i=1}^{n} f_i$$

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:

$$f_{zero\_mean} = f - f_{ave}$$

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:

$$C = \frac{1}{n}\sum_{i}^{n}\left(f_{i,zero\_mean}\right)^{T}\left(f_{i,zero\_mean}\right)$$

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. The training faces are computed as follows:

$$Training\ Face = \left(f_{zero\_mean}\right)^{T} v$$, 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.

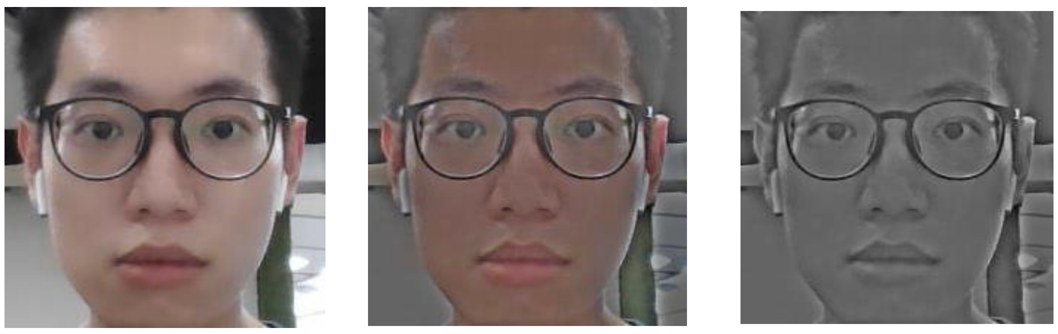


Figure 2: Illumination Removal by Luma-Chroma Conversion Method.

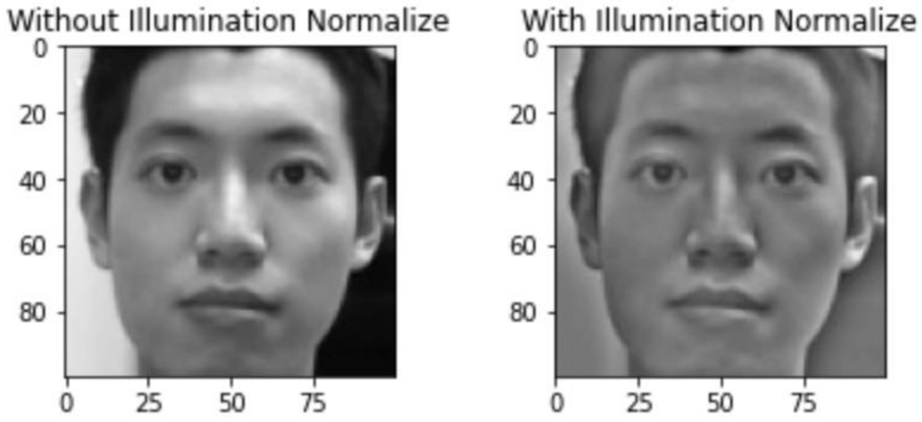
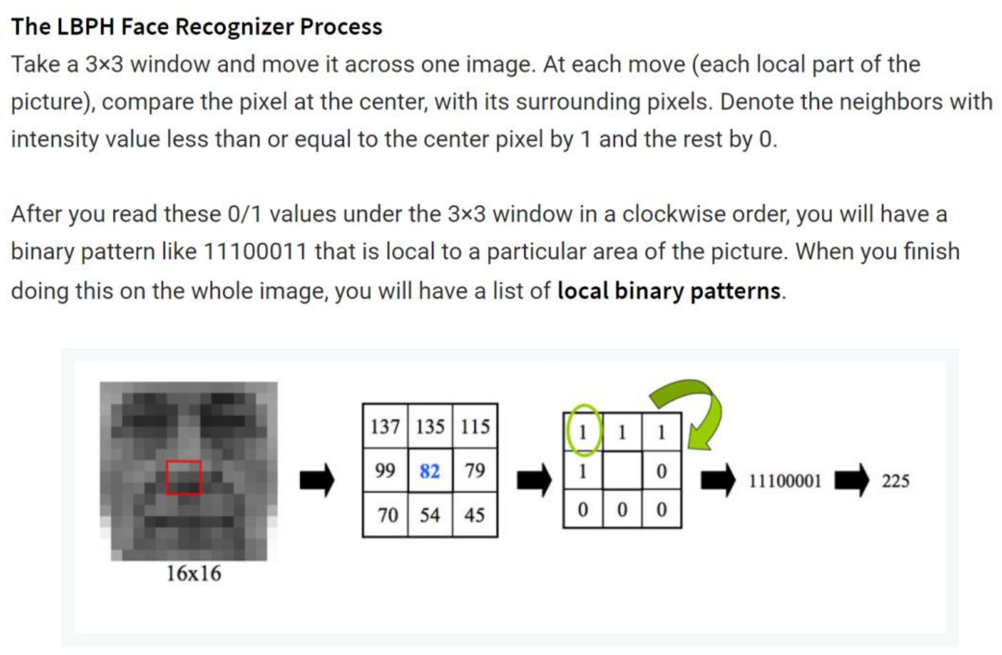


Figure 3: Background Illumination Effect.

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.

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.



The LBPH Face Recognizer Process

Take a 3×3 window and move it across one image. At each move (each local part of the picture), compare the pixel at the center, with its surrounding pixels. Denote the neighbors with intensity value less than or equal to the center pixel by 1 and the rest by 0.

After you read these 0/1 values under the 3×3 window in a clockwise order, you will have a binary pattern like 11100011 that is local to a particular area of the picture. When you finish doing this on the whole image, you will have a list of **local binary patterns**.

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 histograms obtained during training via Euclidean Distance.

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. The forehead must not be covered with hair. This is to ensure enough Haar-like features for face detection and recognition with mask. Each face image is resized to 100×100 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 because face mask covers most of the important face features such as nose and mouth. Therefore, forehead and eye are used as Haar-like features when face mask is worned.

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.

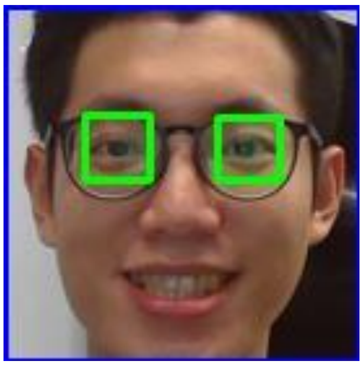
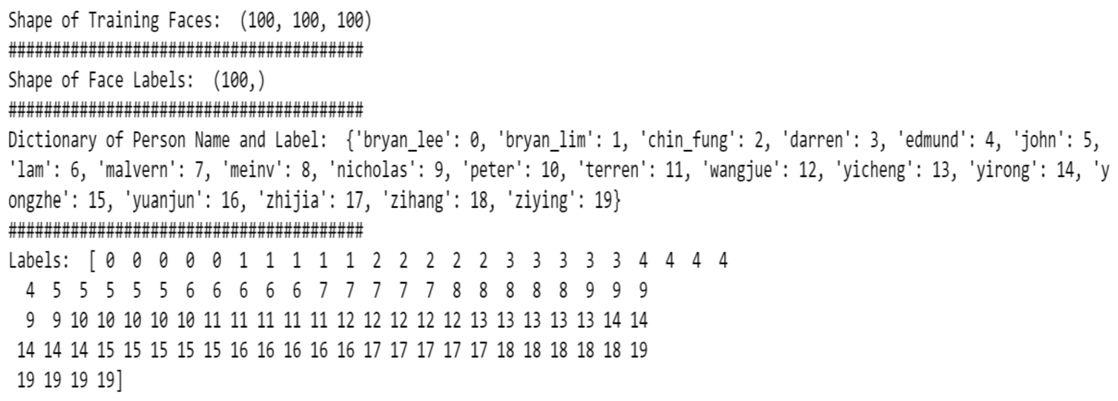


Figure 5: Sample ROI Face Image.

Chapter 5: Face Training

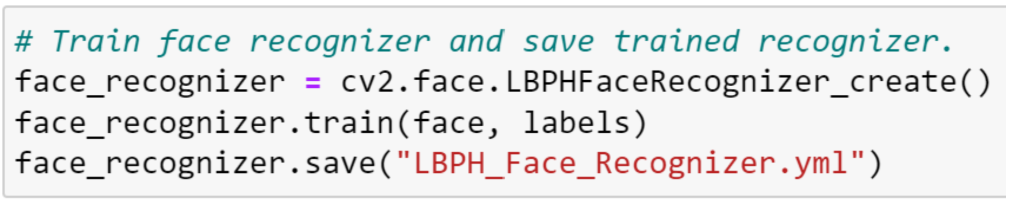
All 100 faces were used as training data to train the face recognizer. These faces were extracted from database via *get_faces_labels()* function (refer to Appendix D). All extracted faces were represented in an array format of shape (100, 100, 100), where each face is a 2D array of pixel values. Each 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.



```
Shape of Training Faces:  (100, 100, 100)
########################################
Shape of Face Labels:  (100,)
########################################
Dictionary of Person Name and Label:  {'bryan_lee': 0, 'bryan_lim': 1, 'chin_fung': 2, 'darren': 3, 'edmund': 4, 'john': 5,
'lam': 6, 'malvern': 7, 'meinv': 8, 'nicholas': 9, 'peter': 10, 'terren': 11, 'wangjue': 12, 'yicheng': 13, 'yirong': 14, 'y
ongzhe': 15, 'yuanjun': 16, 'zhijia': 17, 'zihang': 18, 'ziying': 19}
########################################
Labels:  [ 0  0  0  0  0  1  1  1  1  1  2  2  2  2  2  3  3  3  3  3  4  4  4  4
  4  5  5  5  5  5  6  6  6  6  6  7  7  7  7  7  8  8  8  8  8  9  9  9
  9  9 10 10 10 10 10 11 11 11 11 11 12 12 12 12 12 13 13 13 13 13 14 14
 14 14 14 15 15 15 15 15 16 16 16 16 16 17 17 17 17 17 18 18 18 18 18 19
 19 19 19 19]
```

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.



```
# 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")
```

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 computed by Euclidean Distance. 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 and Appendix G.

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 with high speed and accuracy. 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

```
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(3,1280) # set Width
    cap.set(4,960) # 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)
        )

        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)

                if conf<=100:
                    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)



                print(person, round(conf,1))
            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()
```