FACE DETECTION AND RECOGNITION USING OPENCV

MINI PROJECT REPORT

Submitted by

KARAN KUMAR(12180048)
MANU SHARMA(12180052)

In partial fulfillment of the requirements for the award of the Degree of Bachelor and Technology in Computer Science and Engineering

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF ENGINEERING

COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

MAY 2020

DIVISION OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF ENGINEERING
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY

CERTIFICATE

Certified that this is a bonafide record of the Mini Project titled

FACE DETECTION AND RECOGNITION USING OPENCV

done by
Manu Sharma(12180052)
Karan Kumar(12180048)

of VI Semester, Computer Science and Engineering in the year 2020 in partial fulfillment requirements for the award of Degree of Bachelor of Technology in Computer Science and Engineering of Cochin University of Science and Technology.

Dr. Latha R Nair	**V. Damodaran**
Head of Division & Project Coordinator	Project Guide

ACKNOWLEDGEMENT

We take this opportunity to express our deep sense of gratitude and sincere thanks to all who helped us to complete the work successfully. Our first and foremost thanks goes to God Almighty who showered in immense blessings on our effort. We wish to express our sincere thanks to **V. Damodaran** for providing us with all the necessary facilities and support.We would like to express our sincere gratitude to **Dr. Latha R. Nair**(Head of the department), for her support and co-operation. We wish to express our sincere gratitude towards all the teaching and non teaching staff members of our Department. Finally we thank our parents, all our friends, near and dear ones who directly and indirectly contribute to the successful of this work.

Thank You,

KARAN KUMAR(12180048)
MANU SHARMA(12180052)

DECLARATION

We undersigned hereby declare that the project report **Face Detection and Recognition using OpenCV** , submitted for partial fulfillment of the requirements for the award of the Degree of Bachelor and Technology in Computer Science and Engineering of the School of Engineering, Cochin University of Science and Technology, Kerala is a bonafide work done by us. This submission represents our ideas in our own words and where ideas or words of others have been included, We have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in my submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and/or the University and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed the basis for the award of any degree, diploma or similar title of any other University.

KARAN KUMAR(12180048)
MANU SHARMA(12180052)

ABSTRACT

Face Detection and Recognition from an image or a video is a popular topic in computer vision. Face recognition technology has widely attracted attention due to its enormous application value and market potential, such as real-time video surveillance system. It is widely acknowledged that the face recognition has played an important role in surveillance system as it doesn't need the object's co-operation. We design a real-time face recognition system based on input camera and image set algorithm by way of OpenCV and Python programming development. The system includes three parts: Detection module,training module and recognition module.

Contents

List of Figures

ABBREVIATIONS

LBP	Local Binary Patterns
LBPH	Local Binary Pattern Histograms
SRS	Software Requirement Specifications

Chapter 1

INTRODUCTION

Face recognition is the task of identifying an already detected object as a known or unknown face. Often the problem of face recognition is confused with the problem of face detection. Face Recognition on the other hand is to decide if the "face" is someone known, or unknown, using for this purpose a database of faces in order to validate this input face.

1.1 Face Detection

Face detection involves separating image windows into two classes; one containing faces (training the background (clutter)). It is difficult because although commonalities exist between faces, they can vary considerably in terms of age, skin colour and facial expression. The problem is further complicated by differing lighting conditions, image qualities and geometries, as well as the possibility of partial occlusion and disguise. An ideal face detector would therefore be able to detect the presence of any face under any set of lighting conditions, upon any background.

There are two pre-trained classifiers of OpenCV for Face Detection :
1. Haar Cascade Classifier
2. LBP Classifier

1.2 Face Recognition

When you look at an apple, your mind immediately tells you: that is an apple. This process is recognition in the simplest of terms. So, what's facial recognition? The same, but for faces, obviously.

But, the real question is: How can a computer recognize a face? Take a real life example: When you meet someone for the first time, you don't know who that person is at once, right? While he's talking to you or shaking your hand, you're looking at his face: eyes, nose, mouth, skin tone. This process is your mind gathering data and training for face recognition.

Next, that person tells you that his name is ABC. So, our brain has already gotten the face data, and now it has learned that this data belongs to ABC.

The next time we see ABC or see a picture of his face, our mind will follow this exact process:

1. Face Detection: Look at the picture and find a face in it.
2. Data Gathering: Extract unique characteristics of ABC's face such that it can use to differentiate him from another person, like eyes, mouth, nose, etc.
3. Data Comparison: Despite variations in light or expression, it will compare those unique features to all the features of all the people we know.
4. Face Recognition: It will determine that the face belongs to ABC.

Computers aren't able, yet, to do this automatically, so we need to teach them how to do it step-by-step. These steps are:

A. Data Gathering
B. Train the Recognizer
C. Recognition

Chapter 2

LITERATURE SURVEY

Face detection is a computer technology that determines the location and size of human face in arbitrary (digital) image. The facial features are detected and any other objects like trees, buildings and bodies etc are ignored from the digital image. It can be regarded as a specific‘ case of object-class detection, where the task is finding the location and sizes of all objects in an image that belong to a given class. Face detection, can be regarded as a more general‘ case of face localization. In face localization, the task is to find the locations and sizes of a known number of faces. We are using two pre-trained classifier to do Face Detection, that is, Haar Classifier and LBP Classifier.

2.1 Related Work in Pose Variation

Focusing on the aspect of pose invariance, face recognition approaches may be divided into two categories:
1. Global Approach and
2. Component-based approach

In **global approach**, a single feature vector that represents the whole face image is used as input to a classifier. Several classifiers have been proposed in the literature e.g. minimum distance classification in the eigen space , Fishers discriminant analysis , and neural networks. Global techniques work well for classifying frontal views of faces. However, they are not robust against pose changes since global features are highly sensitive to translation and rotation of the face.

To avoid this problem, an alignment stage can be added before classifying the face. Aligning an input face image with a reference face image requires computing correspondence between the two face images. The correspondence is usually determined for a small number of prominent points in the face like the center of the eye, the nostrils, or the corners of the mouth.

Based on these correspondences, the input face image can be warped to a reference face image. An affine transformation is computed to perform the warping. Active shape models are used in to align input faces with model faces. A semi-automatic alignment step in combination with support vector machines classification was proposed. An alternative to the global approach is to classify local facial components.

The main idea of **component based recognition** is to compensate for pose changes by allowing a flexible geometrical relation between the components in the classification stage. Face recognition was performed by independently matching templates of three facial regions (eyes, nose and mouth). The configuration of the components during classification was unconstrained since the system did not include a geometrical model of the face.
A similar approach with an additional alignment stage was proposed. In a geometrical model of a face was implemented by a 2D elastic graph. The recognition was based on wavelet coefficients that were computed on the nodes of the elastic graph.

2.2 Related work in expression variation

Recent advances in image analysis and pattern recognition open up the possibility of automatic detection and classification of emotional and conversational facial signals.
Automating facial expression analysis could bring facial expressions into man-machine interaction as a new modality and make the interaction tighter and more efficient. Such a system could also make classification of facial expressions widely accessible as a tool for research in behavioral science and medicine.

Pantic surveyed the automatic facial expression analysis, and Daugman pointed out several critical issues involved in an effective face recognition system, while the most recent and comprehensive survey is possibly from that of Zhao et al, where many of the latest techniques are reviewed. One of the fundamental issues about the facial expression analysis is the representation of the visual information that an examined face might reveal. The results of Johansson's point-light display experiments gave a clue to this problem.

The experiments suggest that the visual properties of the face, regarding the information about the shown facial expression, could be made clear by describing the movements of points belonging to the facial features (eyebrows, eyes, and mouth) and then by analyzing the relationships between those movements. This triggered the researchers of vision-based facial gesture analysis to make different attempts to define point-based visual properties of facial expressions.

In another approach to face representation (holistic approach), the face is represented as a whole unit. The face can be also modeled using a socalled hybrid approach, which typifies a combination of analytic and holistic approaches to face representation. In this approach, a set of facial points is usually used to determine an initial position of a template that models the face.

Irrespectively of the kind of the face model applied, attempts must be made to model and then extract the information about the displayed facial expression without losing any (or much) of that information. Several factors make this task complex. The first is the presence of facial hair, glasses, etc., which obscure the facial features. Another problem is the variation in size and orientation of the face in input images. This disables a search for fixed patterns in the images.

Chapter 3

FACE DETECTION

3.1 Theory of OpenCV Face Detection Classifiers

A computer program that decides whether an image is a positive image (face image) or negative image (non-face image) is called a classifier. A classifier is trained on hundreds of thousands of face and non-face images to learn how to classify a new image correctly.

OpenCV provides us with two pre-trained and ready to be used for face detection classifiers:

1. Haar Classifier
2. LBP Classifier

3.1.1 HAAR Classifier

The Haar Classifier is a machine learning based approach, an algorithm created by Paul Viola and Michael Jones, which are trained from many many positive images (with faces) and negatives images (without faces).It starts by extracting Haar features from each image as shown by the windows in Figure 3.1:

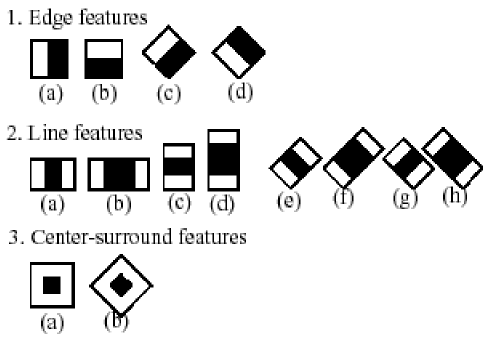


Figure 3.1: HAAR FEATURES

.

Each window is placed on the picture to calculate a single feature. This feature is a single value obtained by subtracting the sum of pixels under the white part of the window from the sum of the pixels under the black part of the window.

Now, all possible sizes of each window are placed on all possible locations of each image to calculate plenty of features. This is shown in Figure 3.2

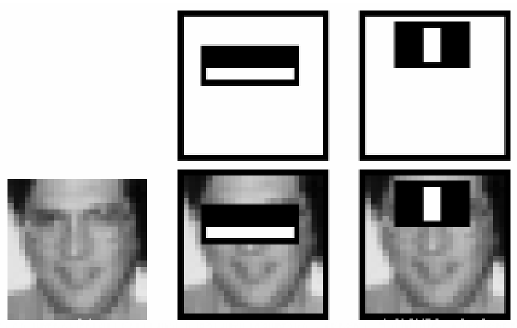


Figure 3.2: Different Stages in Visualisation

For example, in above image, we are extracting two features. The first one focuses on the property that the region of the eyes is often darker than the area of the nose and cheeks. The second feature relies on the property that the eyes are darker than the bridge of the nose.

But among all these features calculated, most of them are irrelevant. For example, when used on the cheek, the windows become irrelevant because none of these areas are darker or lighter than other regions on the cheeks, all sectors here are the same. So we promptly discard irrelevant features and keep only those relevant with a fancy technique called Adaboost. AdaBoost is a training process for face detection, which selects only those features known to improve the classification (face/non-face) accuracy of our classifier.

In the end, the algorithm considers the fact that generally most of the region in an image is a non-face region. Considering this, it's a better idea to have a simple method to check if a window is a non-face region, and if it's not, discard it right away and don't process it again. So we can focus mostly on the area where a face is.

3.1.2 LBP Cascade Classifier

As any other classifier, the Local Binary Patterns, or LBP in short, also needs to be trained on hundreds of images. LBP is a visual/texture descriptor, and thankfully, our faces are also composed of micro visual patterns.

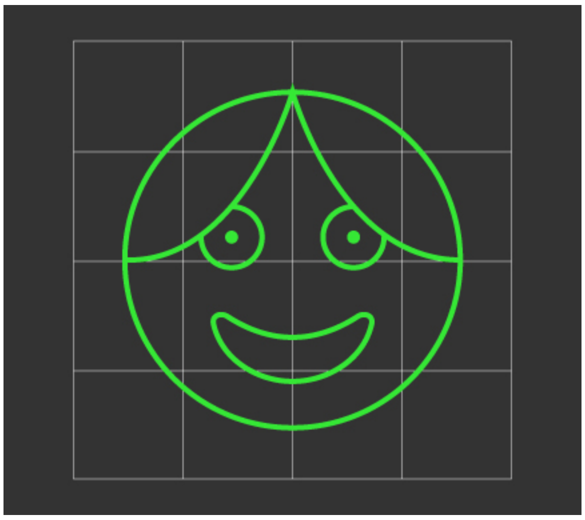


Figure 3.3: LBP Windows

For each block, LBP looks at 9 pixels (3x3 window) at a time, and with a particular interest in the pixel located in the center of the window.

Then, it compares the central pixel value with every neighbor's pixel value under the 3x3 window. For each neighbor pixel that is greater than or equal to the center pixel, it sets its value to 1, and for the others, it sets them to 0.

After that, it reads the updated pixel values (which can be either 0 or 1) in a clockwise order and forms a binary number. Next, it converts the binary number into a decimal number, and that decimal number is the new value of the center pixel. We do this for every pixel in a block.

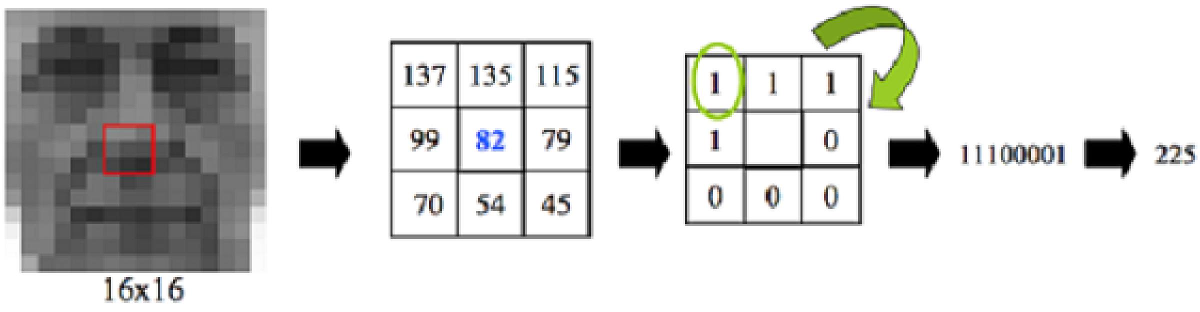


Figure 3.4: LBP Conversion to Binary

Then, it converts each block values into a histogram, so now we have gotten one histogram for each block in an image, like shown in Figure 3.5

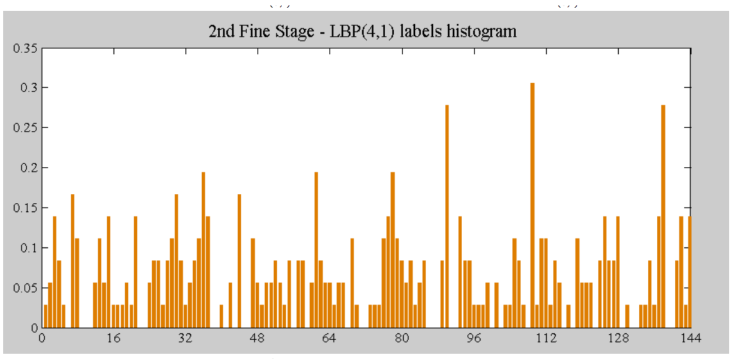


Figure 3.5: LBP Histogram

Finally, it concatenates these block histograms to form a one feature vector for one image, which contains all the features we are interested. So, this is how we extract LBP features from a picture.

Chapter 4

FACE RECOGNITION

4.1 Theory of OpenCV face Recognizers

In OpenCV, there are three easy steps to computer coding facial recognition, which are similar to the steps that our brains use for recognizing faces. These steps are:

Data Gathering: Gather face data (face images in this case) of the persons you want to identify.
Train the Recognizer: Feed that face data and respective names of each face to the recognizer so that it can learn.
Recognition: Feed new faces of that people and see if the face recognizer you just trained recognizes them.

OpenCV has three built-in face recognizers and we can use any of them just by changing a single line of code. These are:

A. EigenFaces: cv2.face.createEigenFaceRecognizer()
B. FisherFaces: cv2.face.createFisherFaceRecognizer()
C. Local Binary Patterns Histograms (LBPH): cv2.face.createLBPHFaceRecognizer()

4.1.1 Eigenfaces Face Recognizer

This algorithm considers the fact that **not all parts of a face are equally important or useful for face recognition**. Indeed, when we look at someone, we recognize that person by his distinct features, like the eyes, nose, cheeks or forehead and how they vary respect to each other.

In that sense, we are focusing on the areas of maximum change. For example, from the eyes to the nose there is a significant change, and same applies from the nose to the mouth. When you look at multiple faces, you compare them by looking at these areas, because by catching the maximum variation among faces, they help you differentiate one face from the other.

In this way, is how Eigenfaces face recognizer works. It looks at all the training images of all the people as a whole and tries to extract the components which are relevant and useful and discards the rest. These important features are called **principal components.**

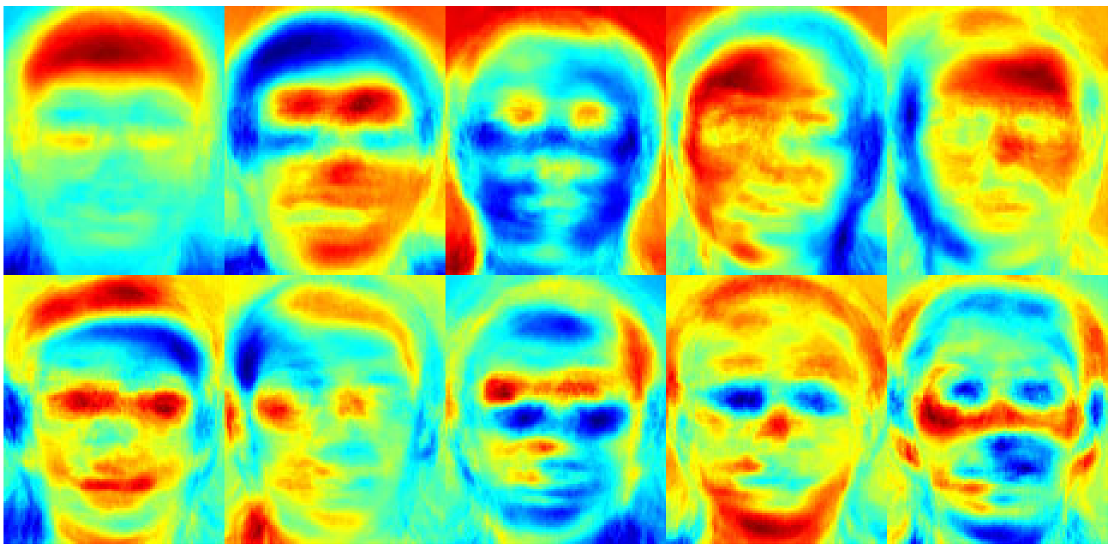


Figure 4.1: Eigenfaces Face Recogniser Principal Component

We can see that the useful features represent faces which receive the name of Eigen Face.

EigenFaces Face Recognizer trains itself by extracting principal components, but it also keeps a record of which ones belong to which person. Thus, whenever we introduce a new image to the algorithm, it repeats the same process as follows:

1. Extract the principal components from the new picture.
2. Compare those features with the list of elements stored during training.
3. Find the ones with the best match.
4. Return the ‘person’ label associated with that best match component.

However, one thing to note in above image is that EigenFaces algorithm also considers illumination as an important feature. In consequence, lights and shadows are picked up by EigenFaces, which classifies them as representing a ‘face.’

4.1.2 Fisherfaces Face Recognizer

This algorithm is an improved version of the last one. As we just saw, EigenFaces looks at all the training faces of all the people at once and finds principal components from all of them combined. By doing that, it doesn't focus on the features that discriminate one individual from another. Instead, it concentrates on the ones that represent all the faces of all the people in the training data, as a whole.

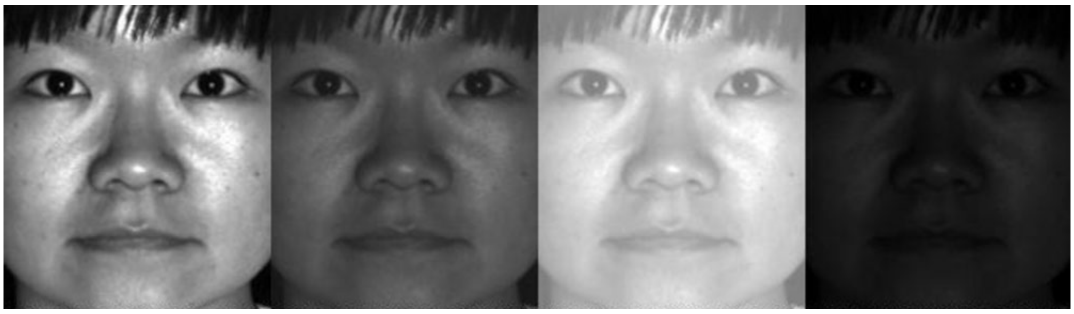


Figure 4.2: Consider the lighting changes in the above Image

Since EigenFaces also finds illumination as a useful component, it will find this variation very relevant for face recognition and may discard the features of the other people's faces, considering them less useful. In the end, the variance that EigenFaces has extracted represents just one individual's facial features.

We can fix this issue by tunning EigenFaces so that it extracts useful features from the faces of each person separately instead of extracting them from all the faces combined. In this way, even if one person has high illumination changes, it will not affect the other people's features extraction process.

Precisely, FisherFaces face recognizer algorithm extracts principal components that differentiate one person from the others. In that sense, an individual's components do not dominate over the others.

We can see that the features represent faces which receive the name of Fisher Faces. Also, FisherFaces only prevents features of one person from becoming dominant, but it still considers illumination changes as a useful feature. We know that light variation is not a useful feature to extract as it is not part of the actual face. It is shown in Figure 4.3.

4.1.3 Local Binary Patterns Histograms Face Recognizer(LBPH)

We know that Eigenfaces and Fisherfaces are both affected by light and, in real life, we can't guarantee perfect light conditions. LBPH face recognizer is an improvement to overcome this drawback.

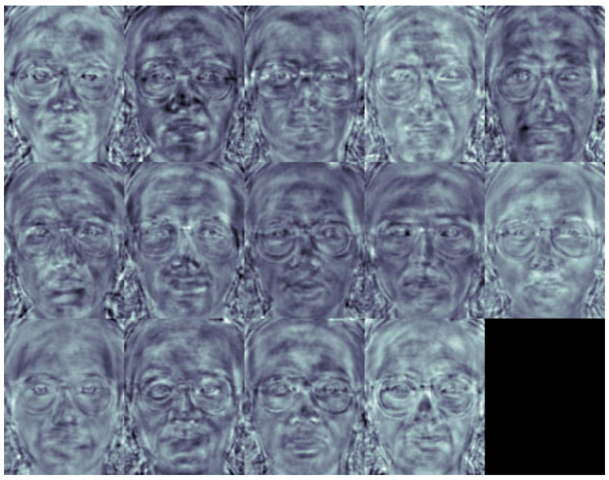


Figure 4.3: FisherFaces Face Recognizer Principal Component

The idea with **LBPH** is not to look at the image as a whole, but instead, try to find its local structure by comparing each pixel to the neighboring pixels.

In LBPH Face Recognizer Process, convert the image into a binary pattern by taking a 3x3 window in a clockwise order and when we finish doing this on the whole image , we will have list of **Local Binary Patterns.** Detailed Explaination about LBPH is described in previous chapter **Section 3.1.2**.

Now, after we get a list of local binary patterns, we convert each one into a decimal number using binary to decimal conversion as shown in **Figure 3.4** and then we make a histogram of all of those decimal values. A sample histogram looks like this:

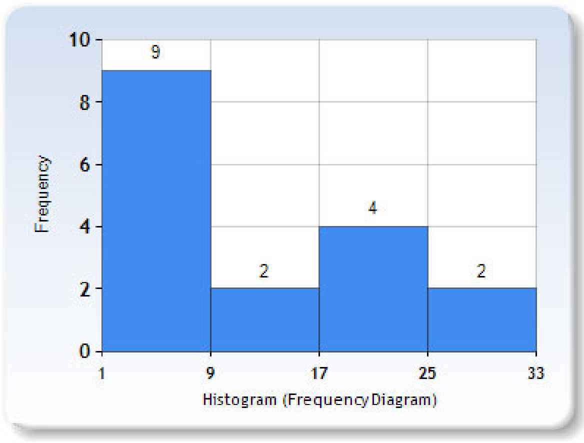


Figure 4.4: Histogram Sample

In the end, we will have one histogram for each face in the training data set. That means that if there were 100 images in the training data set then LBPH will extract 100 histograms after training and store them for later recognition. The algorithm also keeps track of which histogram belongs to which person.

During recognition, the process is as follows:

1. Feed a new image to the recognizer for face recognition.
2. The recognizer generates a histogram for that new picture.
3. It then compares that histogram with the histograms it already has.
4. Finally, it finds the best match and returns the person label associated with that best match.

Below is a group of faces and their respective local binary patterns images. You can see that the LBP faces are not affected by changes in light conditions:

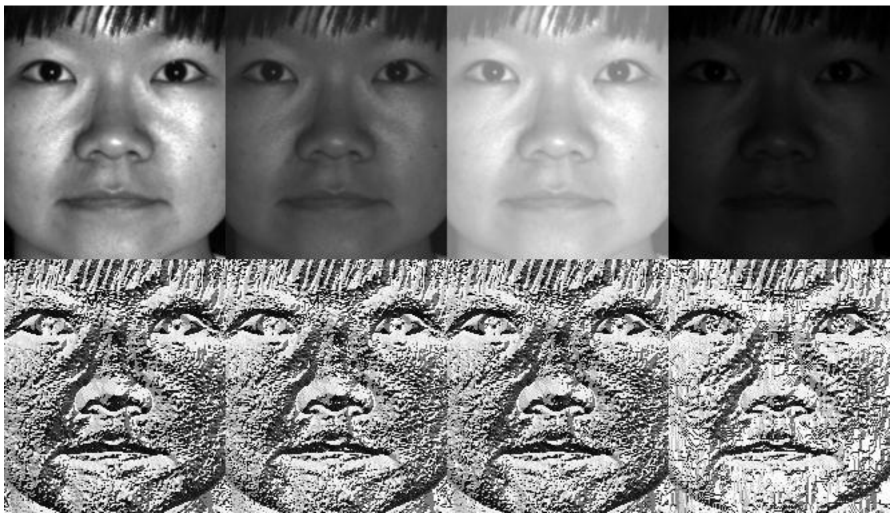


Figure 4.5: LBPH Face Recognizer Principal Components

Chapter 5

SYSTEM STUDY

5.1 Software Requirements Specifications

A software requirements specification (SRS) is a document that captures complete description about how the system is expected to perform.

5.1.1 Purpose

The purpose of this document is to capture, in natural language and at a functional level, the description and requirements of a Face Detection and Recognition using OpenCV. The focus here is to recognize a person using webcam or video source. This document gives a brief idea of how computer can detect and recognize human face.

5.1.2 Scope

Face detection is a computer technology being used in a variety of applications that identifies human faces in digital images. It can be used for simple surveillance applications such as ATM user security, while the implemented manual face detection and automated recognition system is ideal of mugshot matching.

5.2 Hardware and Software Requirements

5.2.1 Hardware Requirements

1. Processor - Intel Pentium V and higher
2. RAM - 2GB and Higher
3. Webcam

5.2.2 Software Requirements

1. Front End - OpenCV, Numpy
2. Back End - Python
3. Operating System - Windows, Ubuntu, MAC

5.3 OpenCV

5.3.1 About

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

It has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDAand OpenCL interfaces are being actively developed right now. There are over 500 algorithms and about 10 times as many functions that compose or support those algorithms. OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers.

5.3.2 Features of OpenCV

Various features can be achieved using OpenCV Library such as:

1. Read and write images
2. Capture and save videos
3. Process images (filter, transform)
4. Perform feature detection
5. Detect specific objects such as faces, eyes, cars, in the videos or images.
6. Analyze the video, i.e., estimate the motion in it, subtract the background, and track objects in it.

7. Core Operations on image such as pixel editing, geometric transformations, code optimization, some mathematical tools etc.
8. Other Operations such as Camera Calibration and 3D Reconstruction, Machine Learning, Computational Photography, OpenCV Python Bindings etc.

5.4 OpenCV using Python

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. Python is a general purpose programming language started by **Guido van Rossum** that became very popular very quickly, mainly because of its simplicity and code readability. It enables the programmer to express ideas in fewer lines of code without reducing readability.

Compared to languages like C/C++, Python is slower. That said, Python can be easily extended with C/C++, which allows us to write computationally intensive code in C/C++ and create Python wrappers that can be used as Python modules. This gives us two advantages: **First,** the code is as fast as the original C/C++ code and **Second,** it easier to code in Python than C/C++. OpenCV-Python is a Python wrapper for the original OpenCV C++ implementation.

OpenCV-Python makes use of Numpy, which is a highly optimized library for numerical operations with a MATLAB-style syntax. All the OpenCV array structures are converted to and from Numpy arrays. This also makes it easier to integrate with other libraries that use Numpy such as SciPy and Matplotlib.

Chapter 6

IMPLEMENTATION

Before we start the actual coding, we need to install the **Code Dependencies** and import the Required Modules.

Code Dependencies

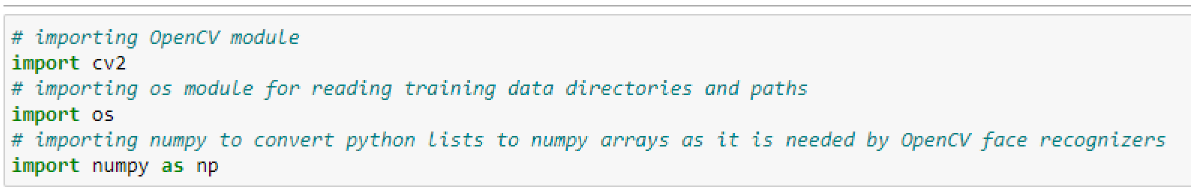
Install the following Dependencies:

1. OpenCV 4.1.0
2. Python v3.7
3. NumPy that makes computing in Python easy. It contains a powerful implementation of N-dimensional arrays which we will use for feeding data as input to OpenCV functions.

Required Modules

Import the following modules:

1.**cv2:** This is the OpenCV module for Python used for face detection and face recognition.
2.**os:** We will use this Python module to read our training directories and file names.
3.**numpy:** This module converts Python lists to numpy arrays as OpenCV face recognizer needs them for the face recognition process.

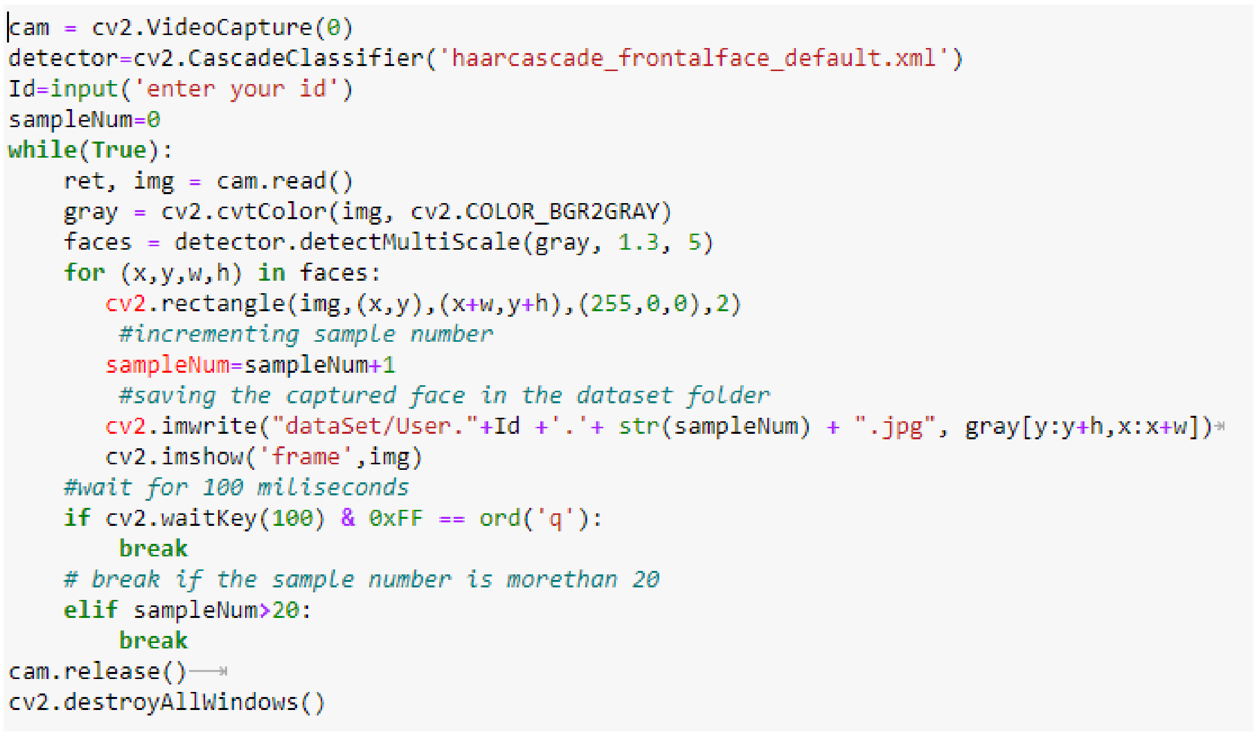


```python
# importing OpenCV module
import cv2
# importing os module for reading training data directories and paths
import os
# importing numpy to convert python lists to numpy arrays as it is needed by OpenCV face recognizers
import numpy as np
```

6.1 Prepare Training Data

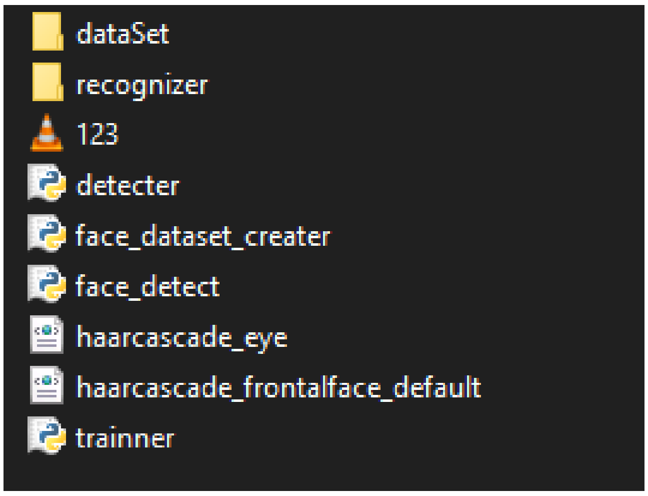
The more Images used in Training, the better will be the result. Being thorough with this principle is important because it is the only way for training a face recognizer so it can learn the different ‘faces’ of the same person; for example: with glasses, without glasses, laughing, sad, happy, crying, with a beard, without a beard, etc.

We are creating our DataSet using webcam or any video source from the storage. The Webcam is turned on for a duration of 100 msec in which our algorithm will create 20 sample images of each individual and it is saved in a directory.



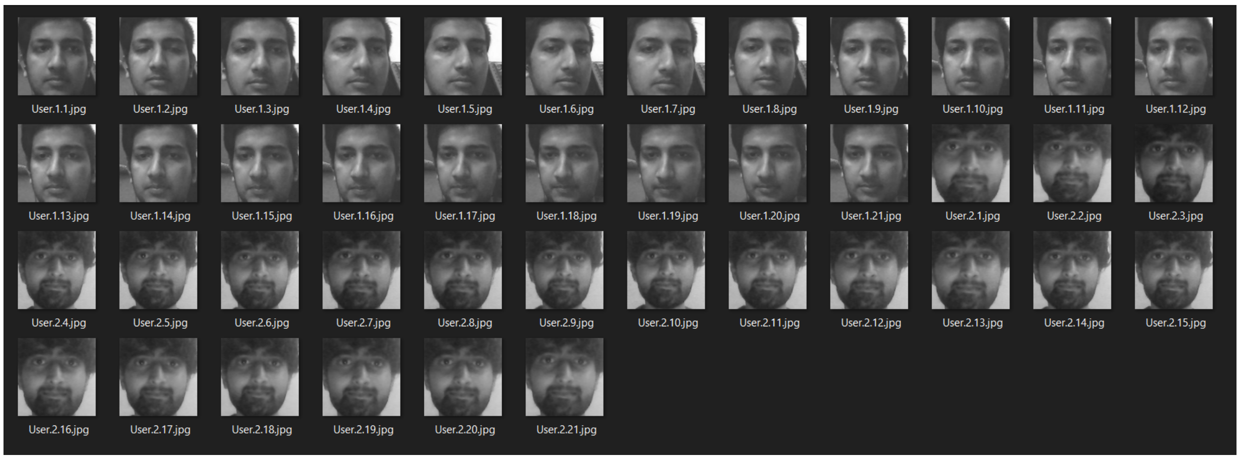
```
cam = cv2.VideoCapture(0)
detector=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
Id=input('enter your id')
sampleNum=0
while(True):
    ret, img = cam.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = detector.detectMultiScale(gray, 1.3, 5)
    for (x,y,w,h) in faces:
       cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,0),2)
        #incrementing sample number
       sampleNum=sampleNum+1
        #saving the captured face in the dataset folder
       cv2.imwrite("dataSet/User."+Id +'.'+ str(sampleNum) + ".jpg", gray[y:y+h,x:x+w])
       cv2.imshow('frame',img)
    #wait for 100 miliseconds
    if cv2.waitKey(100) & 0xFF == ord('q'):
        break
    # break if the sample number is morethan 20
    elif sampleNum>20:
        break
cam.release()
cv2.destroyAllWindows()
```

The Directory folder contains one sub-folder named dataSet which contains all the images of the individuals and another sub-folder named recognizer which contains XML files which stores LBPH value for each image. And the directory also contains HAAR Cascade Files and Python Executable files.



dataSet
recognizer
123
detecter
face_dataset_creater
face_detect
haarcascade_eye
haarcascade_frontalface_default
trainner

The Dataset contains images that we will use to test our face recognition program after we have trained it successfully as shown below:

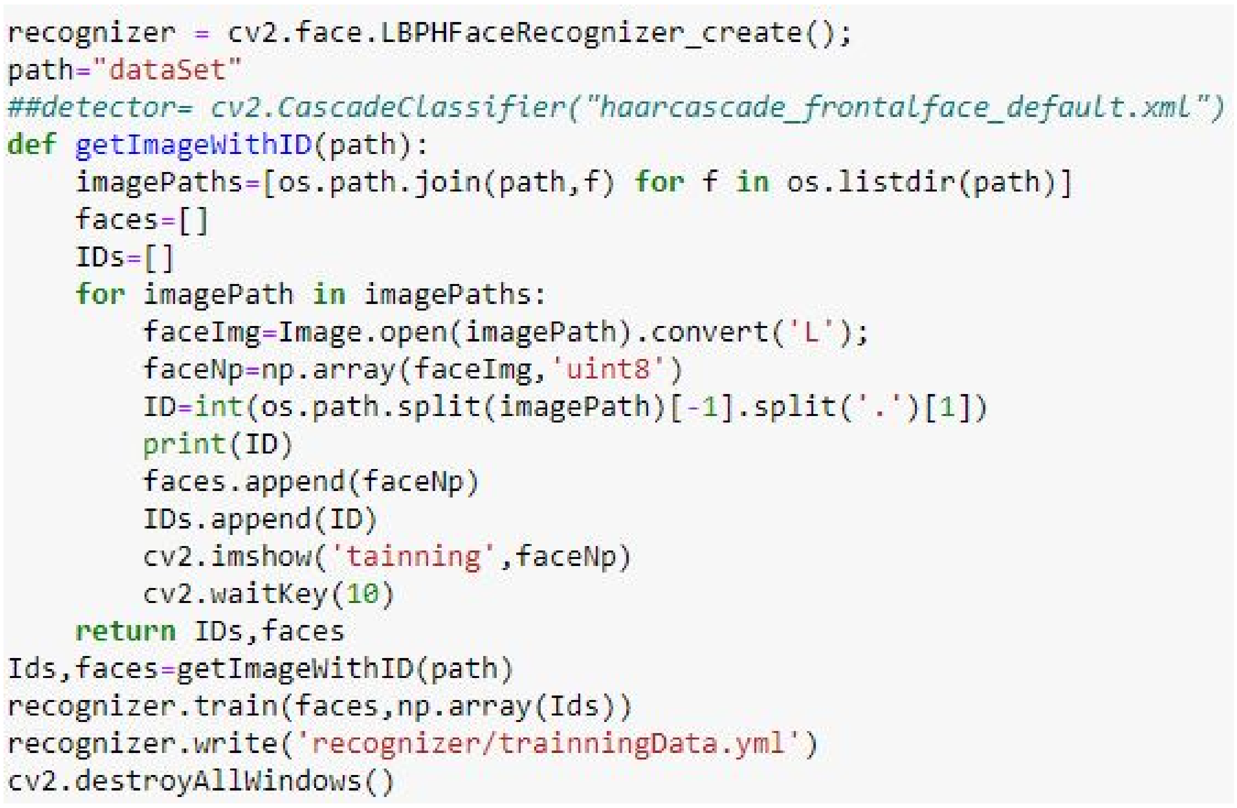


6.2 Train Face Recognizer

As we mentioned earlier, OpenCV comes equipped with three face recognizers:

1. EigenFaces: cv2.face.createEigenFaceRecognizer()
2. FisherFaces: cv2.face.createFisherFaceRecognizer()
3. Local Binary Patterns Histogram (LBPH): cv2.face.LBPHFisherFaceRecognizer()

We are going to use LBPH recognizer here due to it's advantage over Eigenfaces and Fichesrfaces Face Recognizers.

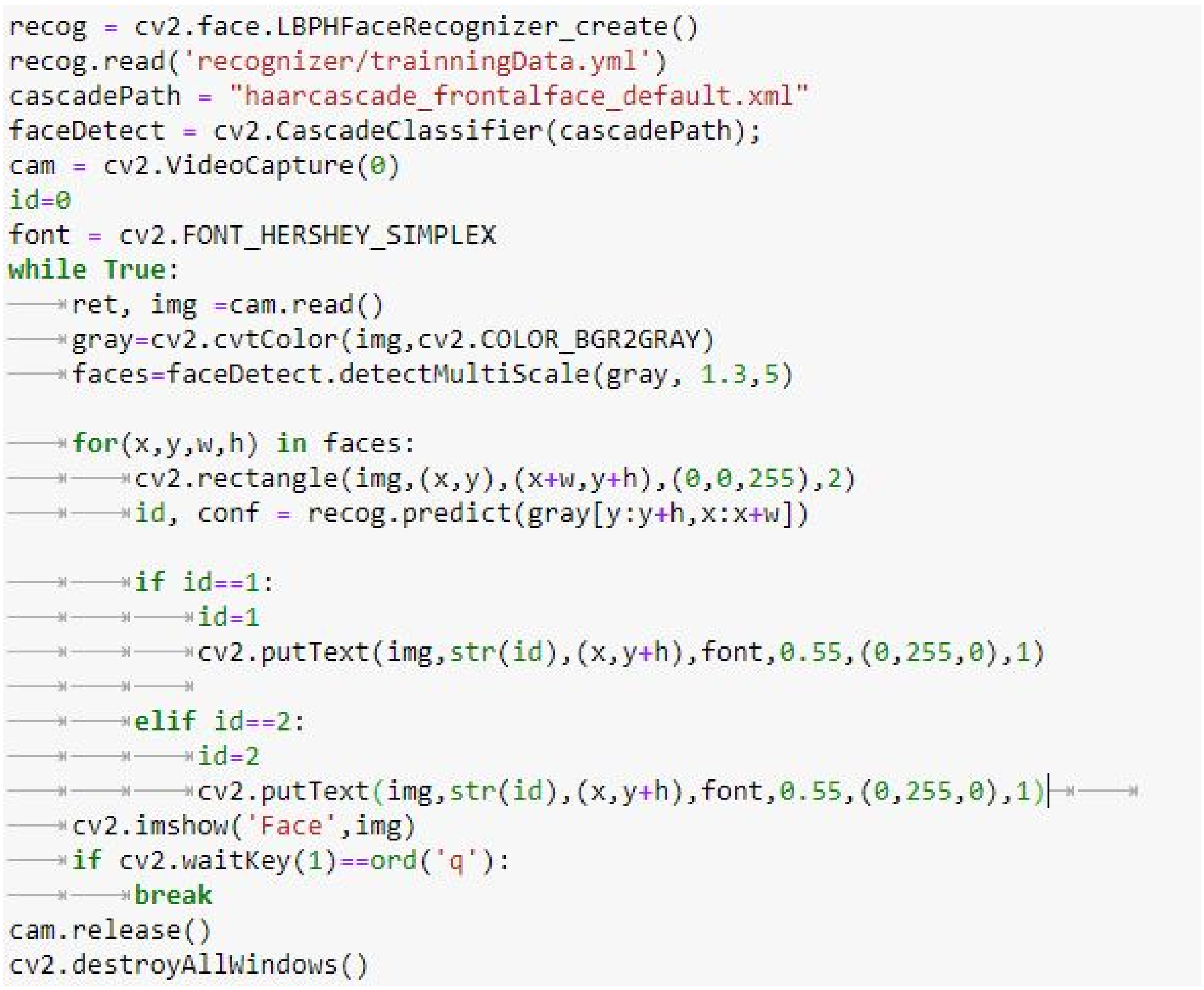


```
recognizer = cv2.face.LBPHFaceRecognizer_create();
path="dataSet"
##detector= cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
def getImageWithID(path):
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    faces=[]
    IDs=[]
    for imagePath in imagePaths:
        faceImg=Image.open(imagePath).convert('L');
        faceNp=np.array(faceImg,'uint8')
        ID=int(os.path.split(imagePath)[-1].split('.')[1])
        print(ID)
        faces.append(faceNp)
        IDs.append(ID)
        cv2.imshow('tainning',faceNp)
        cv2.waitKey(10)
    return IDs,faces
Ids,faces=getImageWithID(path)
recognizer.train(faces,np.array(Ids))
recognizer.write('recognizer/trainningData.yml')
cv2.destroyAllWindows()
```

The above code creates a yml file which stores the local binary values obtained from all the images from the dataset after applying LBPH Recognizer Algorithm.

6.3 Prediction

This is where we get to see if our algorithm is detecting and recognizing individual faces or not.

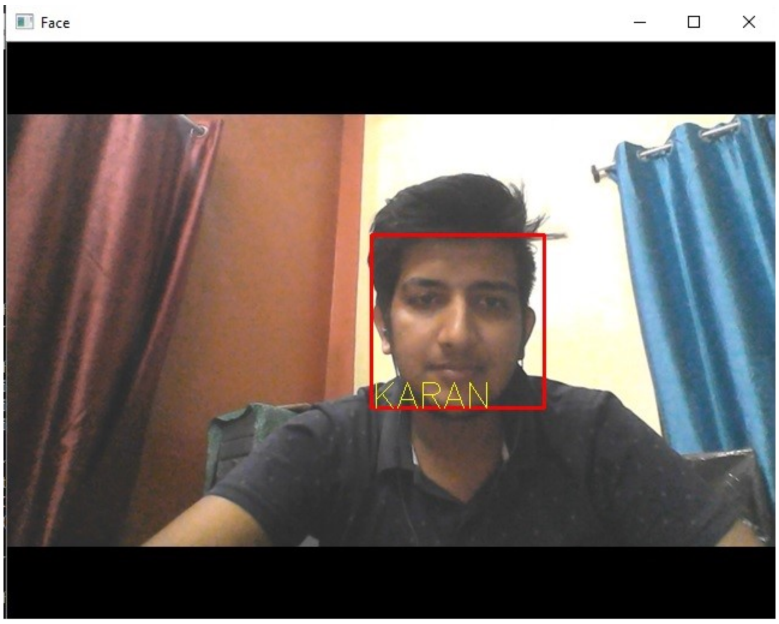


```
recog = cv2.face.LBPHFaceRecognizer_create()
recog.read('recognizer/trainningData.yml')
cascadePath = "haarcascade_frontalface_default.xml"
faceDetect = cv2.CascadeClassifier(cascadePath);
cam = cv2.VideoCapture(0)
id=0
font = cv2.FONT_HERSHEY_SIMPLEX
while True:
    ret, img =cam.read()
    gray=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    faces=faceDetect.detectMultiScale(gray, 1.3,5)

    for(x,y,w,h) in faces:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),2)
        id, conf = recog.predict(gray[y:y+h,x:x+w])

        if id==1:
            id=1
            cv2.putText(img,str(id),(x,y+h),font,0.55,(0,255,0),1)

        elif id==2:
            id=2
            cv2.putText(img,str(id),(x,y+h),font,0.55,(0,255,0),1)
    cv2.imshow('Face',img)
    if cv2.waitKey(1)==ord('q'):
        break
cam.release()
cv2.destroyAllWindows()
```

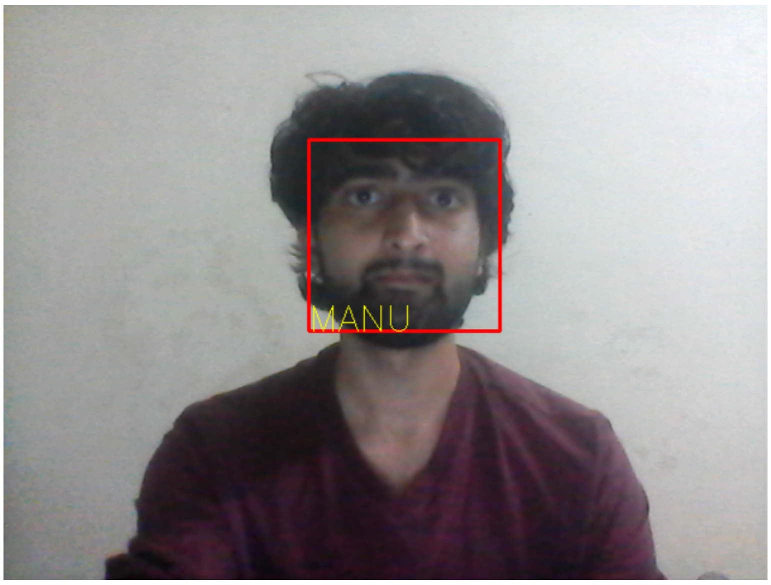
In the above code, first webcam is turned on and then using the HAAR Cascade Classifier face is detected and then pass those faces to our trained face recognizer. By using some utility functions we will draw bounding box (rectangle) around the face and putting the person's name near the face bounding box if the match is found and if not then write "NOT FOUND" near the bounding box.

6.4 Output

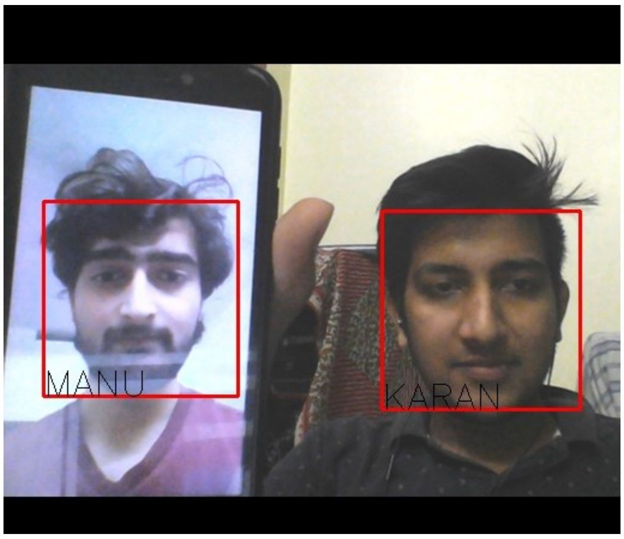
Output 1



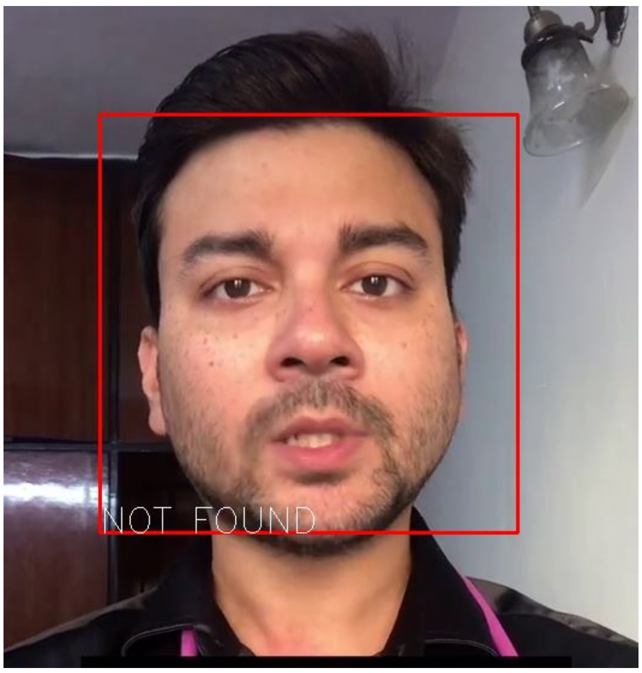
Output 2



Output 3



Output 4



Chapter 7

CONCLUSION

To improve the recognition performance, there are many things that can be improved here, some of them being fairly easy to implement. For example, we could add color processing, edge detection, etc. We can usually improve the face recognition accuracy by using more input images at least 50 per person by taking more photos of each person, particularly from different angles and lighting conditions. If we can't take more photos there are several simple techniques we could use to obtain more training images, by generating new images from our existing ones. We could create mirror copies of our facial images, so that we will have twice as many training images and it won't have a bias towards left or right. We could translate or resize or rotate our facial images slightly to produce many alternative images for training, so that it will be less sensitive to exact conditions.

It is important to have a lot of variation of conditions for each person, so that the classifier will be able to recognize the person in different lighting conditions and positions, instead of looking for specific conditions. But it's also important to make sure that a set of images for a person is not too varied.

Histogram Equalization is a very basic image pre-processing method that can make things worse in some situations, so we will probably have to combine several different methods until we get decent results. And something important to understand is that at the heart of the algorithm, it is matching images by basically doing the equivalent of subtracting the testing image with a training image to see how similar they are. This would work fairly well if a human performed it, but the computer just thinks in terms of pixels and numbers. So if we imagine that it is looking at one pixel in the test image, and subtracting the gray scale value of that pixel with the value of the pixel in the exact same location of each training image, and the lower the difference then the better the match.

So, if we just move an image by a few pixels across, or use an image that is just a few pixels bigger or has a few more pixels of the forehead showing than the other image etc, then it will think they are completely different images. This is also true if the background is different, because the code doesn't know the difference between background and foreground (face), which is why its important to crop away as much of the background as we can, such as by only using a small section inside the face that doesn't include any background at all.

Since the images should be almost perfectly aligned, it actually means Jan Erik Solemthat in many cases, using small low-resolution images (such as by shrinking the images to thumbnail size) can give better recognition results than large high-resolution images. Also, even if the images are perfectly aligned, if the testing image is a bit brighter than the training image then it will still think there is not much of a match.

The implemented fully automated face detection and recognition system (with an eye detection system) could be used for simple surveillance applications such as ATM user security, while the implemented manual face detection and automated recognition system is ideal of mugshot matching. Since controlled conditions are present when mugshots are gathered, the frontal view face recognition scheme should display a recognition accuracy far better than the results, which were obtained in this study, which was conducted under adverse conditions.

The automated vision systems implemented in this thesis did not even approach the performance, nor were they as robust as a human's innate face recognition system. However, they give an insight into what the future may hold in computer vision.

REFERENCES

BOOKS:

[**1**] **Luis Pedro Coelho and Willi Richert**, Building Machine Learning Systems with Python, Packt Publishing Ltd., Second Edition, March 2015, Chapter-10, Computer Vision, Page No-219
[**2**] **Mark Lutz**, Learning Python, Oreilly, Fifth Edition, June 2013

WEBLINKS

[**1**] https://www.superdatascience.com/blogs/opencv-face-recognition
[**2**] https://www.superdatascience.com/blogs/opencv-face-detection
[**3**] https://docs.opencv.org/4.3.0/d6/d00/tutorial_py_root.html