FACELOCK ALGORITHM

APARNA

PROBLEM STATEMENT

The security of applications in mobile phones or websites is always a major issue to ensure the security and privacy, we designed a Machine Learning Algorithm which will only unlock the application or website when it will scan your face.

OBJECTIVE AND SCOPE

The objective of the project is to provide user an algorithm which will detect the face of the user and unlock the applications or websites 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

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.

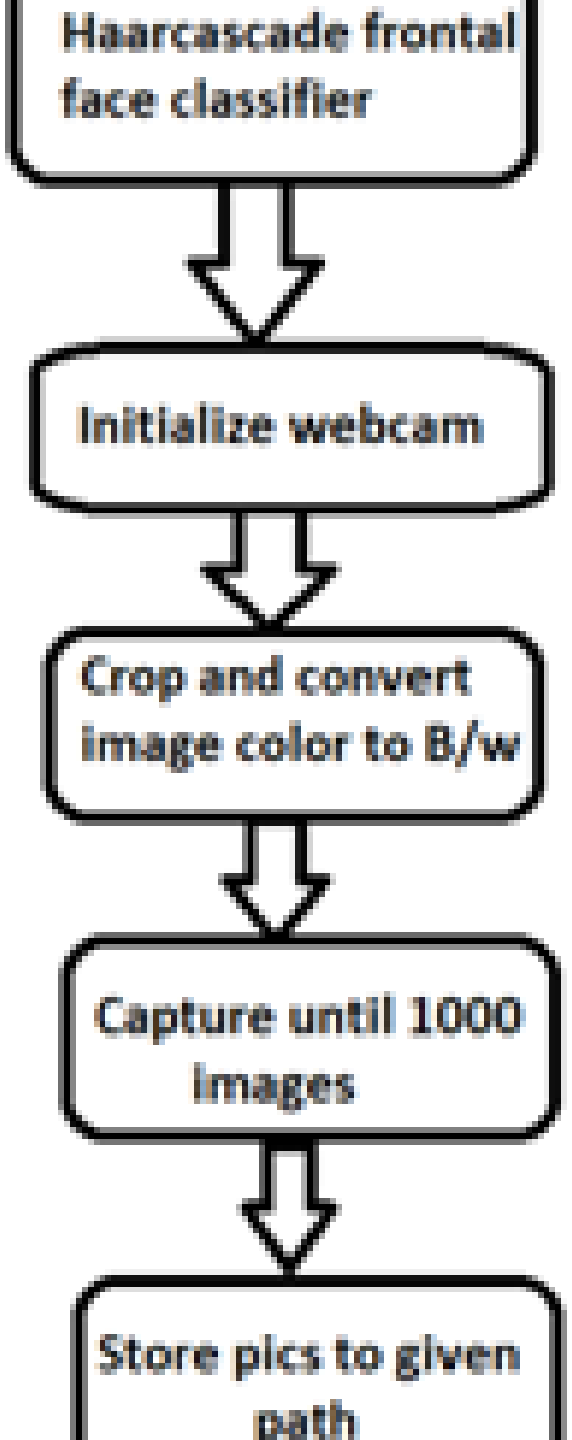
Anaconda :- Anaconda Enterprise is an enterprise-ready, secure, and scalable data science platform that empowers teams to govern data science assets, collaborate, and deploy data science projects.

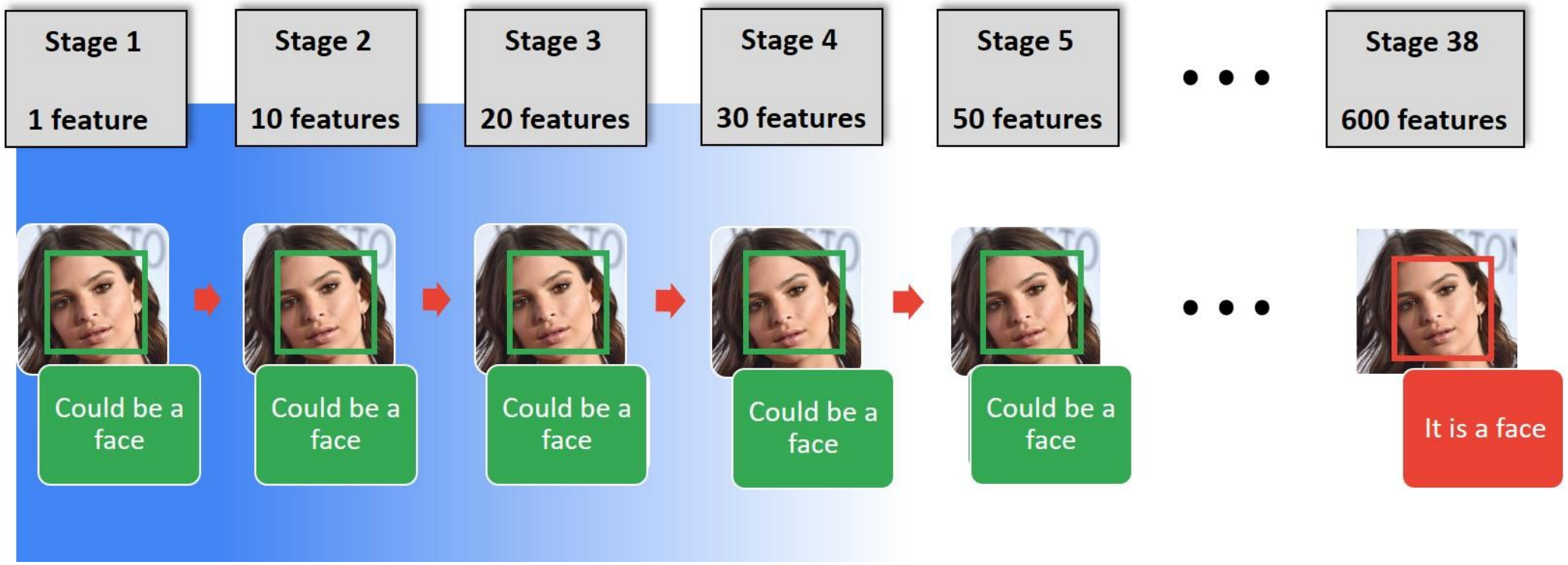
HARDWARE USED

A WINDOWS SYSTEM WITH WEBCAM IN IT.

METHODOLOGY

WORKFLOW OF MODULE 1





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

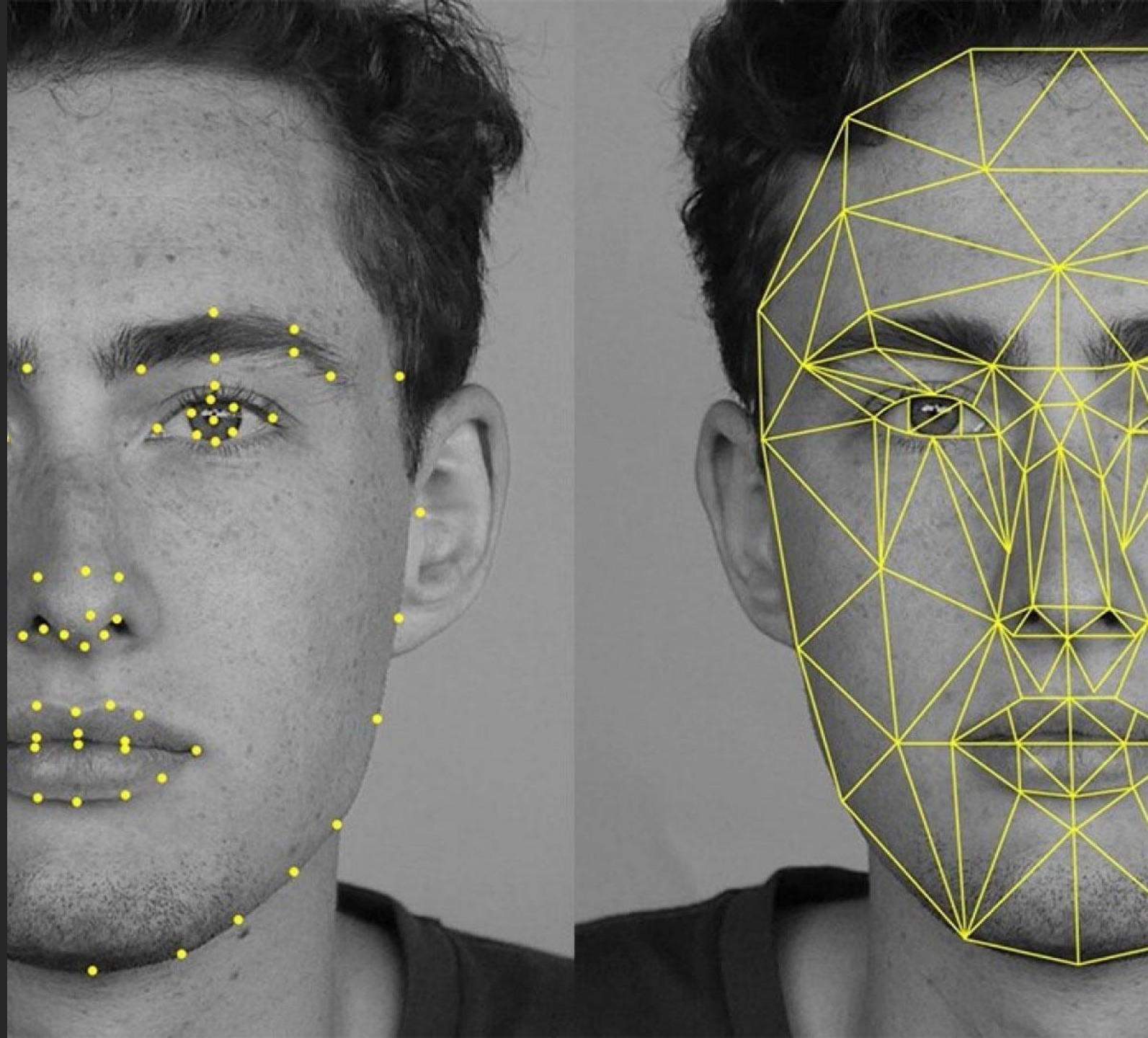
2. FACE AND EYE DETECTION
using haarcascade_eye.xml

3. VEHICLE DETECTION FROM STREAMING VIDEO using haarcascade_car.xml

4. PEDESTRIAN DETECTION FROM STREAMING VIDEO
using haarcascade_fullbody.xml

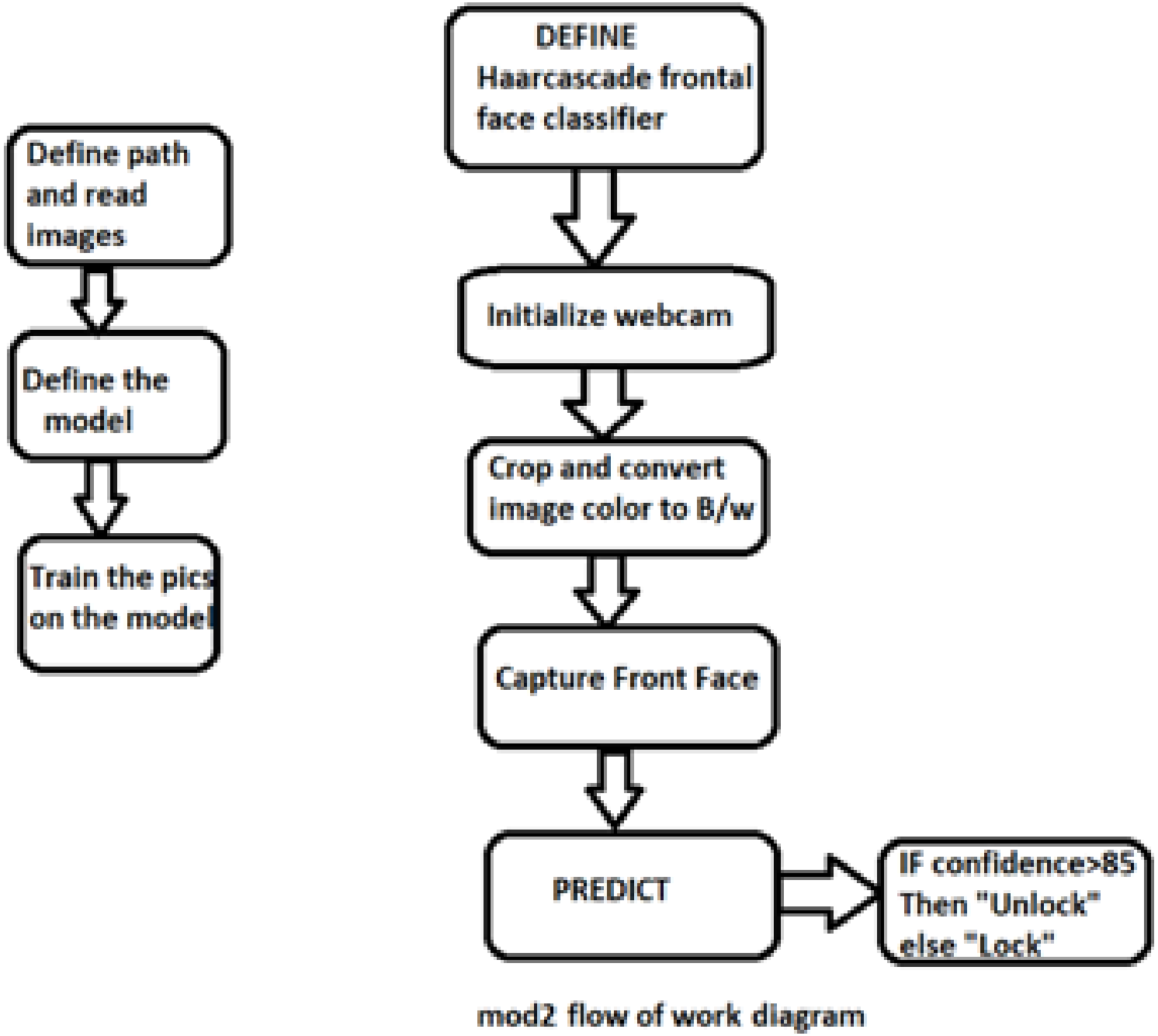
HARCASCADE FRONTAL FACE CLASSIFIER

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



METHODOLOGY

WORKFLOW OF MODULE 2



mod2 flow of work diagram

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 ALGORITHM

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.

Neighbors: 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.

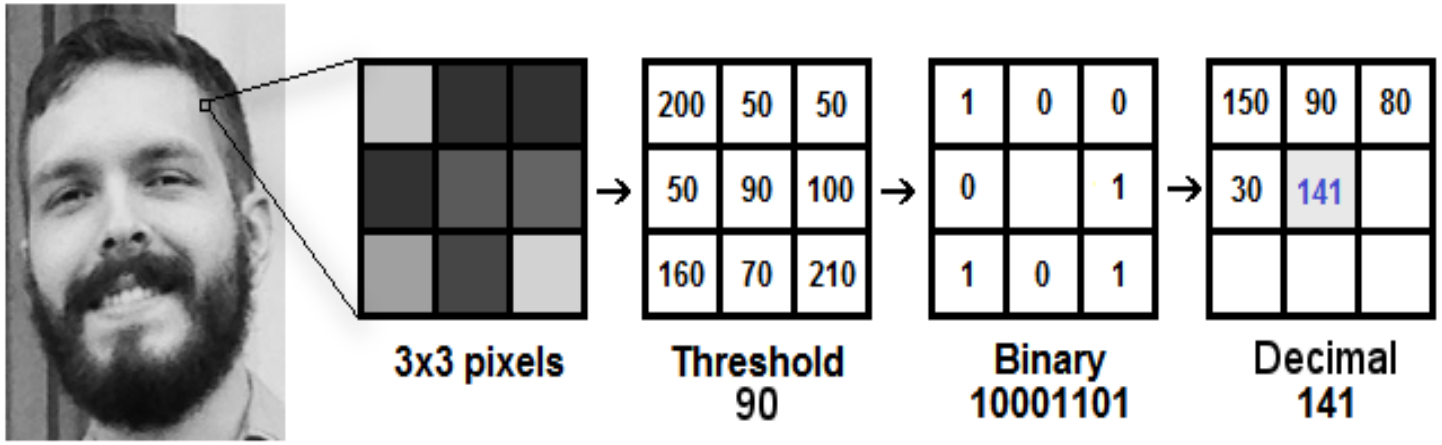
Grid Y: the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.

TRAINING THE LBPH 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.

APPLYING THE 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**.



APPLYING LBPH---CONTD--------

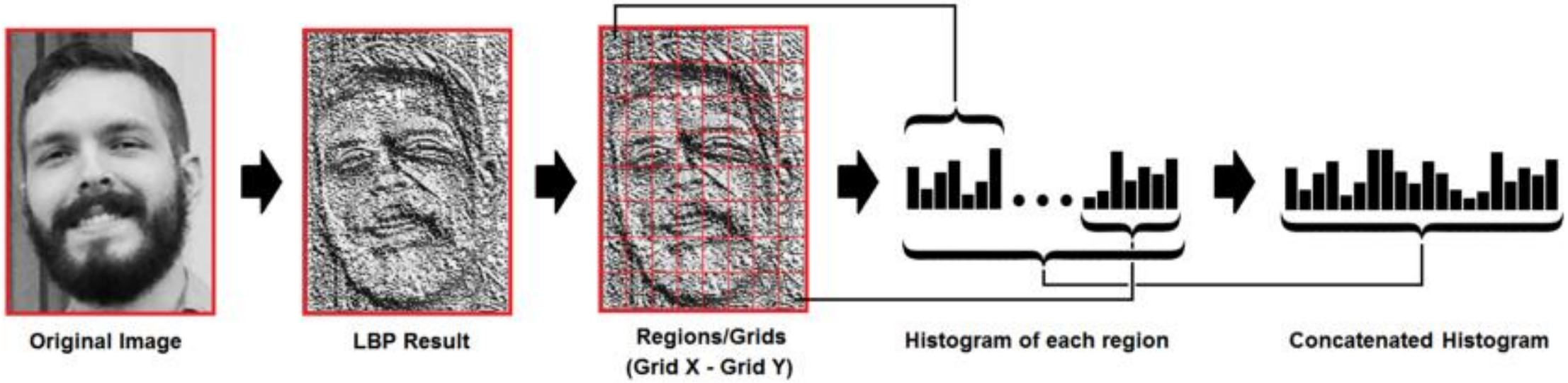
we need take the central value of the matrix of pixel values to be used as the threshold.

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).

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.



EXTRACTING HISTOGRAMS IN LBPH

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

FACE RECOGNITION BY LBPH

1. 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.

2. We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: euclidean distance, chi-square, absolute value, etc.

3. 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

```python
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 = 'E:/apa/pics/' + str(count) + '.jpg'
        cv2.imwrite(file_name_path, face)

        # Put count on images and display live count
```

SNAPSHOTS OF CODE

Module 1

```python
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 = 'E:/apa/pics/'
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_d

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, []
```

```python
# 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)
```

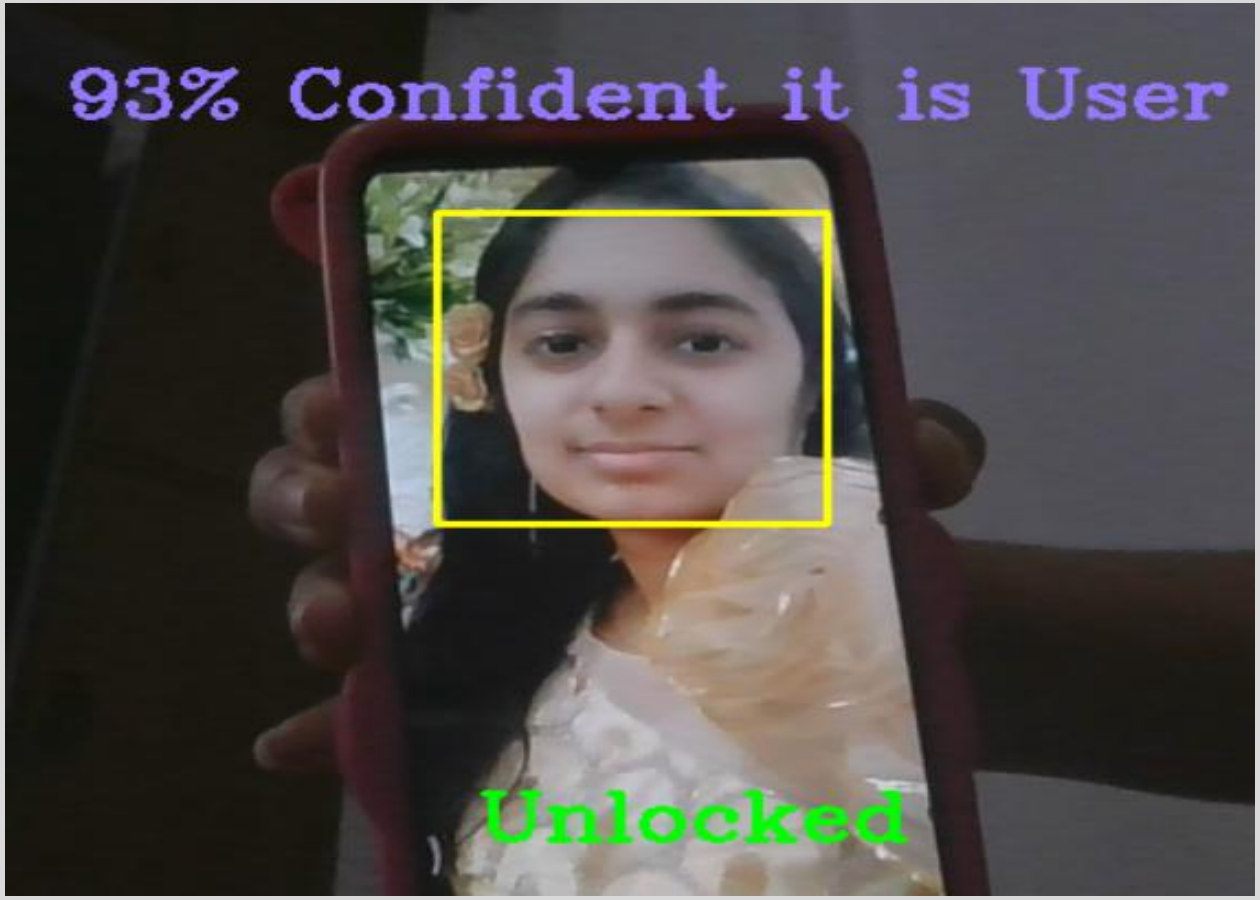
```python
    except:
        cv2.putText(image, "No Face Found", (220, 120) , cv2.FONT_HERSHEY_COMPLEX, 1, (0,0,255), 2)
        cv2.putText(image, "Locked", (250, 450), cv2.FONT_HERSHEY_COMPLEX, 1, (0,0,255), 2)
        cv2.imshow('Face Recognition', image )
        pass

    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
```

SNAPSHOTS

Module 2

SCREENSHOTS

This screenshot shows capturing 1000 images of the user to be used as dataset



RESULTS AND OUTCOMES

This snapshot shows the application being unlocked because the confidence is greater than 85%

References

https://docs.opencv.org/3.4/df/d25/classcv_1_1face_1_1LBPHFaceRecognizer.html

Facelock: familiarity-based graphical authentication

Research article

Human-Computer Interaction

Psychiatry and Psychology

Rob Jenkins[1], Jane L. McLachlan[2], Karen Renaud[3]

Published June 24, 2014


THANK YOU