Bachelor Thesis Project

Face Detection and Recognition using KNN and LBPH Algorithm

Submitted By:

Akshay Pilania(213CO14)

Ayush Pal(237CO14)

Hitesh Kumar(260CO14)

Under the guidance of

Mr. Bijendra Kumar Singh

(Professor and Head, Department of Computer Engineering)

DISSERTATION SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE DEGREE OF BACHELOR OF ENGINEERING

Department of

Computer Engineering

Netaji Subhas Institute of Technology

University of Delhi

May 2018

CERTIFICATE

This is to certify that the project entitled "Face Detection and Recognition using KNN and LBPH Algorithm" is a bona-fide work carried out by Akshay Pilania(213CO14), Ayush Pal(237CO14) and Hitesh Kumar(260CO14), students of 8th semester, B.E. (Computer Engineering), Netaji Subhas Institute of Technology.

This project work has been prepared as a partial fulfilment of the requirements for the award of the degree of Bachelor of Engineering in Computer Engineering, University of Delhi, in the academic year 2017-2018. This work has not been presented earlier for any other academic activity.

Date:

Bijendra Kumar Singh

(Professor and Head of Department)

Department of Computer Engineering

Netaji Subhas Institute of Technology

ACKNOWLEDGEMENT

We take this opportunity to express our heartfelt gratitude to our respected project guide, Mr. Bijendra Kumar Singh, Professor and Head, Department of Computer Engineering, NSIT who kindly consented to be our guide for this project.

We would like to thank him for his invaluable guidance, time and efforts in guiding us throughout this project, and for encouraging us to constantly challenge ourselves and come up with innovative ideas.

We would like to thank all the faculty members and staff at the Computer Engineering Department of NSIT, for their support in terms of guidance and resources.

Akshay Pilania	Ayush Pal	Hitesh Kumar
(212CO14)	(237CO14)	(260CO14)

ABSTRACT

Computer Vision is a field that includes methods for acquiring, processing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information, e,g., in the forms of decisions.

Face Detection is a computer technology that identifies human faces in digital images. Face Recognition is to classify those detected faces into various set of pre-stored classes. This technology is being used in a variety of applications nowadays such as Biometrics, Photography, Marketing, Security etc.

Face Detection and Recognition involves the following 3 steps:

1. Detecting face/s in a frame.
2. Processing on the detected faces to check their similarity with the faces stored in datasets.
3. Putting the faces into various classes using the information gained in second step.

In this paper, we propose an approach to detect and recognise faces in a frame using K Nearest Neighbours(KNN) Algorithm and Local Binary Patterns Histogram(LBPH) Algorithm. We have used various OpenCV libraries for Python to detect faces in the frame. We have compared both of the algorithms on the basis of their accuracy and concluded our report.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 4:

Chapter 1: Introduction

Data mining is widely used in various areas such as finance, marketing, communication, web service, surveillance and security. The continuing growth in computing hardware and consumer demand has led to a rapid increase of multimedia data searching. With the rapid development of computer vision, face recognition methods are becoming increasingly prevalent in replacing other older methods and technologies in various areas.

Apple Inc. recently used Face Recognition in iPhone X, and replaced their older method of Thumb Impression Recognition to authenticate the phones and apps. According to them, Face ID is far more reliable and secure than Touch ID. An Apple support document on Face ID reads in part: "The probability that a random person in the population could look at your iPhone X and unlock it using Face ID is approximately 1 in 1,000,000 (versus 1 in 50,000 for Touch ID)."

1.1 Objective:

This research project aims at:

- To study the various machine learning algorithms that can be used to recognise faces in a frame.
- To detect faces in a frame.
- To be able to feed into the database various faces in a representation format which will be later used as datasets to recognise and classify a face.
- To implement KNN and LBPH algorithms to recognise faces in a frame.
- To connect the software to a database and show the details of the person recognised.
- To compare both of the algorithms on the basis of Accuracy and Complexity.

1.2 Motivation:

The motivation for this project stems from lack of the availability of actuate face recognition software which can be used in daily life. We were also motivated to do this project due to our growing interest in machine learning and its algorithms.

1.3 Organisation of chapters:

In the upcoming chapters we will discuss this project in more detail providing more insight into the literature survey done and project methodology employed by us.

In chapter 2, the concepts and terminology related to the field of Face Detection and Recognition are discussed at length.

In chapter 3, we have described the proposed approach that has been adopted in our research. This includes describing the overview of our approach, and tools and datasets used for implementation.

Chapter 4 talks about the analysis of the patterns observed and the results finally obtained.

We finally conclude this report by presenting the conclusions and future work.

Chapter 2: Literature Survey

2.1 Face Detection:

It is a necessary first step in all of the face processing systems and its performance can severely influence on the overall performance of recognition. Three main approaches are proposed for face detection: **feature based, image based, and template matching.**

2.1.1 Feature based:

These approaches attempt to utilize some priori knowledge of human face characteristics and detect those representative features as edges, texture, colour or motion. Edge features have been applied in face detection from the beginning (Colmenarez & Huang 1996), and several variation have been developed (Fan, Yau, Elmagarmid & Aref 2001; Froba & Kublbeck 2002; Suzuki & Shibata 2004). Edge detection is a necessary first step for edge representation. Two edge operators that are commonly used are the Sobel Operator and Marr-Hildreth operator. Edge features can be easily detected with a very short time but are not robust for face detection in complex environments. Others have proposed texture-based approaches by detecting local facial features such as pupils, lips and eyebrows based on an observation that they are normally darker than the regions around them. Colour feature-based face detection is derived from the fact that the skin colour of different humans (even from different races) cluster very closely. Several colour models are normally used, including RGB(Satoh, Nakamura & Kanade 1999), normalised RGB(Sun, Huang & Wu 1998), HSI(Lee, Kim & Park 1996), YIQ(Wei & Sethi 1999), YES(Saber & Tekalp 1996), and YUV(Marques & Vilaplana 2000). In these colour models, HSI is shown to be very suitable when there is a large variation in feature colours in facial areas such as the eyes, eyebrows, and lips. Motion information is appropriate to detect faces or heads when video sequences are vailable (Espinosa-Duro, FaundezZanuy & Ortega 2004; Deng, Su, Zhou & Fu 2005). Recently, researchers tend to focus more on multiple feature methods which combine shape analysis, colour segmentation, and motion information to locate or detect faces (Qian & Li 2000; Widjojo & Yow 2002).

2.1.2 Template based:

This approach can be further divided into two classes: feature searching and face models. Feature searching techniques first detect the prominent facial features, such as eyes, nose, mouth, then use knowledge of face geometry to verify the existence of a face by searching for less prominent facial features (Jeng, Liao, Liu & Chern 1996). Deformable templates are generally used for face models for face detection. Yuille et al. (1989) extends the snake technique to describe features such as eyes and the mouth by a parameterized template. The snake energy comprises a combination of valley, edge, image brightness, peak, and internal energy. In Cootes and Taylor's work (1996), a point distributed model is described by a set of labelled points and Principal Component Analysis is used to define a deformable model.

2.1.3 Image based:

This approach treat face detection as a two class pattern recognition problem and avoid using a priori face knowledge. It uses positive and negative samples to train a face/non-face classifier. Various pattern classification. Various pattern classification methods are used, including Eigenfaces (Wong, Lam, Siu & Tse 2001), Neural Network (Tivive & Bouzerdoum 2004), Support Vector Machine (Shih & Liu 2005), Adaboost (Hayashi & Hasegawa 2006).

In summary, there are many varieties of face detection methods and to choose a suitable method is heavily application dependent. Generally speaking, feature-based methods are often used in real-time systems when colour, motiom, or texture information is available. Template-matching and image-based approach can attaion superior detection performance then feature-based method, but most of the algorithms are computationally expensive and are difficult to apply in a real-time system.

2.2 Viola-Jones Face Detection Algorithm:

The Viola-Jones object detection framework is the first object detection framework to provide competitive rates in real-time. It was proposed in 2001 by Paul Viola and Michael Jones. Although it can be trained to detect a variety of object classes, it was motivated primarily to the problem of face detection.

2.3 Open CV Library:

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.

In OpenCV library we are using the Viola Jones face detection algorithm which uses the following Haar-like features which are the input to the basic classifiers.

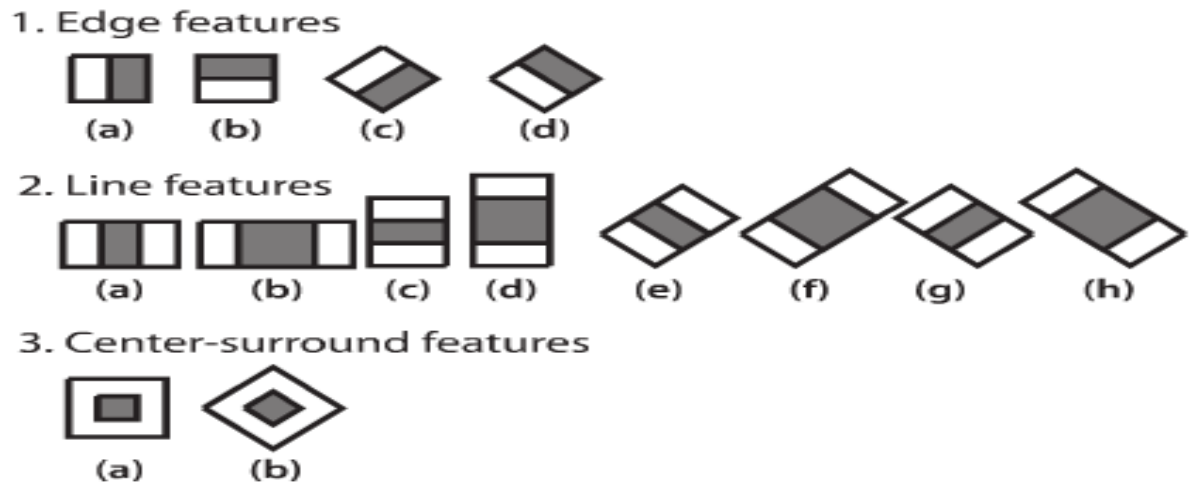
Haar Features – All human faces share some similar properties. This knowledge is used to construct certain features known as Haar Features.

The properties that are similar for a human face are:

- The eyes region is darker than the upper-cheeks.
- The nose bridge region is brighter than the eyes.

That is useful domain knowledge:

- Location – Size: Eyes and Nose bridge region
- Value: Darker/Brighter



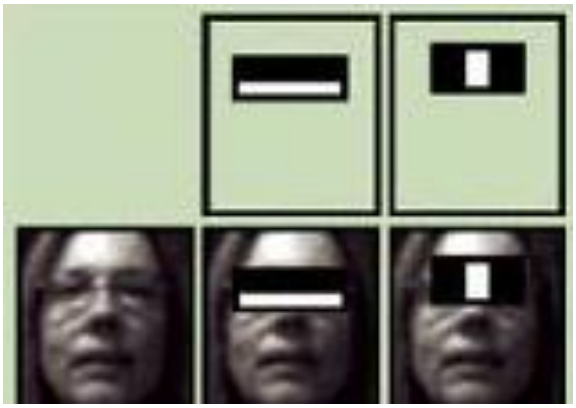


Fig 2.1: Different Haar-Like Features and their implementation

2.4 Cascade Architecture of Classifiers:

- On average only 0.01% of all sub-windows are positive (faces).
- Equal computation time is spent on all sub-windows.
- Must spend most time only on potentially positive sub-windows.
- A simple 2-feature based classifier can achieve almost 100% detection rate with 50% FP rate.
- That classifier can act as a 1st layer of a series to filter out most negative windows.
- 2nd layer with 10 features can tackle 'harder' negative-windows which survived the 1st layer, and so on…
- A cascade of gradually more complex classifiers achieves even better detection rates. The evaluation of the strong classifiers generated by the learning process can be done quickly, but it isn't fast enough to run in real time. For this reason, the strong classifiers are arranged in a cascade in order of complexity, where each successive classifier is trained only on those selected samples which pass through the preceding classifiers. If at any stage in the cascade a classifier rejects the sub-window under inspection, no further processing is performed and continue on searching the next sub-window.

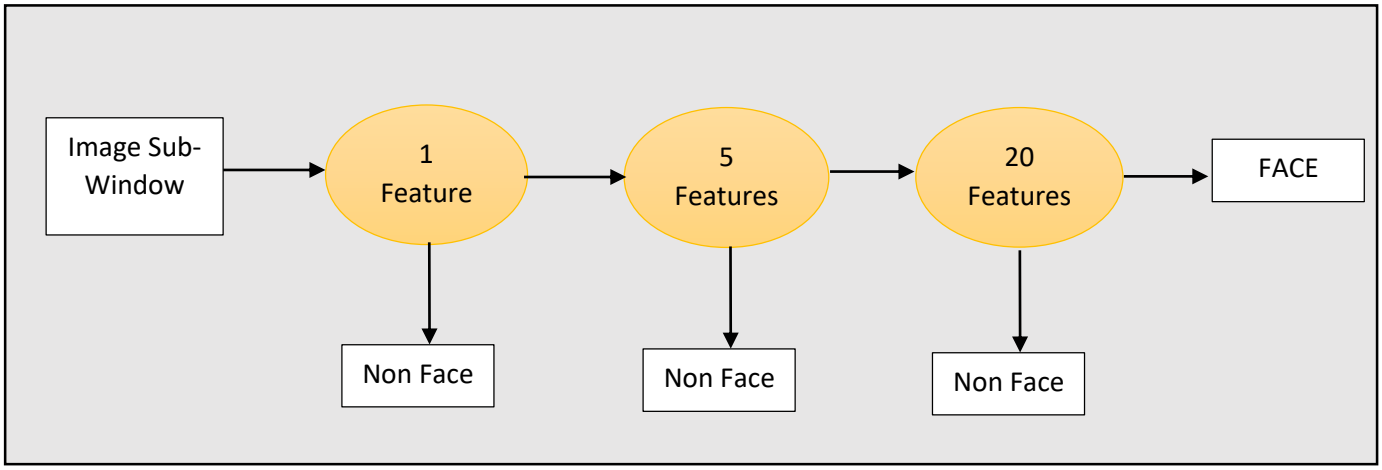


Fig 2.2: Working of Cascaded Classifier

2.5 Face Recognition:

- Face recognition is a process to detect a face, or faces and recognise them using some technique used.
- Face recognition is widely used in many fields such as Security, Policing and Criminal Investigation, Authentication.
- Face recognition uses Biometrics to detect and recognise a face.
- A biometric is a unique, measurable characteristic of a human being that can be used to automatically recognise an individual or verify an individual's identity.
- Face recognition promises to eliminate the need to remember and fill passcodes to authenticate an individual at various events.
- The authentication using 'Face Recognition' with a suitably good algorithm is considered to be much better than any other biometric recognition.

2.6 Face recognition Algorithms:

2.6.1 K Nearest Neighbours:

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbours, with the object being assigned to the class most common among its k nearest neighbours (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbour.

- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbours.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The k-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, a useful technique can be to assign weight to the contributions of the neighbours, so that the nearer neighbours contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbour a weight of 1/d, where d is the distance to the neighbour. 9

The neighbours are taken from a set of objects for which the class (for k-NN classification) or the object property value (for k-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

A peculiarity of the k-NN algorithm is that it is sensitive to the local structure of the data. The algorithm is not to be confused with k-means, another popular machine learning technique.

2.6.1.1 Woking of KNN:

The algorithm can be summarized as:

1. A positive integer k is specified, along with a new sample.
2. We select the k entries in our database which are closest to the new sample.
3. We find the most common classification of these entries
4. This is the classification we give to the new sample

A few other features of KNN:

- KNN stores the entire training dataset which it uses as its representation.
- KNN does not learn any model.
- KNN makes predictions just-in-time by calculating the similarity between an input sample and each training instance.

- Increasing the value of K does not mean that accuracy will also be increased.

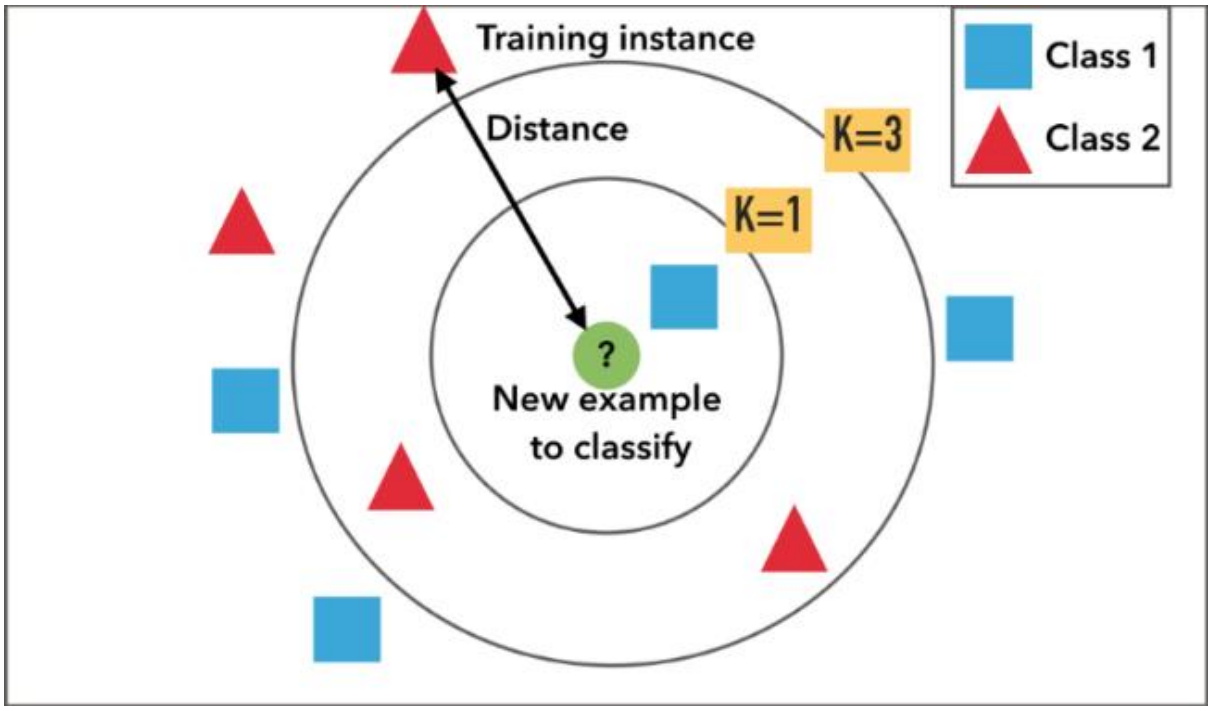


Fig 2.3: Working of KNN Algorithm

Example of KNN classification. The test sample (green circle) should be classified either to the first class of blue squares or to the second class of red triangles. If k = 3 (solid line circle) it is assigned to the second class because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the first class (3 squares vs. 2 triangles inside the outer circle).

2.6.2 Local Binary Patterns Histograms (LBPH):

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighbourhood of each pixel and considers the result as a binary number.

It was first described in 1994 (LBP) and has since been found to be a powerful feature for texture classification. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.

Using the LBP combined with histograms we can represent the face images with a simple data vector.

As LBP is a visual descriptor it can also be used for face recognition tasks, as can be seen in the following step-by-step explanation.

2.6.2.1 Woking of LBPH:

1. **Parameters**: the LBPH uses 4 parameters:
   - **Radius**: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1.
   - **Neighbours**: the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.
   - **Grid X**: the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
   - **Grid Y**: the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

2. **Training the Algorithm**: First, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information

to recognize an input image and give you an output. Images of the same person must have the same ID. With the training set already constructed, let's see the LBPH computational steps.

3. **Applying the LBP operation**: The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting the facial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameters radius and neighbours.

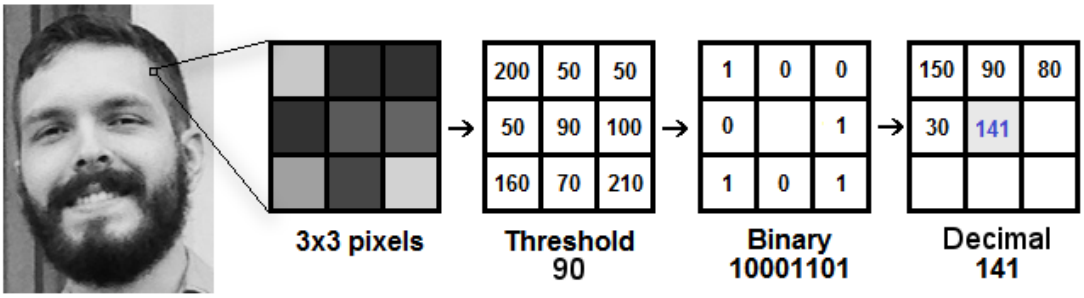


Fig 2.4: Applying the LBP operation

Based on the image above, let's break it into several small steps so we can understand it easily:

- Suppose we have a facial image in grayscale.
- We can get part of this image as a window of 3x3 pixels.
- It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).
- Then, we need to take the central value of the matrix to be used as the threshold.
- This value will be used to define the new values from the 8 neighbors.

- For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
- Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g. 10001101). Note: some authors use other approaches to concatenate the binary values (e.g. clockwise direction), but the final result will be the same.
- Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is actually a pixel from the original image.
- At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.
- Note: The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.

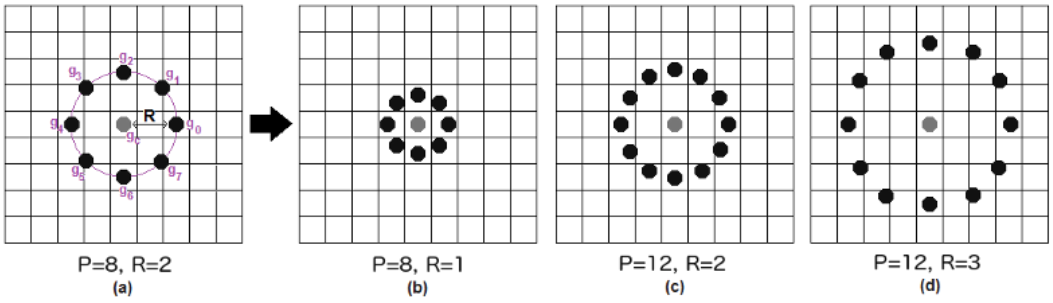


Fig 2.5: Circular LBP

It can be done by using bilinear interpolation. If some data point is between the pixels, it uses the values from the 4 nearest pixels (2x2) to estimate the value of the new data point.

4. Extracting the Histograms: Now, using the image generated in the last step, we can use the Grid X and Grid Y parameters to divide the image into multiple grids, as can be seen in the following image:

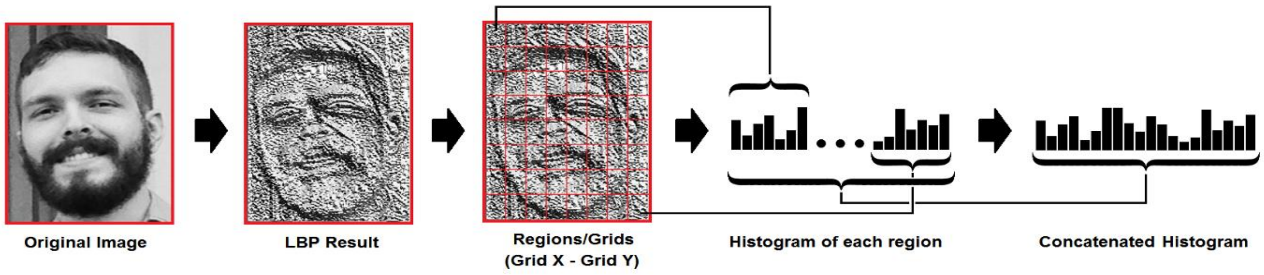


Fig 2.6: Circular LBP

Based on the image above, we can extract the histogram of each region as follows:

- As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.
- Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have 8x8x256=16.384 positions in the final histogram. The final histogram represents the characteristics of the image original image.

5. Performing the face recognition: In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

- So to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.

- We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: euclidean distance, chi-square, absolute value, etc. In this example, we can use the Euclidean distance (which is quite known) based on the following formula:
- So the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a 'confidence' measurement. Note: don't be fooled about the 'confidence' name, as lower confidences are better because it means the distance between the two histograms is closer.

We can then use a threshold and the 'confidence' to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.

2.6.2.2 Features of LBPH:

- LBPH is one of the easiest face recognition algorithms.
- It can represent local features in the images.
- It is possible to get great results (mainly in a controlled environment).
- It is robust against monotonic gray scale transformations.
- It is provided by the OpenCV library (Open Source Computer Vision Library).

2.6.3 Other Face Recognition Algorithms:

2.6.3.1 Eigen face

Eigenface is a practical approach for face recognition. Due to the simplicity of its algorithm, we could implement an Eigenface recognition system easily. Besides, it is efficient in processing time and storage. PCA reduces the dimension size of an image greatly in a short period of time. The accuracy of Eigenface is also satisfactory (over 90 %) with frontal faces.

However, as there has a high correlation between the training data and the recognition data. The accuracy of Eigenface depends on many things. As it takes the pixel value as comparison for the projection, the accuracy would decrease with varying light intensity. Besides, scale and orientation of an image will affect the accuracy greatly. Preprocessing of image is required in order to achieve satisfactory result Advantages of this algorithm are that the eigentfaces were invented exactely for that purpose what makes the system very efficient. A drawback is that it is very sensitive for lightening conditions and the position of the head, it Fast on Recognition, and Easy to implement Disadvantages-Finding the eigenvectors and eigenvalues are time consuming on PPC The size and location of each face image must remain similar PCA (Eigenface) approach maps features to principle subspaces that contains most energy

2.6.3.2 **Fisher face**

Fisherface is similar to Eigenface but with improvement in better classification of different classes image. With FLD, we could classify the training set to deal with different people and different facial expression. We could have better accuracy in facial expression than Eigen face approach. Besides, Fisherface removes the first three principal components which is responsible for light intensity changes, it is more invariant to light intensity.

Fisherface is more complex than Eigenface in finding the projection of face space. Calculation of ratio of between-class scatter to within-class scatter

requires a lot of processing time. Besides, due to the need of better classification, the dimension of projection in face space is not as compact as Eigenface, results in larger storage of the face and more processing time in recognition.

• Fisher linear discriminating (FLD, Fisherface) approach maps the feature to subspaces that most separate the two classes.

Chapter 3: Proposed Approach and Implementation

3.1 Outline:

We are using 'Python' programming language as our main coding language. We chose Python as it is widely used to implement various Machine Learning Algorithms and also OpenCV library for Python has various useful functions to assist in Face Detection and Recognition.

IDE Used: Pycharm(v2017.3.3)

Operating System Used: Linux(Ubuntu), Ubuntu has inbuilt Python compiler installed.

'OpenCV' Library for Python provides us trainer and detector both. This library is widely used to detect objects like Car, Chair, etc. but we will be using the predefined, face and eye detection classifiers of OpenCV. The detector function is based on Viola-Jones Face Detection Algorithm and is implemented using the "cvHaarDetectObjects" function of OpenCV.

As we are doing only frontal-face detection we are using this function to detect face.

cv2.CascadeClassifier('./haarcascade_frontalface_default.xml')

here cv2 is the library used to instantiate camera object.

Also, we are using the 'numpy' representation format to store the datasets.

3.2 Training Process:

The training process for both the algorithms is almost same and it consists of the following steps:

1. **Dataset selection**
2. **Running the trainer code**

3.2.1 Dataset Selection:

As we are doing real time face detection and recognition, we used ourselves and our friends as our datasets for the algorithms.

We fed faces of many different friends of ours in the dataset and almost all of the time we had more than 10 persons' data present in our database for recognition.

3.2.2 Running the trainer code:

- **For KNN:**

  The python code and explanation of each line is given here.

  Trainer code:

```
import numpy as np
import cv2

# instantiate a camera object to capture images
cam = cv2.VideoCapture(0)

# create a haar-cascade object for face detection
face_cas = cv2.CascadeClassifier('./haarcascade_frontalface_default.xml')

# create a placeholder for storing the data
data = []
ix = 0 # current frame number

while True:
    # retrieve the ret (boolean) and frame from camera
    ret, frame = cam.read()

    # if the camera is working fine, we proceed to extract the face
    if ret == True:
        # convert the current frame to grayscale
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # apply the haar cascade to detect faces in the current frame
    # the other parameters 1.3 and 5 are fine tuning parameters
        # for the haar cascade object
        faces = face_cas.detectMultiScale(gray, 1.3, 5)
```

```
            # for each face object we get, we have
            # the corner coords (x, y)
            # and the width and height of the face
            for (x, y, w, h) in faces:

                # get the face component from the image frame
                face_component = frame[y:y+h, x:x+w, :]

                # resize the face image to 50X50X3
                fc = cv2.resize(face_component, (50, 50))

                # store the face data after every 10 frames
                # only if the number of entries is less than 20
                if ix%10 == 0 and len(data) < 20:
                    data.append(fc)

            # for visualization, draw a rectangle around the face
                # in the image
            cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 255, 0), 2)
            ix += 1        # increment the current frame number
            cv2.imshow('frame', frame)    # display the frame

            # if the user presses the escape key (ID: 27)
            # or the number of images hits 20, we stop
            # recording.
            if cv2.waitKey(1) == 27 or len(data) >= 20:
                break
        else:
            # if the camera is not working, print "error"
            print "error"

# now we destroy the windows we have created
cv2.destroyAllWindows()
```

```
# convert the data to a numpy format
data = np.asarray(data)

# print the shape as a sanity-check
print data.shape

# save the data as a numpy matrix in an encoded format
np.save('akshay', data)
```

- **For LBPH:**

SetCreater code:

```
# for LBPH implementation, as we have connected our dataset
# with a database to provide
# information of the recognised people we have used sqlite3 DB

import cv2
import sqlite3

def check(a):
    conn=sqlite3.connect("pdb.sqlite")
    c=conn.cursor()
    conn.commit()
    tempe=[]
    for val in c.execute('SELECT * from people WHERE Username=?',(a,)):
        tempe.append(val)
    conn.commit()
    conn.close()
    if len(tempe)==0:
        return "TRUE"
    else:
        return "FALSE"
```

```
def counttablerows():
    conn=sqlite3.connect('pdb.sqlite')
    c=conn.cursor()
    conn.commit()
    count=0
    for val in c.execute("select count(Name) from people"):
        count=int(val[0])
    conn.commit()
    conn.close()
    return count

def adddata(b,f,d,e):
    inicount=counttablerows()
    conn=sqlite3.connect("pdb.sqlite")
    c=conn.cursor()
    conn.commit()

    c.execute('INSERT INTO people (Name,Username,Email,Mobile)
VALUES (?,?,?,?)',(b,f,d,e))
    conn.commit()
    fincount=counttablerows()
    if fincount==(inicount+1):
        return "OK"
    else:
        return "ERROR"

def getId(a):

    conn=sqlite3.connect("pdb.sqlite")
    c=conn.cursor()
    conn.commit()
    tempe=[]
    for val in c.execute('SELECT * from people WHERE
Username=?',(a,)):
```

```
        tempe.append(val)
    identity=tempe[0][0]

    return (identity)


def takedata():
    name=""
    username=""
    mobile=""
    email=""
    while True:
        name=input("Enter Your Name: ").strip()
        while True:
            username=input("Enter Username: ").strip()
            if check(username)=="TRUE":
                break
            else:
                print("Username already exists.")
                continue
        mobile=input("Enter your Mobile Number: ").strip()
        email=input("Enter your Email ID: ").strip()
        res=adddata(name,username,email,mobile)
        if res=="OK":
            print("DATA Added Successfully.")
            break
        else:
            print("Error Occured")
            continue
    return getId(username)


if __name__ == "__main__":
    # Choose the Video Camera to initialize
    cam = cv2.VideoCapture(0)
```

```
    # Adding classifier - Haarcascade xml file

detector=cv2.CascadeClassifier('haarcascade_frontalface_default.x
ml')
    # Taking Data from user
    Id=takedata()
    sampleNum=0
    # Taking a set of 21 pictures of the person.
    try:

        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."+str(Id)+'.'+  str(sampleNum)  +
".jpg", gray[y:y+h,x:x+w])

                cv2.imshow('SetCreator',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
    except:
        cam.release()
    cam.release()
    cv2.destroyAllWindows()

    print("Dataset Created successfully..")
```

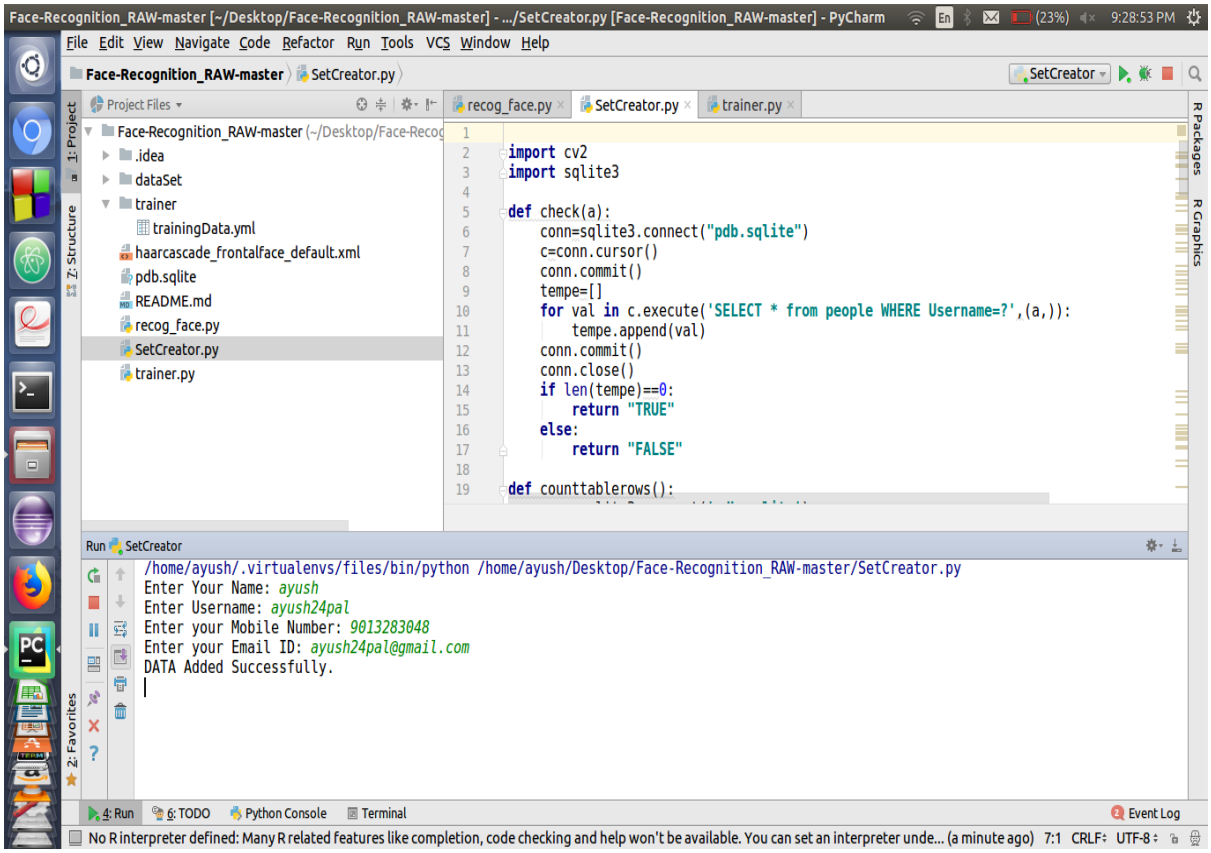
Trainer code:

```
from PIL import Image
import cv2,os
import numpy as np

recognizer = cv2.face.LBPHFaceRecognizer_create()
path = 'dataSet'


def getImageswithId(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])
        faces.append(faceNp)
        IDs.append(ID)
        cv2.imshow("Training.",faceNp)
        cv2.waitKey(10)
    return np.array(IDs),faces

Ids,faces=getImageswithId(path)
recognizer.train(faces,Ids)
recognizer.write('trainer/trainingData.yml')
cv2.destroyAllWindows()
```



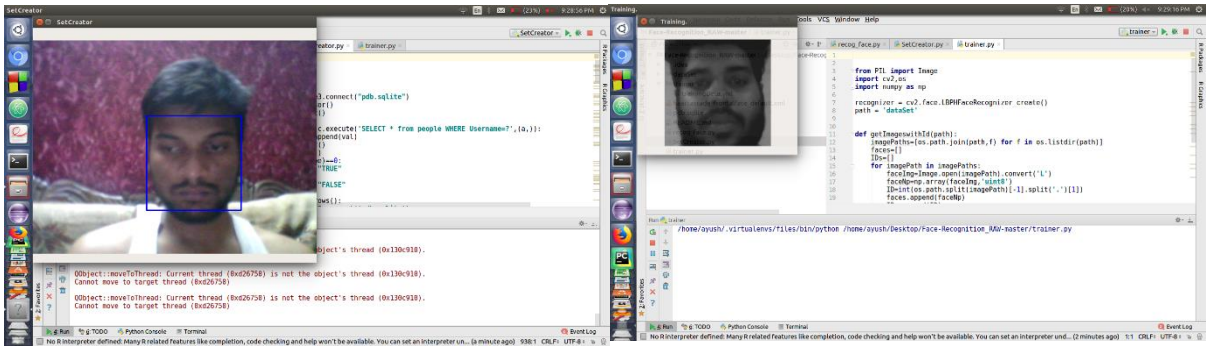


Fig 3.1: Snapshots taken during Training Process

3.3 Recognising Process:

The recognising process is fairly simple for both of the algorithms, it consists of the following steps:

- Run the recogniser code.
- Show your front face to the camera and let the recogniser, recognise your face.
- Wait for some time for the result to get stable.
- Note the result.

Recogniser Code:

- **For KNN:**

```
import numpy as np
import cv2

# instantiate the camera object and haar cascade
cam = cv2.VideoCapture(0)
face_cas = cv2.CascadeClassifier('./haarcascade_frontalface_default.xml')

# declare the type of font to be used on output window
font = cv2.FONT_HERSHEY_COMPLEX

# load the data from the numpy matrices and convert to linear vectors
f_01 = np.load('ayush.npy').reshape((20, 50*50*3))

f_02 = np.load('akshay.npy').reshape((20, 50*50*3))

f_03 = np.load('hitesh.npy').reshape((20, 50*50*3))
```

```
print f_01.shape, f_02.shape, f_03.shape
# create a look-up dictionary
names = {
        0: 'ayush',
        1: 'akshay',
        2:  'hitesh'
}

# create a matrix to store the labels
labels = np.zeros((60, 1))
labels[0:20, :] = 0.0       # first 20 for shubham (0)
labels[20:40, :] = 1.0      # next 20 for prateek (1)
labels[40:60, :] = 2.0

# combine all info into one data array
data = np.concatenate([f_01, f_02])         # (60, 7500)
print data.shape, labels.shape         # (60, 1)

# the distance and knn functions we defined earlier
def distance(x1, x2):
   return np.sqrt(((x1-x2)**2).sum())

def knn(x, train, targets, k=5):
   m = train.shape[0]
   dist = []
   for ix in range(m):
      # compute distance from each point and store in dist
      dist.append(distance(x, train[ix]))
   dist = np.asarray(dist)
   indx = np.argsort(dist)
   sorted_labels = labels[indx][:k]
   counts = np.unique(sorted_labels, return_counts=True)
   return counts[0][np.argmax(counts[1])]

while True:
```

```
    # get each frame
    ret, frame = cam.read()

    if ret == True:
        # convert to grayscale and get faces
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        faces = face_cas.detectMultiScale(gray, 1.3, 5)

        # for each face
        for (x, y, w, h) in faces:
            face_component = frame[y:y+h, x:x+w, :]
            fc = cv2.resize(face_component, (50, 50))

            # after processing the image and rescaling
            # convert to linear vector using .flatten()
            # and pass to knn function along with all the data

            lab = knn(fc.flatten(), data, labels)
        # convert this label to int and get the corresponding name
            text = names[int(lab)]

            # display the name
        cv2.putText(frame, text, (x, y), font, 1, (255, 255, 0), 2)

            # draw a rectangle over the face
        cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
        cv2.imshow('face recognition', frame)

        if cv2.waitKey(1) == 27:
            break
    else:
        print 'Error'


cv2.destroyAllWindows()
```

- **For LBPH:**

```
import numpy as np
import cv2
import sqlite3

def recognizeperson(Id):
    conn=sqlite3.connect("pdb.sqlite")
    c=conn.cursor()
    conn.commit()
    tempe=[]
    for val in c.execute('SELECT * from people where Id=?',(id,)):
        tempe.append(val)
    conn.commit()
    conn.close()
    name=tempe[0][1]
    username=tempe[0][2]
    mobile =tempe[0][4]
    email=tempe[0][3]
    st="Recognized person : NAME: "+name+" ,Username: "+username+" ,MOBILE: "+mobile+" and EMAIL: "+email
    return (st,name)


if __name__ == "__main__":

faceDetect=cv2.CascadeClassifier('haarcascade_frontalface_default.xml')
    cam=cv2.VideoCapture(0)
    rec=cv2.face.LBPHFaceRecognizer_create()
    rec.read("//home//ayush//Desktop//Face-Recognition_RAW-master//trainer//trainingData.yml")
    id=0
    name="NONE"
    font=cv2.FONT_HERSHEY_COMPLEX_SMALL
```

```
try:

    while(True):
        ret, img = cam.read()
        if ret:
            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),(255,0,0),2)
                id,conf=rec.predict(gray[y:y+h,x:x+w])
                stat,name=recognizeperson(id)
                print(stat)
                cv2.putText(img,name,(x,y+h),font,1,(0,0,255),2)
                cv2.imshow("Face",img)
            if(cv2.waitKey(1)==ord('q')):
                break
except:
    cam.release()

cam.release()

cv2.destroyAllWindows()
```

Chapter 4: Analysis and Results

4.1 Detection Results:

Detection results came out very positively as we have used the 'HaarCascadeClassifier' of OpenCV library for 'Frontal_Face' detection, for both of the algorithms and almost 100% of time the detection of face was done accurately and Non-Face Objects were never detected by the detector.

It is because of the reason that 'HaarCascadeClassifier' of OpenCV library uses a number of stages of classifiers to detect and filter images of face and non-face.

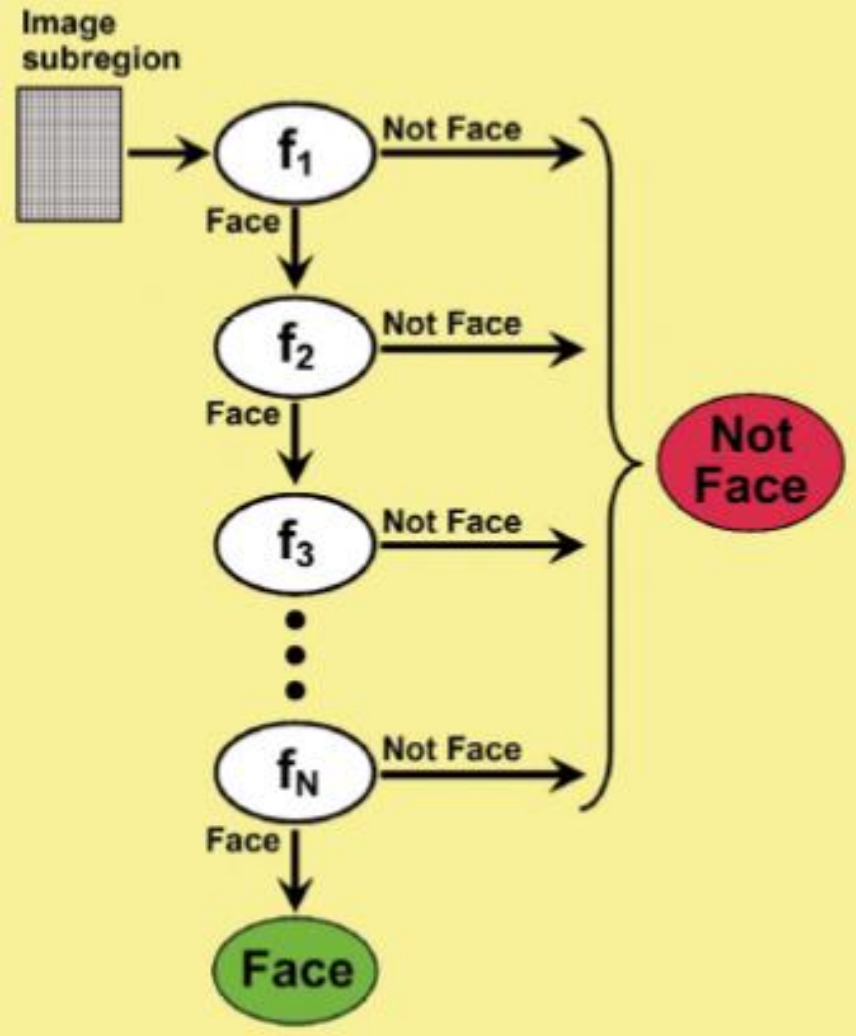


Fig 4.1: Showing Stages of Classifiers

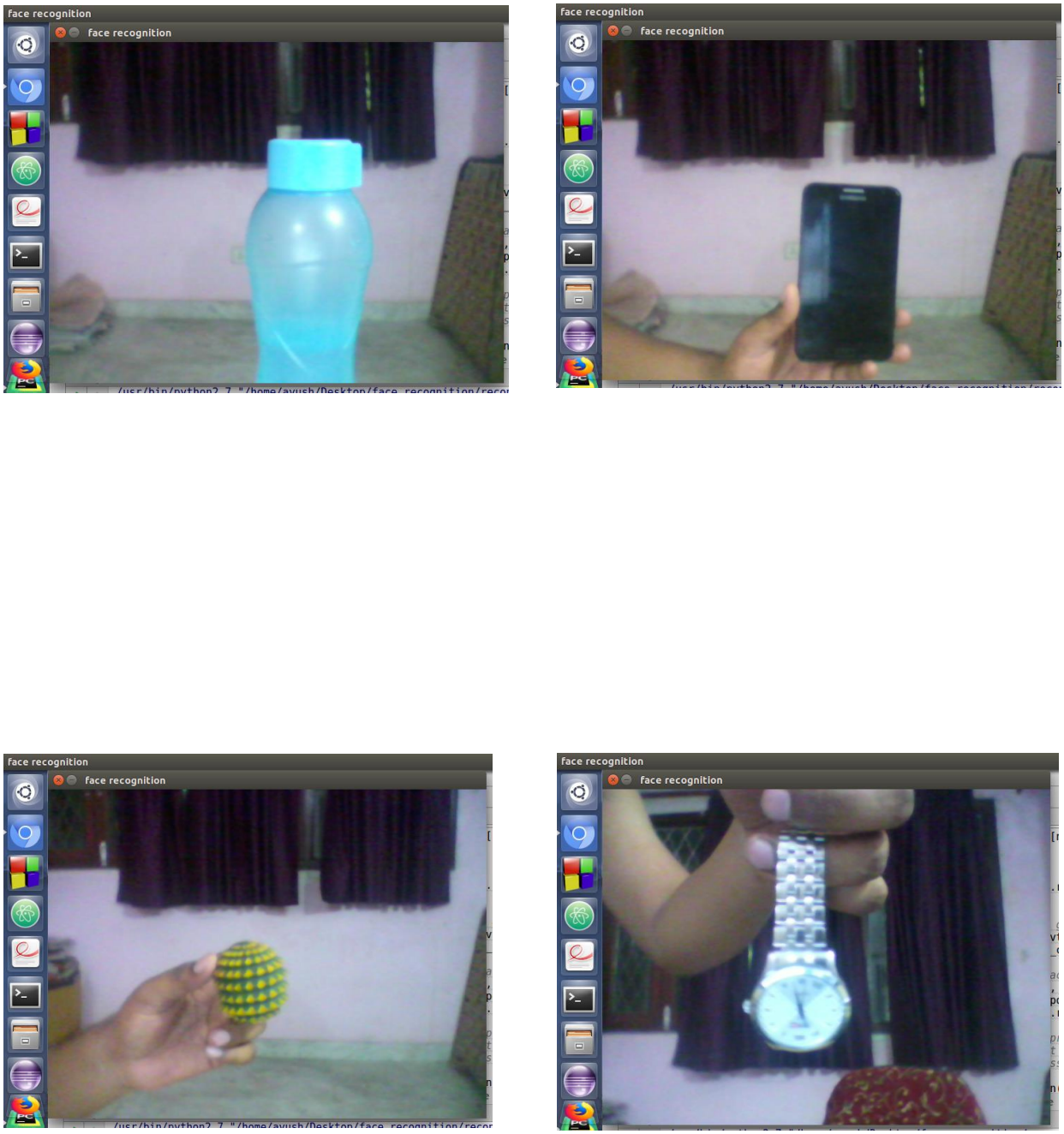


Fig 4.2: Non Detection of Non-Face Images

4.2 Recognition Results using KNN:

NO. OF FACES IN THE FRAME	TOTAL EXPERIMENTS PERFORMED	POSITIVE RESULTS
1	20	12
2	20	10
3	10	4

4.2.1 KNN Results Summary:

- As KNN Algorithm is a fairly simple algorithm and has its own shortcomings, the average accuracy came out to be 50% approximately.

- The stability of results was not good enough and the recogniser kept on changing the names of the recognised faces.

- The recognition **was sensitive** to lighting changes, on changing the light during recognition, the results came out poorly and the recogniser showed poor accuracy with KNN.

- The recognition **was not sensitive** to background change, on changing the background the results remain unaffected.

- Increasing the number of persons in the frame resulted in poor accuracy of recognition.

- The recognition remained heavily dependable on the datasets and how(orientation) the person trained his face during the training phase.

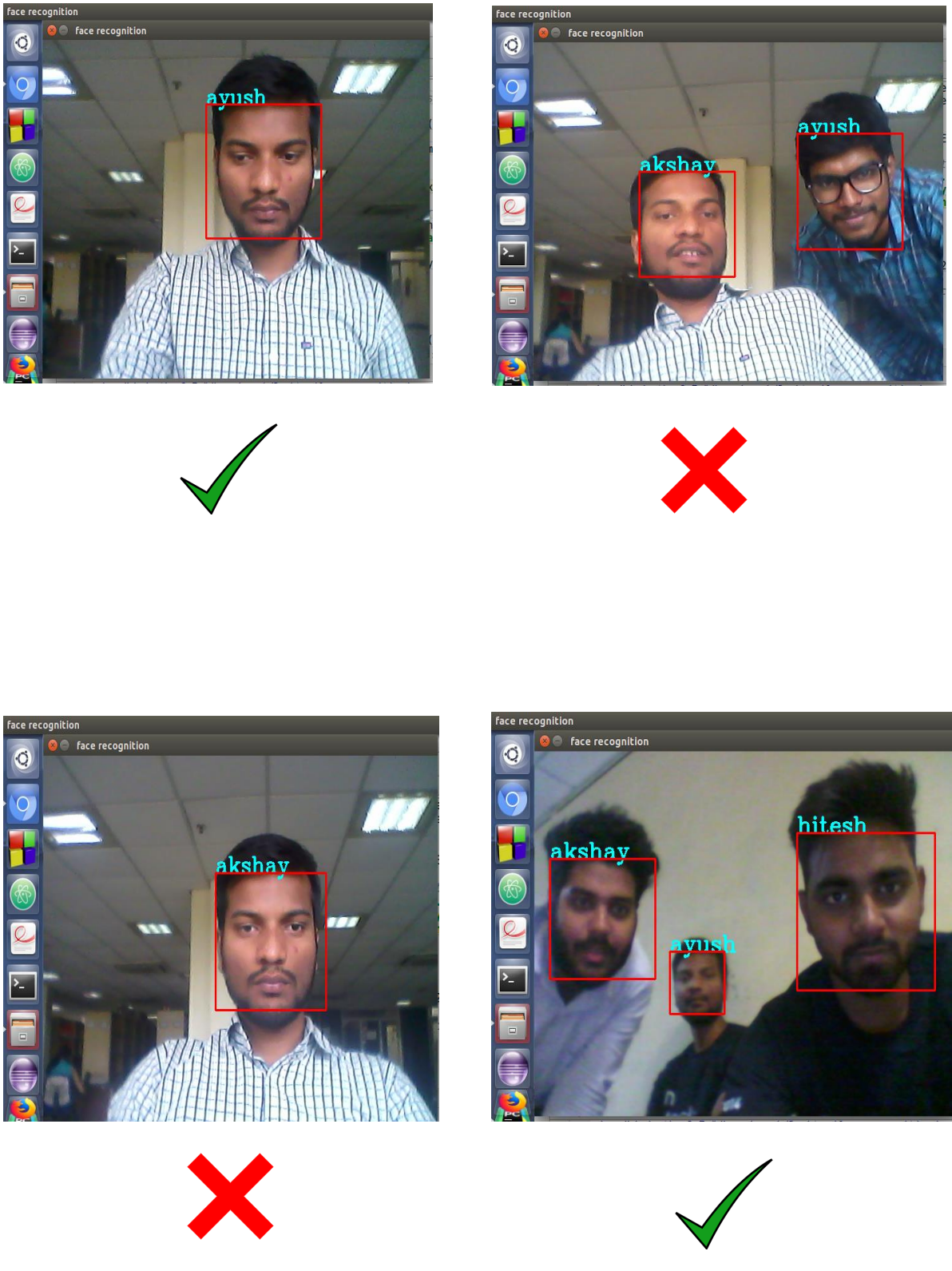


Fig 4.3: Recognition Results of KNN

4.3 Recognition Results using LBPH:

NO. OF FACES IN THE FRAME	TOTAL EXPERIMENTS PERFORMED	POSITIVE RESULTS
1	20	18
2	20	16
3	10	7

4.3.1 LBPH Results Summary:

- LBPH is a complex algorithm and came out with suitably positive results in our use case, the average accuracy came out to be 80% approximately.

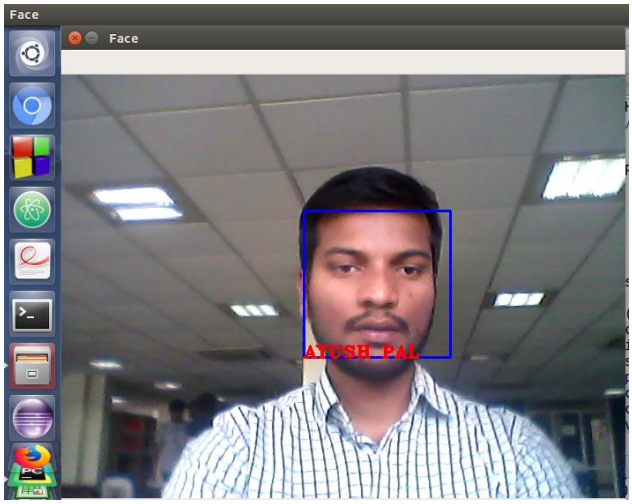
- The stability of results was good enough and the recogniser stabled to a certain name quickly.

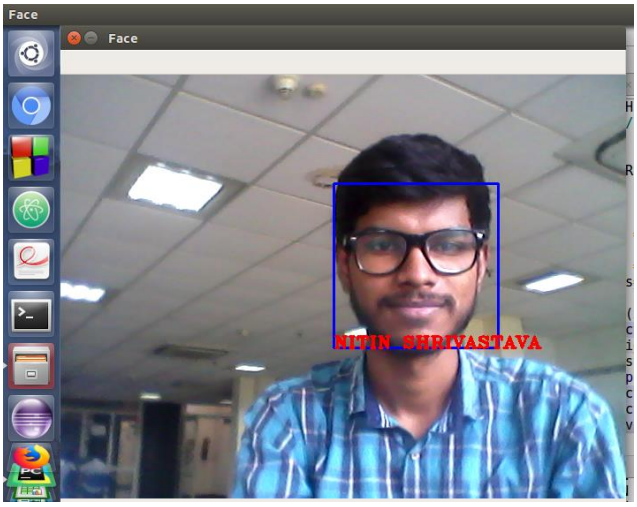
- The recognition **was not sensitive** to lighting changes, the results remain unaffected generally to light changes.

- The recognition **was not sensitive** to background change, on changing the background the results remain unaffected.

- Increasing the number of persons in the frame resulted in poor accuracy of recognition.

- The recognition remained heavily dependable on the datasets and how(orientation) the person trained his face during the training phase.





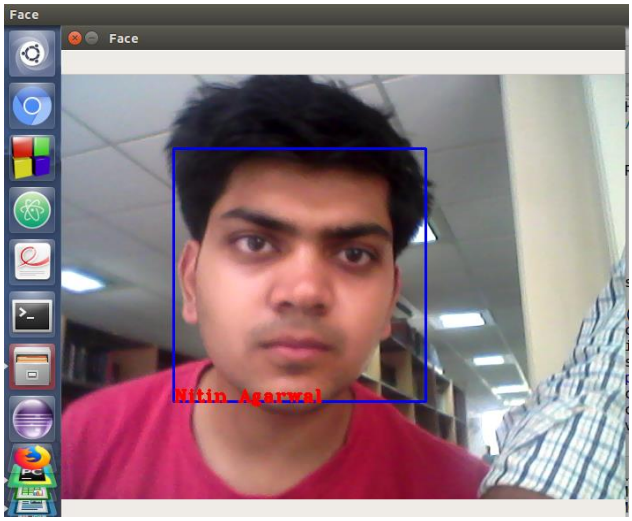


Fig 4.4: Recognition Results of LBPH with only 1 person in frame

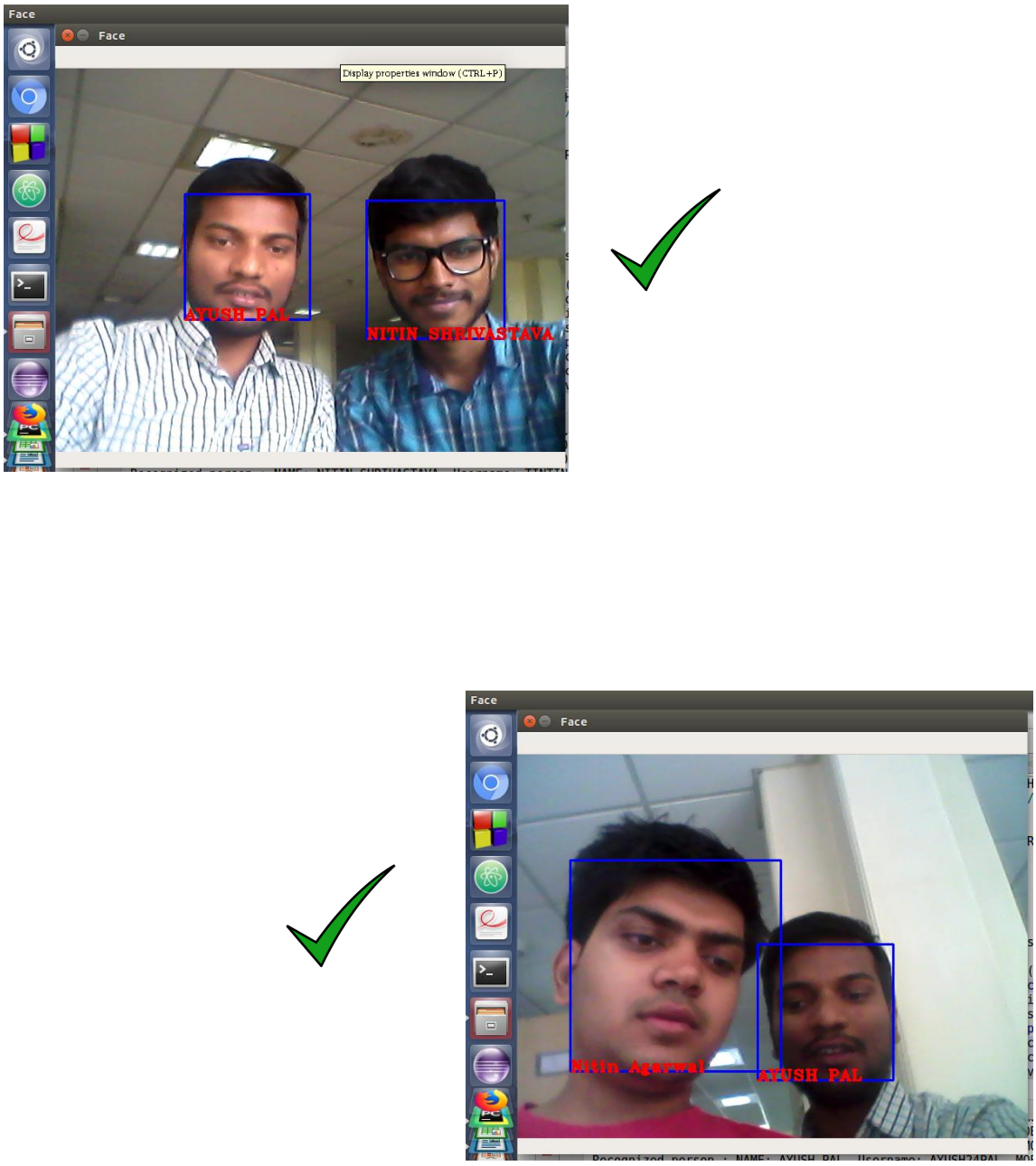
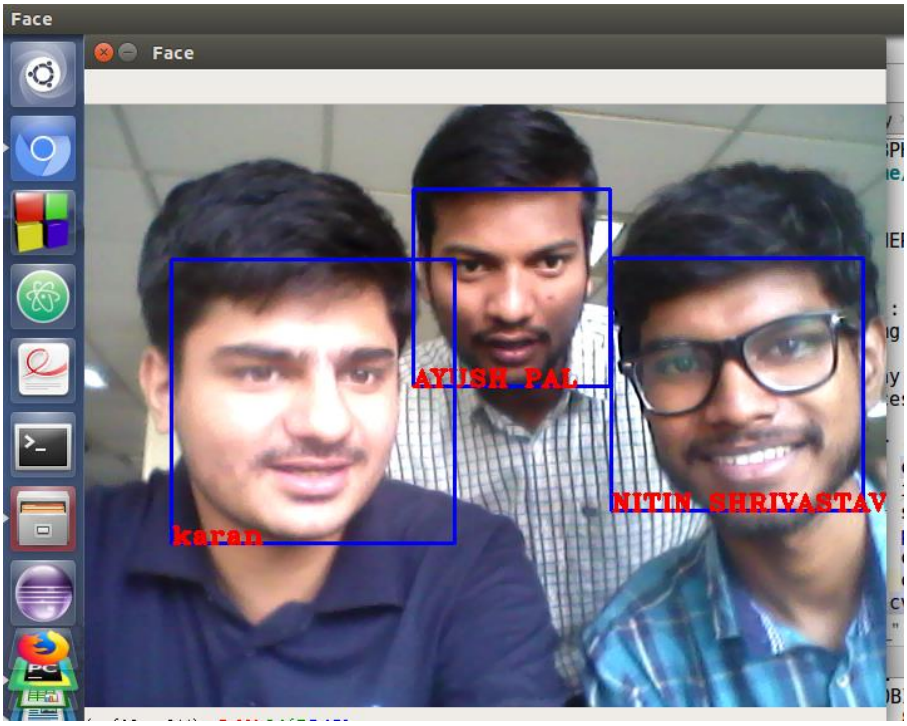


Fig 4.5: Recognition Results of LBPH with only 2 persons in frame



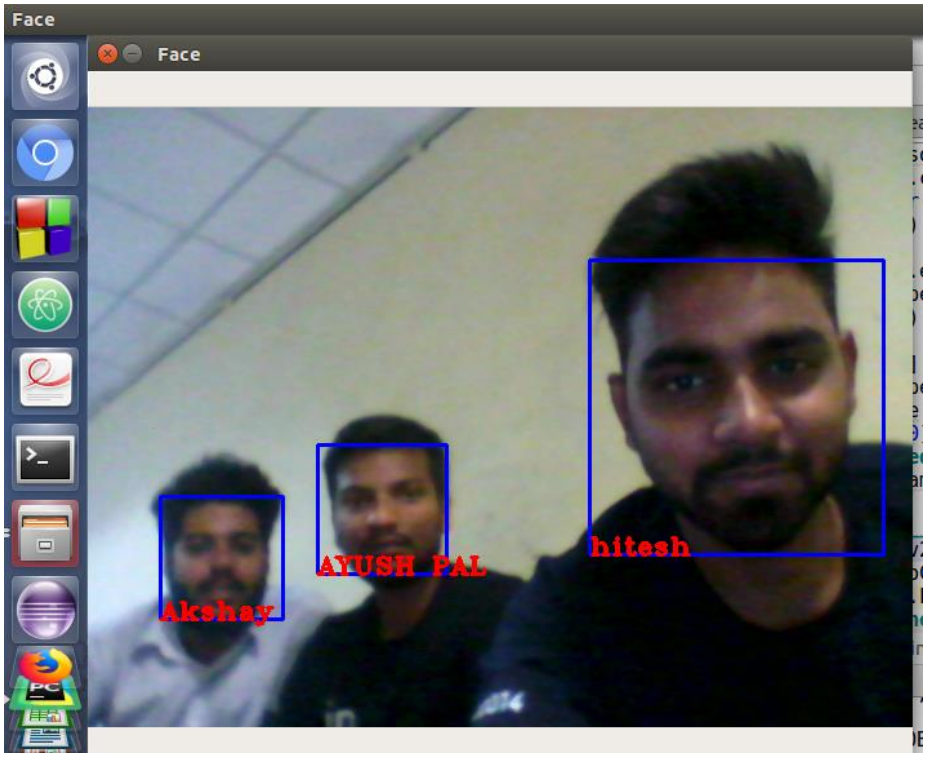


Fig 4.6: Recognition Results of LBPH with only 3 persons in frame

4.4 Advantages and Disadvantages of KNN:

Advantages:

- Simple to implement
- Flexible to feature / distance choices
- Naturally handles multi-class cases
- Can do well in practice with enough representative data
- Versatile — useful for classification or regression both.

Disadvantages:

- Computationally expensive — because the algorithm stores all of the training data.
- High memory requirement.
- Prediction stage might be slow (with big K).
- Sensitive to light, orientation and facial(e.g. beard) changes.

4.5 Advantages and Disadvantages of LBPH:

Advantages:

- LBPH is one of the easiest face recognition algorithms.
- LBPH has high accuracy in detecting and recognising face.
- It can be used to recognise expressions like smile or sad face.
- It is robust against varying lighting intensities.
- Available in OpenCV library.

Disadvantages:

- Complex algorithm requires expertise to implement.
- Comparatively harder to understand.

Chapter 5: Conclusion and Future Work

Face recognition systems used today work very well under constrained conditions, with frontal mug-shot images and constant lighting. Almost all of these fail under the vastly varying conditions under which humans need to and are able to identify other people. Next generation face detection systems will need to detect and recognize faces in real-time and in much less constrained situations.

The research and implementation presented in this thesis show promising results, but still have a wide scope for further improvements:

1. **Use of more specialised algorithms:**
   We could only implement KNN and LBPH algorithms due to limited time and resources we had at our disposal. But we can use more complex and specialised algorithms which uses concepts of Machine Learning to recognize faces for e.g. FaceNet, provided by Google.

2. **Taking into considerations other cascade classifiers:**
   We have used only frontal_face detection, we can make use of side_face, eye, smile, upper_body etc.

3. **Modifying KNN or LBPH algorithms:**
   We can modify KNN algorithm for e.g. we can research to find the best suitable value of 'K' in this use case or the number of images of a particular face to be captured for dataset.
   Similarly we can try to modify LBPH to best fit our use case by finding a trade-off between accuracy, speed and memory.

References

- https://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/
- https://www.geeksforgeeks.org/k-nearest-neighbours/
- https://docs.opencv.org/3.3.0/d7/d8b/tutorial_py_face_detection.html
- https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b
- https://www.coursera.org/learn/machine-learning
- http://www.scholarpedia.org/article/Local_Binary_Patterns
- Ahonen, Timo, Abdenour Hadid, and Matti Pietikainen. “Face description with local binary patterns: Application to face recognition.” IEEE transactions on pattern analysis and machine intelligence 28.12 (2006): 2037–2041.
- Ojala, Timo, Matti Pietikainen, and Topi Maenpaa. “Multiresolution gray-scale and rotation invariant texture classification with local binary patterns.” IEEE Transactions on pattern analysis and machine intelligence24.7 (2002): 971–987.
- Ahonen, Timo, Abdenour Hadid, and Matti Pietikäinen. “Face recognition with local binary patterns.” Computer vision-eccv 2004 (2004): 469–481.
- LBPH OpenCV: https://docs.opencv.org/2.4/modules/contrib/doc/facerec/facerec_tutorial.html#local-binary-patterns-histograms