**Training Report**

**On**

**Face-lock Algorithm**

**Submitted in partial fulfilment of the requirements**

**for the award of the degree**

**of**

**BACHELOR OF TECHNOLOGY**

**in**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By**

**Arpit Gaur**

**18015001004**

**4th Year**



**International Institute of Technology and Management**

**CANDIDATE’S DECLARATION**

We hereby declare that the work presented in this training report entitled **“Face-lock Algorithm”**, submitted in partial fulfilment of the requirement for the award of thedegree of Bachelor of Technology in **Computer Science And Engineering**, submitted to International Institute of Technology and Management, **DCRUST,** is anauthentic record of our own work carried out during the period from **4th June 2021** to **20th July 2021** under the guidance of**Mrs. Jyoti, Assistant Professor (CSE).**

The work reported in this project report has not been submitted by us for the award of any other degree or diploma.

**Arpit Gaur ( 18015001004 )**

**CERTIFICATE**

This is to certify that summer training report entitled “**Face-lock Algorithm**” done by Mr.Arpit Gaur is an authentic work carried by him under my guidance. The matter embodied in this “Face-lock Algorithm” has not been submitted earlier for the award of any degree or diploma to the best of my knowledge and belief.

**Signature of Guide Signature of H.O.D External**

Mrs. Jyoti Ms. Divya Sapra Examiner

(IITM, Murthal) (IITM, Murthal)

**ACKNOWLEDGEMENT**

I would like to acknowledge and extent our heartfelt gratitude to all those who helped me in the completion of this project.

I am highly indebted to **Mrs. Jyoti, Assistant Professor (CSE)** for his constant support and guidance for providing necessary information and details about the project.

I am also thankful to my institution, my training in-charge and my mentor for giving me an opportunity and guiding me throughout the tenure of this project.

I shall also extend my thanks to my friends, my family and last but not the least The Almighty God for encouraging me to complete the project with ease.

I perceive this opportunity as a big milestone in my career development. I will strive to use gained skills and knowledge in the best possible way, and will continue to work on their improvement, in order to attain desired career objectives.

**ABSTRACT**

**Introduction:**

This report describes my minor project that has been done in partial fulfilment of the requirements for the award of the degree of Bachelor of Technology. The title of this project is “Security using facial recognition”. This project is basically divided into 2 Modules namely Mod 1 and Mod 2.

This project report is divided into four different chapters:

Chapter 1 includes the problem statement, nee of the study, introduction and the objective of the project. It also includes the theoretical explanation about the same.

Chapter 2 includes all the necessary introduction to the projects with brief into about the functionality of Mod1 and Mod2. All the important data flow diagrams, flow hart related to project is covered in this chapter. Software and hardware required for this project is also included in this chapter.

Chapter 3 includes experimental result with all the necessary output and result with screenshots attached. This chapter also included the merit and demerit of the project and the output obtained from this project.

Chapter 4 includes conclusions and the future scope of this project. All the future idea that how can this project be extended will be cover in this chapter. The final conclusion and accuracy of this project is discussed in this chapter briefly.

**CONTENTS**

**Candidate's Declaration** i

**Certificate** ii

**Acknowledgement** iii

**Abstract** iv

|  |  |
| --- | --- |
| **Chapter No** | **Description** |
| **Chapter 1:** | **INTRODUCTION**   * 1. Objective   1.2 Modules |
| **Chapter 2:** | **SOFTWARE AND HARDWARE REQUIREMENTS** |
| **Chapter 3:** | **SOFTWARE REQUIREMENT ANALYSIS**  3.1 Overview  3.2 Feasibility Study  3.3 Software Requirement Specification  3.4 Software Requirement Validation |
| **Chapter 4:** | **SOFTWARE DESIGN**  4.1 Introduction  4.2 Control Flow Diagram  4.3 Entity- Relationship Diagram |
| **Chapter 5:** | **CODING** |
| **Chapter 6:** | **TESTING**  6.1 Introduction  6.2 Testing Methods  6.3 Testing Levels |
| **Chapter 7:** | **OUTPUT SCREENS** |
| **Chapter 8:** | **CONCLUSION** |
| **Chapter 9:** | **RECOMMENDATIONS** |
| **Chapter 10:** | **REFERENCES** |

**CHAPTER 1:**

**INTRODUCTION**

**1.1 Introduction**

The objective of the project is to provide user an algorithm which will detects the face of the user and unlock the applications or website’s according to that.

It will first take the training data as an input from the camera of the device and then will train the model from the input and will detect the face according to that training of the model.

To work on the project, we will use Facial Recognition Technique in the project.

In order to understand how Face Recognition works, let us first get an idea of the concept of a feature vector.

Every Machine Learning algorithm takes a dataset as input and learns from this data. The algorithm goes through the data and identifies patterns in the data.

For instance, suppose we wish to identify whose face is present in a given image, there are multiple things we can look at as a pattern:

* + 1. Height/width of the face.
    2. Height and width may not be reliable since the image could be rescaled to a smaller face.
    3. However, even after rescaling, what remains unchanged are the ratios – the ratio of height of the face to the width of the face won’t change.
    4. Colour of the face.
    5. Width of other parts of the face like lips, nose, etc.

Clearly, there is a pattern here – different faces have different dimensions like the ones above. Similar faces have similar dimensions. The challenging part is to convert a particular face into numbers – Machine Learning algorithms only understand numbers. This numerical representation of a “face” (or an element in the training set) is termed as a feature vector. A feature vector comprises of various numbers in a specific order.

As a simple example, we can map a “face” into a feature vector which can comprise various features like:

1. Height of face (cm)
2. Width of face (cm)
3. Average color of face (R, G, B)
4. Width of lips (cm)
5. Height of nose (cm)

Essentially, given an image, we can map out various features and convert it into a feature vector like:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Height of face  (cm) | Width of face  (cm) | Average color of  face (RGB) | Width of lips  (cm) | Height of nose  (cm) |
| 23.1 | 15.8 | (255, 224, 189) | 5.2 | 4.4 |

So, our image is now a vector that could be represented as (23.1, 15.8, 255, 224, 189, 5.2, 4.4). Ofcourse there could be countless other features that could be derived from the image (for instance, hair color, facial hair, spectacles, etc). However, for the example, let us consider just these 5 simple features.

Now, once we have encoded each image into a feature vector, the problem becomes much simpler. Clearly, when we have 2 faces (images) that represent the same person, the feature vectors derived will be quite similar. Put it the other way, the “distance” between the 2 feature vectors will be quite small.

Machine Learning can help us here with 2 things:

1. Deriving the feature vector: it is difficult to manually list down all of the features because there are just so many. A Machine Learning algorithm can intelligently label out many of such features. For instance, a complex features could be: ratio of height of nose and width of forehead. Now it will be quite difficult for a human to list down all such “second order” features.
2. Matching algorithms: Once the feature vectors have been obtained, a Machine Learning algorithm needs to match a new image with the set of feature vectors present in the corpus.

We will use OPENCV library in this project to detect the face of the person and to scan it. The name [OpenCV](http://opencv.willowgarage.com/wiki/) has become synonymous with computer vision, but what is OpenCV?

OpenCV is a collection of software algorithms put together in a library to be used by industry and academia for computer vision applications and research . OpenCV started at Intel in the mid 1990s as a method to demonstrate how to accelerate certain algorithms in hardware.

In 2000, Intel released OpenCV to the open source community as a beta version, followed by v1.0 in 2006. In 2008, Willow Garage took over support for OpenCV and immediately released v1.1.

[Willow Garage](http://www.willowgarage.com/) dates from 2006. The company has been in the news a lot lately, subsequent to the unveiling of its PR2 robot . Gary Bradski began working on OpenCV when he was at Intel; as a senior scientist at Willow Garage he aggressively continues his work on the library.

OpenCV v2.0, released in 2009, contained many improvements and upgrades. Initially, OpenCV was primarily a C library. The majority of algorithms were written in C, and the primary method of using the library was via a C API. OpenCV v2.0 migrated towards C++ and a C++ API. Subsequent versions of OpenCV added Python support, along with Windows, Linux, iOS and Android OS support, transforming OpenCV (currently at v2.3) into a cross-platform tool.

OpenCV v2.3 contains more than 2500 algorithms; the original OpenCV only had 500. And to assure quality, many of the algorithms provide their own unit tests.

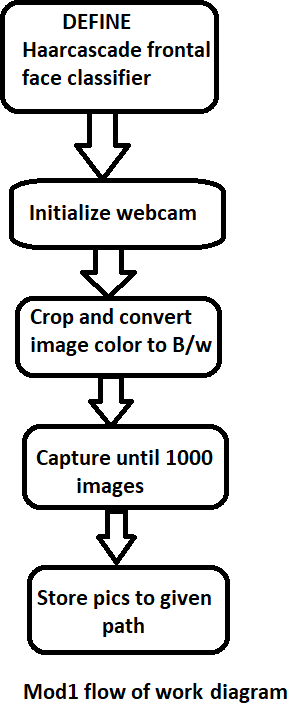
**1.2 Objective of the Study**

To develop a algorithm which provides access when face is scanned and matched.

**1.3 Modules**

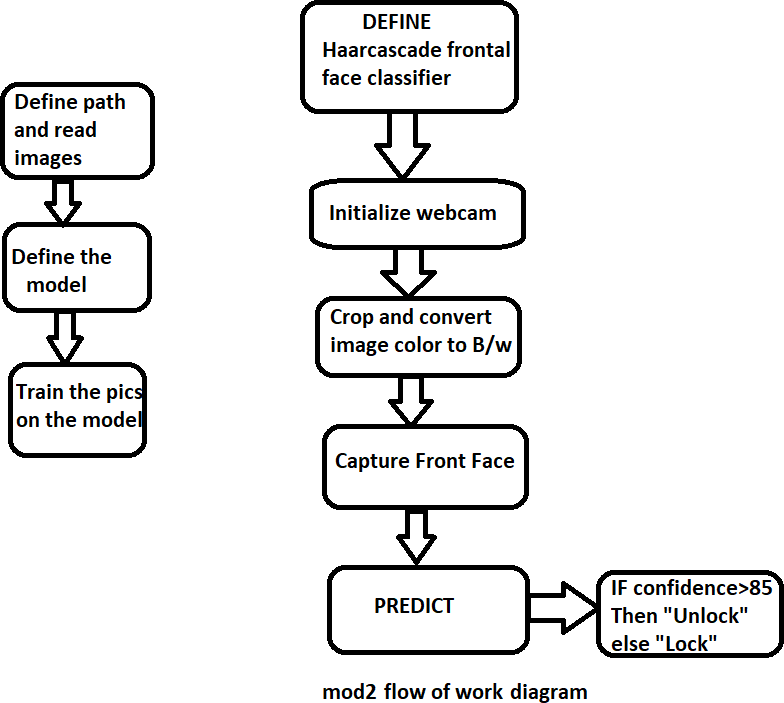
**Mod1 :-**

In the first module by the use of OpenCV library of Python :-

1. We will define the “Haarcascade\_frontal\_face” classifier as the face classifier to detect only the front face of the user for the face lock not background or any other feature.
2. We will initialize webcam using “ cv2.VideoCapture(0)”.
3. We will start capturing images by converting it into “black and white color” and we will crop the image too
4. If face is detected according to face classifier then it will capture the image otherwise “Face Not Found” will be shown .
5. This process will go until 1000 pics are captured or enter key is pressed.
6. Store Pics to given path.

**Mod2:-**

1. Define the path where pics are and then read and load pics using cv2.imread() method.
2. Define the model which we have selected as “cv2.face.LBPHFaceRecognizer\_create()”.
3. Train the training data on this model using model.train().
4. Initialize webcam again and again do same cropping and converting of image into black and white.
5. Use “Haarcascade\_frontal\_face” classifier to detect and capture the face.
6. Use the captured image to predict whether the face matched with the training data or not.
7. If confidence is above 85% then “UNLOCK” otherwise “LOCKED”.
8. “Enter” Key is used to exit.



**CHAPTER 2:**

**SOFTWARE AND HARDWARE REQUIREMENTS**

* 1. **SOFTWARE USED**

**Python :-**

We have used python latest version “ 3.8.5” in the project.

**OpenCV :-**

OpenCV (OPEN SOUCE COMPUTER VISION) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. We will use many methods of opencv in this project.

**Anaconda :-**

Anaconda Enterprise is an enterprise-ready, secure, and scalable data science platform that empowers teams to govern data science assets, collaborate, and deploy data science projects.

* 1. **HARDWARE USED**
* A windows system with webcam in it.
  1. **TECHNOLOGY USED**
* **Python :-** We have used python latest version “ 3.8.5” in the project.
* **OpenCV :-** OpenCV (OPEN SOUCE COMPUTER VISION) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. We will use many methods of opencv in this project.
* **Anaconda :-** Anaconda Enterprise is an enterprise-ready, secure, and scalable data science platform that empowers teams to govern data science assets, collaborate, and deploy data science projects.

The minimum software requirements for a “Face-Algo Project” are described below:

|  |  |  |
| --- | --- | --- |
| 1 | Operating System | Windows 10 or higher/Linux/Mac |
| 2 | Software | Anaconda Navigator |
| 3 | Software Version | 1.7.x or above |
| 4 | Environment | Spyder |
| 5 | Environment Version | 4.9.x or above |
| 6 | Language | Python |
| 7 | Module | OpenCV |
| 8 | Front-end | Anaconda |
| 9 | Back-end | OpenCV |

**Table 2.1: Software Requirements**

The minimum hardware requirements for “Face-Algo Project” are described below:

|  |  |  |
| --- | --- | --- |
| 1 | Processor | i5 6th gen or above |
| 2 | RAM | 8 GB RAM |
| 3 | Hard Disk | 7 GB |
| 4 | I/O Devices | Monitor LCD/LED, Keyboard ,Mouse, Web Camera |
| 5 | Architecture | 32/64 bit |
| 6 | Disk Space | 6.5 GB |

**Table 2.2: Hardware Requirements**

**CHAPTER 3:**

**SOFTWARE REQUIREMENT ANALYSIS**

**3.1 Overview**

In this chapter, we will study about the feasibility of the project, be it technical, economical or operational feasibility. Apart from this, we will study the requirement gathering process and software requirement specification and validation which together constitute the four basic steps of software requirement analysis. Requirements help to convey the expectations of users from the software or from the project which help in an overall analysis. The software requirement analysis is actually requirement engineering which further include these four steps.

The four steps involved in the requirement engineering are described below:-

**Figure 3.1: Software Requirement Analysis Process**

**3.2 Feasibility Study**

Feasibility Study is the complete study of project in a technical, operational and economical manner so that we can come up to a conclusion whether the software is feasible in all ways or not. In case the project is not feasible, we try to make changes or recommend changes that are to be undertaken so that the project is made feasible and is useful for each customer in all ways.

Feasibility study mainly includes testing the technical, operational and economical feasibility of a project or software.

**3.2.1 Technical Feasibility**

A software or project is said to be technically feasible if –

* The technology/tool used is easily available.
* The technology can be easily upgraded to a newer version.
* The tools used in the project and hence, the project can perform the required actions.
* The technology and software assures security in all ways

**“Face-lock Algorithm”** has been made using Anaconda and OpenCV, both of which are easily available, are upgradable and are known for security purposes. Also, the project provides immediate answers and all the necessary actions that a user may require.

**3.2.2 Operational Feasibility**

A software/project is operationally feasible if –

* It supports any number of users.
* Any action performed by the user do not undermine the advantages of system.

**“Face-lock Algorithm”** is a kind of software wherein user’s face can be registered and can be used to unlock the devices and it does not alter the functioning or benefit of the software in any way. Hence, the software is operationally feasible.

**3.2.3 Economical Feasibility**

A software is said to be economically feasible if –

* The software does not involve any extra expenditures.
* It can be viewed as a good investment for the user/company.

**“Face-lock Algorithm”** is made in Anaconda which is easily available, and allows the code to run without recompilation which saves both time and money. Also, once used and purchased, this software can be used for long. Hence, it is economically feasible.

Since software is technically, operationally and economically feasible, the software is feasible, as a whole.

**3.3 Requirement Gathering**

After the completion of feasibility study phase i.e. when the feasibility test comes out to be positive, the concern shift towards the requirement gathering phase.

In this phase, the requirements are gathered from a user or company that they want the analyst to include into the software. As in here, we gathered the requirements like adding face of a person who want to use Face-lock feature and adding terms and conditions to the project.

**3.4 Software Requirement Specification**

Software Requirement Specification, usually referred to as SRS is a document created after completion of gathering requirements from the user. It mainly includes specifying the hardware and software requirements to the user.

**Hardware Requirements:**

* Processor: Intel Core i5 series or above
* RAM: Minimum of 8 GB
* Hard Disk: Minimum of 7 GB
* Architecture: 32/64 bit

**Software Requirements:**

* Operating System: Windows 10 or later
* Front End: Anaconda
* Back End: OpenCV

Also, the requirements of user are written in the natural language. Here, the user aims to check the Schemes and Contacts available for her, the exact address of the Authority, the contact information of it. Also, the questions need to be asked and user should be made aware of the terms and requirements.

**3.5 Software Requirement Validation**

After software requirement, the software requirements need to be validated in order to make the software bug free i.e. the user must not interpret his/her requirements in such a way that it may later result into errors. Hence, validations are used wherever required.

**CHAPTER 4:**

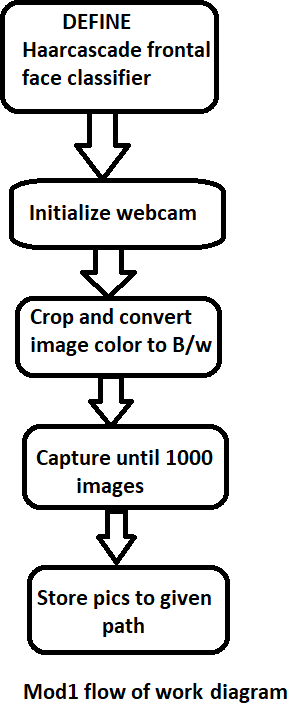
**SOFTWARE DESIGN**

**4.1 Methodology**

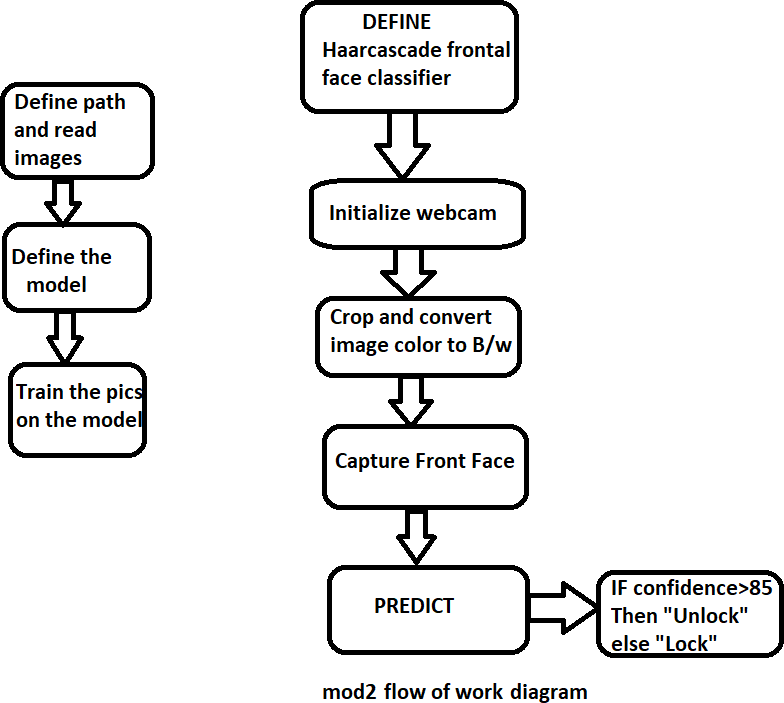
There will be two modules in this project namely “**Mod1”** and “**Mod2”**.

**Mod1 :-**

In the first module by the use of OpenCV library of Python :-

1. We will define the “Haarcascade\_frontal\_face” classifier as the face classifier to detect only the front face of the user for the face lock not background or any other feature.
2. We will initialize webcam using “ cv2.VideoCapture(0)”.
3. We will start capturing images by converting it into “black and white color” and we will crop the image too.
4. If face is detected according to face classifier then it will capture the image otherwise “Face Not Found” will shown.
5. This process will go until 1000 pics are captured or enter key is pressed.
6. Store Pics to given path.

**Mod2:-**

1. Define the path where pics are and then read and load pics using cv2.imread() method.
2. Define the model which we have selected as “ cv2.face.LBPHFaceRecognizer\_create()”.
3. Train the training data on this model using model.train().
4. Initialize webcam again and again do same cropping and converting of image into black and white.
5. Use “Haarcascade\_frontal\_face” classifier to detect and capture the face.
6. Use the captured image to predict whether the face matched with the training data or not.
7. If confidence is above 85% then “UNLOCK” otherwise “LOCKED”.
8. “Enter“ Key is used to exit.

**ABOUT HARCASCADE CLASSIFIER**

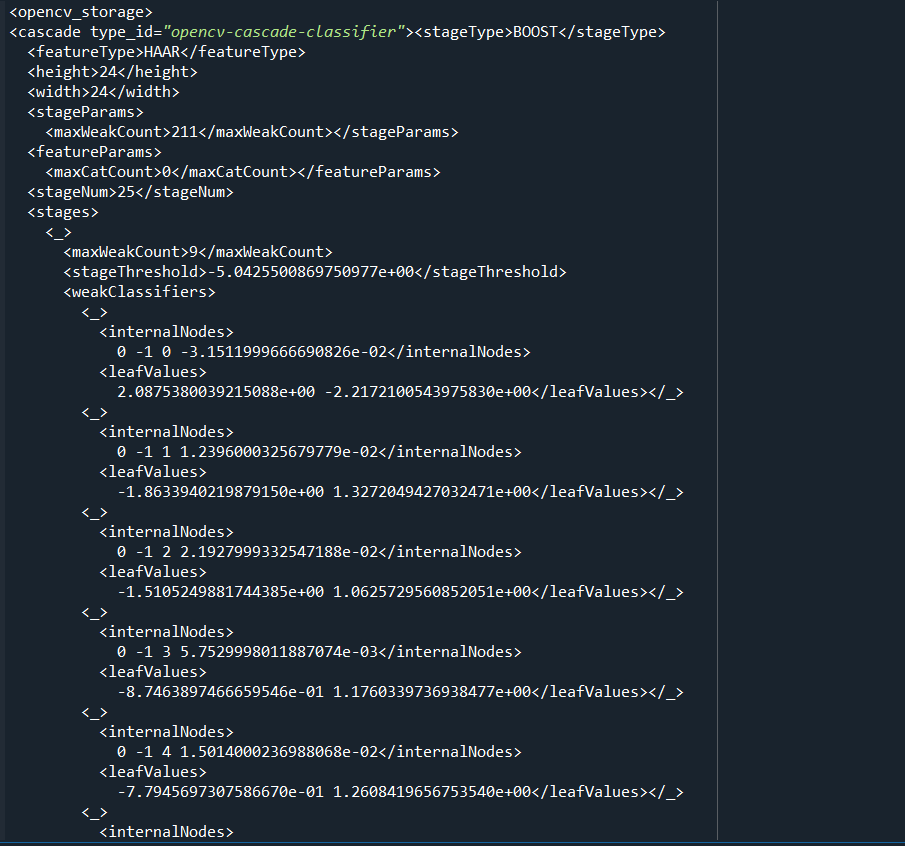
This is basically a machine learning based approach where a cascade function is trained from a lot of images both positive and negative. Based on the training it is then used to detect the objects in the other images.

**USECASES OF HARCASCADE CLASSIFIERS**

* + 1. FACE DETECTION using haarcascade\_frontalface\_default.xml
    2. FACE AND EYE DETECTION using haarcascade\_eye.html
    3. VEHICLE DETECTION FROM STREAMING VIDEO using haarcascade\_car.xml
    4. PEDESTRIAN DETECTION FROM STREAMING VIDEO using haarcascade\_fullbody.xml

## **HARCASCADE FRONTAL FACE CLASSIFIER**

It is used to detect only the front face of the user and eliminates all the background details



**ABOUT LBPH MODEL**

Local Binary Pattern (LBP) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood 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.

**PARAMETERS OF LBPH MODEL**

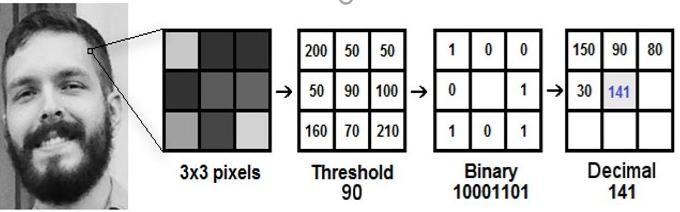
1. Radius: the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1.
2. Neighbours: the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.
3. 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.
4. 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.

**TRAINING LBPH MODEL**

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.

**STEPS TO APPLY LBPH 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 neighbors.



* we need take the central value of the matrix of pixel values to be used as the threshold. For each neighbour 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).
* 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.
* 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 and create histograms.

**FACE RECOGNITION BY LBPH**

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.

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.

**CHAPTER 5:**

**CODING**

**Mod1:**

import cv2

import numpy as np

# Load HAAR face classifier

face\_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade\_frontalface\_default.xml')

# Load functions

def face\_extractor(img):

# Function detects faces and returns the cropped face

# If no face detected, it returns the input image

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

faces = face\_classifier.detectMultiScale(gray, 1.3, 5)

if faces is ():

return None

# Crop all faces found

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

cropped\_face = img[y:y+h, x:x+w]

return cropped\_face

# Initialize Webcam

cap = cv2.VideoCapture(0)

count = 0

# Collect 1000 samples of your face from webcam input

while True:

ret, frame = cap.read()

if face\_extractor(frame) is not None:

count += 1

face = cv2.resize(face\_extractor(frame), (200, 200))

face = cv2.cvtColor(face, cv2.COLOR\_BGR2GRAY)

# Save file in specified directory with unique name

file\_name\_path = ' D:\Project output/' + str(count) + '.jpg'

cv2.imwrite(file\_name\_path, face)

# Put count on images and display live count

cv2.putText(face, str(count), (50, 50), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,255,0), 2)

cv2.imshow('Face Cropper', face)

else:

print("Face not found")

pass

if cv2.waitKey(1) == 13 or count == 1000: #13 is the Enter Key

break

cap.release()

cv2.destroyAllWindows()

print("Collecting Samples Complete")**Mod2:**

import cv2

import numpy as np

from os import listdir

from os.path import isfile, join

# Get the training data we previously made

data\_path = ' D:\Project output/'

onlyfiles = [f for f in listdir(data\_path) if isfile(join(data\_path, f))]

# Create arrays for training data and labels

Training\_Data, Labels = [], []

# Open training images in our datapath

# Create a numpy array for training data

for i, files in enumerate(onlyfiles):

image\_path = data\_path + onlyfiles[i]

images = cv2.imread(image\_path, cv2.IMREAD\_GRAYSCALE)

Training\_Data.append(np.asarray(images, dtype=np.uint8))

Labels.append(i)

# Create a numpy array for both training data and labels

Labels = np.asarray(Labels, dtype=np.int32)

# Initialize facial recognizer

model = cv2.face.LBPHFaceRecognizer\_create()

# NOTE: For OpenCV 3.0 use cv2.face.createLBPHFaceRecognizer()

# Let's train our model

model.train(np.asarray(Training\_Data), np.asarray(Labels))

print("Model trained sucessefully")

face\_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade\_frontalface\_default.xml')

def face\_detector(img, size=0.5):

# Convert image to grayscale

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

faces = face\_classifier.detectMultiScale(gray, 1.3, 5)

if faces is ():

return img, []

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

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

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

roi = cv2.resize(roi, (200, 200))

return img, roi

# Open Webcam

cap = cv2.VideoCapture(0)

while True:

ret, frame = cap.read()

image, face = face\_detector(frame)

try:

face = cv2.cvtColor(face, cv2.COLOR\_BGR2GRAY)

# Pass face to prediction model

# "results" comprises of a tuple containing the label and the confidence value

results = model.predict(face)

if results[1] < 500:

confidence = int( 100 \* (1 - (results[1])/400) )

display\_string = str(confidence) + '% Confident it is User'

cv2.putText(image, display\_string, (100, 120), cv2.FONT\_HERSHEY\_COMPLEX, 1, (255,120,150), 2)

if confidence > 80:

cv2.putText(image, "Unlocked", (250, 450), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,255,0), 2)

cv2.imshow('Face Recognition', image )

else:

cv2.putText(image, "Locked", (250, 450), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,0,255), 2)

cv2.imshow('Face Recognition', image )

except:

cv2.putText(image, "No Face Found", (220, 120) , cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,0,255), 2)

cv2.putText(image, "Locked", (250, 450), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,0,255), 2)

cv2.imshow('Face Recognition', image )

pass

if cv2.waitKey(1) == 13: #13 is the Enter Key

break

cap.release()

cv2.destroyAllWindows()

cv2.imshow()

import time

# Load HAAR face classifier

face\_classifier = cv2.CascadeClassifier('Haarcascades/haarcascade\_frontalface\_default.xml')

# Load functions

def face\_extractor(img):

# Function detects faces and returns the cropped face

# If no face detected, it returns the input image

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

faces = face\_classifier.detectMultiScale(gray, 1.3, 5)

if faces is ():

return None

# Crop all faces found

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

cropped\_face = img[y:y+h, x:x+w]

return cropped\_face

# Initialize Webcam

cap = cv2.VideoCapture(0)

count = 0

lst=[]

start\_time=time.time()

elapsed\_time=time.time()

# Collect 100 samples of your face from webcam input

while True:

ret, frame = cap.read()

if face\_extractor(frame) is not None:

start\_time = time.time()

face = cv2.resize(face\_extractor(frame), (200, 200))

face = cv2.cvtColor(face, cv2.COLOR\_BGR2GRAY)

# Put count on images and display live count

cv2.putText(face, str(time.clock()), (50, 50), cv2.FONT\_HERSHEY\_COMPLEX, 1, (0,255,0), 2)

cv2.imshow('Face Cropper', face)

else:

print("Face not found")

elapsed\_time = time.time() - start\_time

lst.append(elapsed\_time)

seconds=0

time.sleep(1)

pass

if cv2.waitKey(1) == 13: #13 is the Enter Key

break

cap.release()

cv2.destroyAllWindows()

print("Collecting Samples Complete")

**CHAPTER 6:**

**TESTING**

**6.1 Introduction**

Software testing can be thought of an investigation technique which is performed in order to check the software for errors and make it bug free i.e. error free. It provides the information about the quality of the software under investigation.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

* meets the requirements that guided its design and development,
* responds correctly to all kinds of inputs,
* performs its functions within an acceptable time,
* can be installed and run in its intended environments, and
* Achieves the general result its stakeholder’s desire.

**6.2 Testing Methods**

The two traditional testing methods are -

**Figure 6.1: Testing Methods**

**6.2.1 Black Box Testing**

Black box testing is a type of testing in which the software is treated as a “black box”. The tester does not know how the software does it but is only aware of what the software needs to do. One advantage of the black box technique is that no programming knowledge is required.

https://upload.wikimedia.org/wikipedia/commons/thumb/f/f6/Blackbox.svg/200px-Blackbox.svg.png

**Figure 6.2: Black Box Testing**

This type of testing can be applied to various levels of testing.

**6.2.2 White Box Testing**

White box testing is used to test the internal structure of a program. Unlike black box testing, white box testing requires programming skills.  It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

**6.3 Testing Levels**

There are generally four levels of testing which are described in the figure below:

**Figure 6.3: Testing Levels**

**6.3.1 Unit Testing**

Unit testing refers to the testing of a specific section. Here, we mean testing a particular module. In the project titled “Face Lock Algorithm”, unit testing is done. For each module, testing is done individually.

**6.3.2 Integration Testing**

The integration testing refers to testing done when software components are integrated. These software components may be integrated in an iterative way or altogether. Here, for instance, when modules are connected, errors may arise or it may happen that a software component individually works perfectly but when it is integrated, some bugs may arise. Integration testing is done to detect such errors.

**6.3.3 Component Interface Testing**

Such type of testing is used when data is to be handled when passed between various units, or subsystems. The data can be considered as “message packets”. Component interface testing is a variation of [black-box testing](https://en.wikipedia.org/wiki/Black-box_testing), with the focus on the data values beyond just the related actions of a subsystem component.

**6.3.4 System Testing**

System testing tests a completely integrated system to verify that the system meets its requirements. For example, a system test might involve testing a logon interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

**CHAPTER 7:**

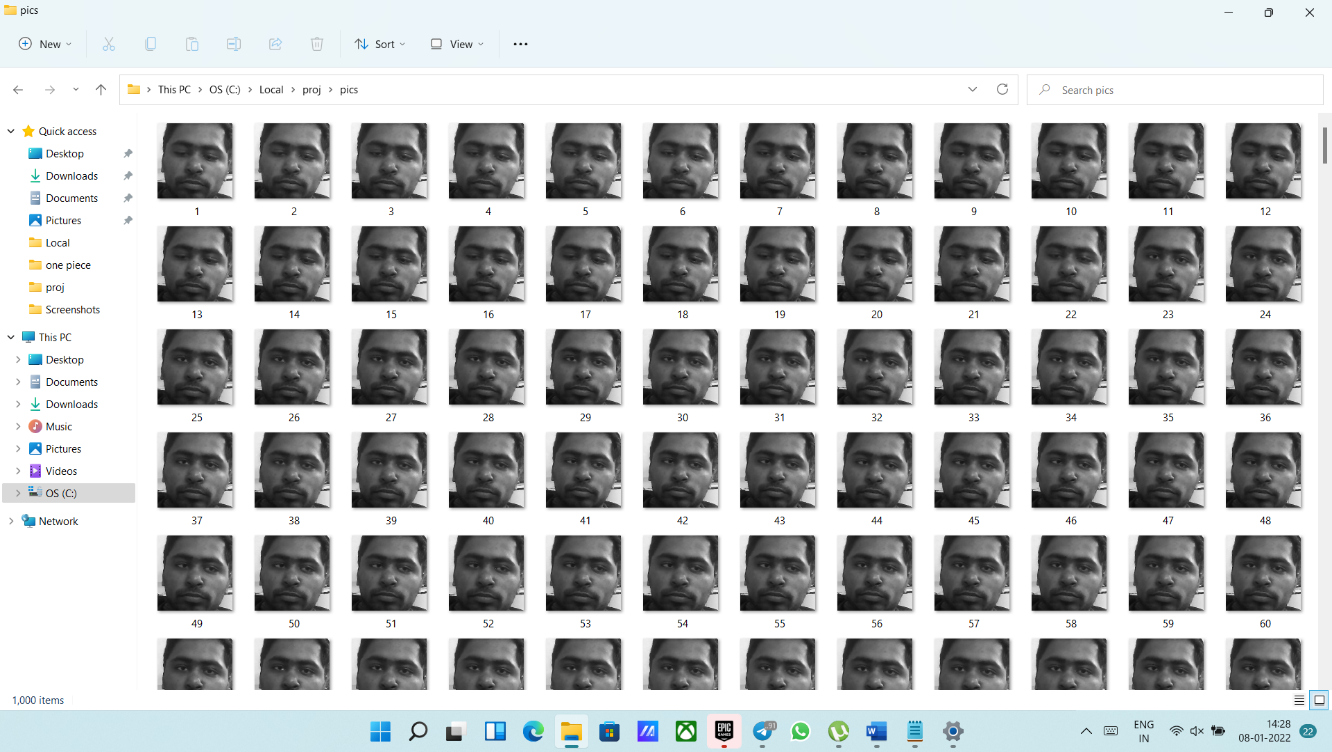
**OUTPUT SCREENS**

**Result and output**

This project has two modules named Mod1 and Mod2. Mod1 will capture and store the 1000 images of user after extracting the frontal face only. Each and every image will be stored with a unique name in the local memory.

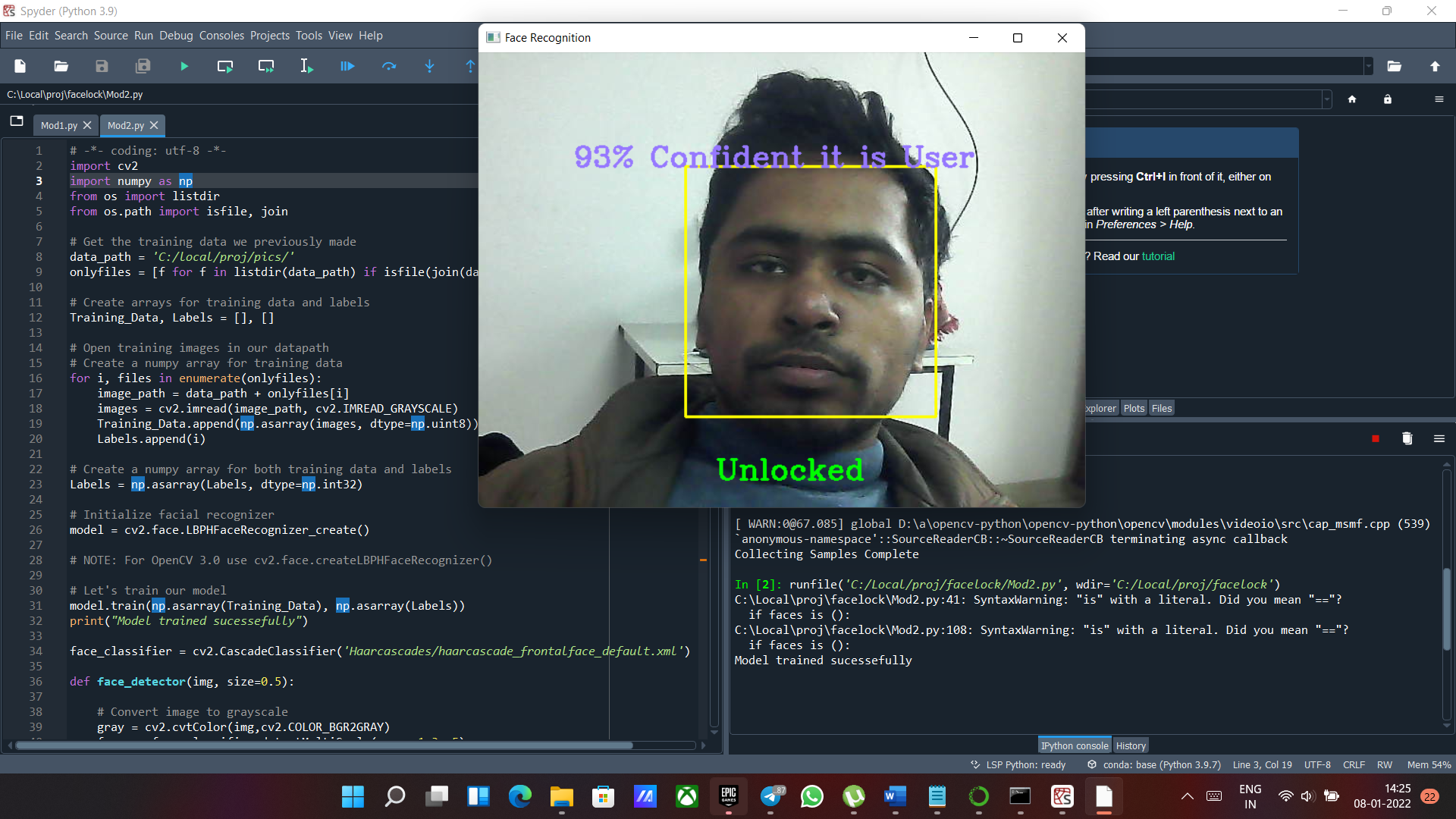


**Figure** **7.1** **Capturing 1000 images of user**



**Figure 7.2 1000 stored images**

These 1000 images will be used in Mod2 as the training data. After completion of training, a window will pop up that try to match the face of user with the training result.



If our model finds the face and that particular face match with the trained data then it will show the Unlocked message else Locked.

Also, this module prints the accuracy of our model as well i.e., in this particular case our model is 93% sure that it’s user.

**Merits:**

* The biggest of this project that we can use this security mechanism in websites, Apps and any other specific platform applications. As mobile phones or website’s security is always a major issue, to ensure the security and privacy, this project will play an important role in the same field.
* It will capture the data in real time which hardly takes 4-5 minutes. The training duration is also very short which completes in about few minutes.
* This project is portable as well and can be used on any OS which has python installed in it.

**Demerit:**

* It requires proper lightning while capturing images.

**CHAPTER 8:**

**CONCLUSION**

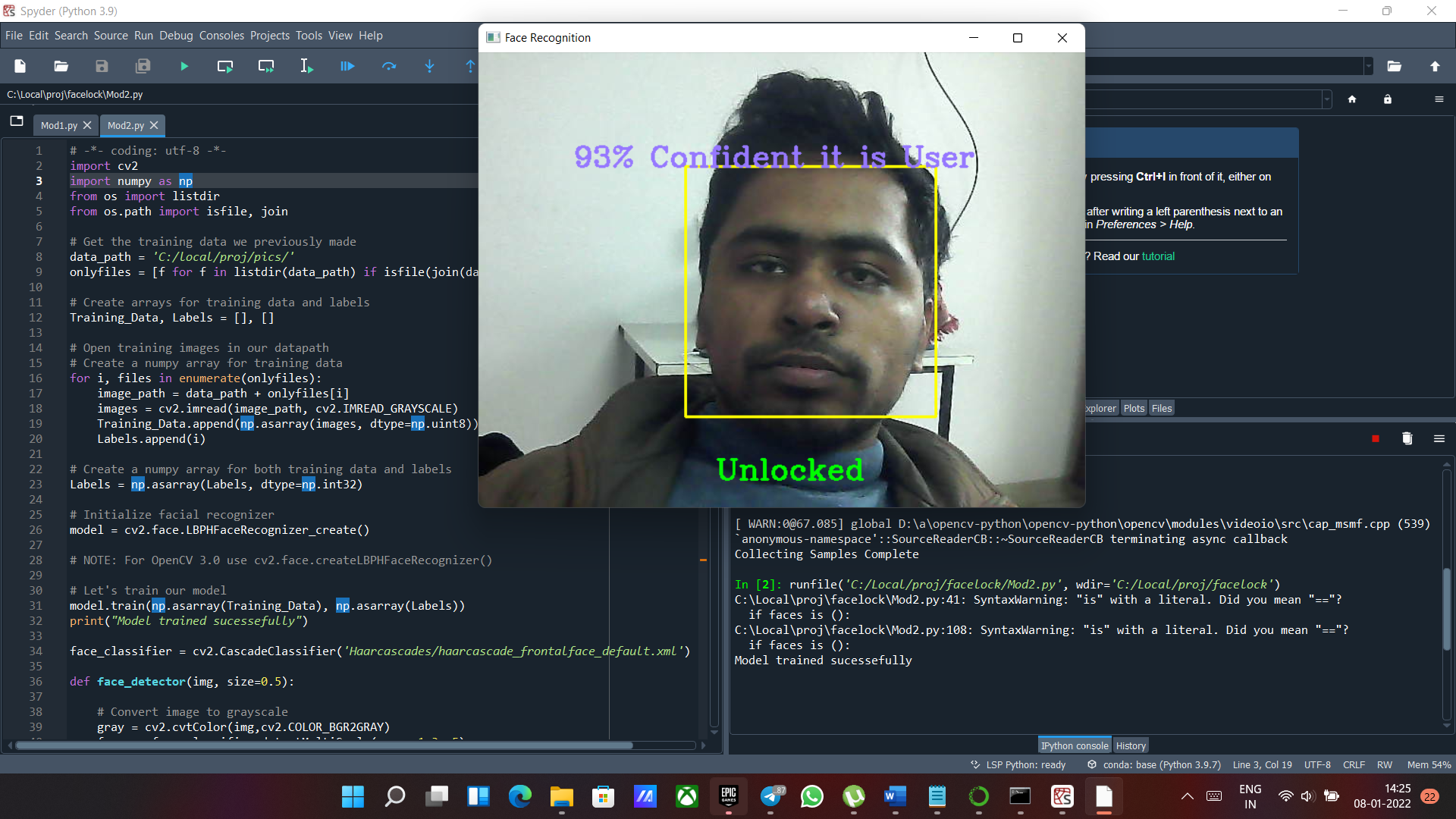
**8.1 Conclusions**

1000 images that has been captured in Mod1 will be used in Mod2 as the training data. After completion of training, a window will pop up that try to match the face of user with the training result.



**Figure 8.1 Capturing image**

**Figure**



**8.2 Final output**

The has been matched successfully with 92% accuracy.

**8.2 Future Scope**

This project provides the vast opportunities of changes it. We can implement this security mechanism in website and apps. Extension and managing this project are quite easy as we can implement age, gender classification also.

We can also implement the face expression classification in this project which will provide more flexibility of our project as it will only unlock our application when normal expression found.

**CHAPTER 9:**

**RECOMMENDATIONS**

* This face-lock algorithm can be used in any device.
* Hereby, this project is an attempt to help all people out there who need to report on how face-lock algorithm works in devices.
* This algorithm based project is fruitful for those people who want to learn about working of face-lock in their devices.

**CHAPTER 10:**

**REFERENCES**

1. <https://docs.opencv.org/master/d6/d00/tutorial_py_root.html>
2. <https://docs.opencv.org/3.4/javadoc/org/opencv/face/LBPHFaceRecognizer.html>
3. D. B. Desai and S. N. Kavitha, "Face Anti-spoofing Technique Using CNN and SVM," 2019 International Conference on Intelligent Computing and Control Systems (ICCS), Madurai, India, 2019, pp. 37-41, doi: 10.1109/ICCS45141.2019.9065873.
4. Jenkins R, McLachlan JL, Renaud K. 2014. Facelock: familiarity-based graphical authentication.

[**https://doi.org/10.7717/peerj.444**](https://doi.org/10.7717/peerj.444)

1. Authentication Lock for Application Integration Face Recognition Security Muhammad Aliff Romi Bin Sharipudin, Firoz bin Yusuf Patel Dawoodi
2. A short review paper on Face detection using Machine learning Farhad Navabifar(1), Mehran Emadi (2)Rubiyah Yusof(3) ,Marzuki Khalid(4)