MINOR PROJECT REPORT

REAL-TIME ATTENDANCE AND ENGAGEMENT DETECTION IN CLASSROOMS USING DEEP LEARNING AND COMPUTER VISION

Submitted in partial fulfilment of the requirements for the award of the degree of

B. TECH IN COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Karan Malik

02216403218

B.Tech CSE 2018-22

Rigved Alankar

40116403218

B.Tech CSE 2018-22

Submitted to:

Dr. Ruchi Sehrawat

Assistant Professor

USICT, New Delhi

UNIVERSITY SCHOOL OF INFORMATION, COMMUNICATION AND TECHNOLOGY

GURU GOBIND SINGH INDRAPRASTHA UNIVERSITY

NEW DELHI, 110078

Index

S No.	Section	Page Number
1	Mentor Approval	3
2	Abstract	4
3	Introduction	4
4	Problem Statement	4
5	Methodology and Algorithms Formulated	5
6	Project Workflow	9
7	Dataflow Diagrams	9
8	Hardware and Software Requirements	10
9	Working and Code	10
10	References	20

Mentor Approval

2 Attachments

Ruchi Sehrawat
to me

Approved for further submission.

Thanks a lot. | Noted with thanks. | Thank you!

Dr. Ruchi Sehrawat

Assistant Professor, USICT, New Delhi

Abstract

For an effective classroom environment, it becomes important to track the activities and the state of the students taking the lecture. This becomes increasingly important owing to the difficulties faced by students and teachers in the transition from offline to online teaching due to the pandemic, followed by the gradual reinstating of offline teaching. This project entails the usage of deep learning algorithms and computer vision techniques including facial detection and recognition for real-time attendance of the students in both offline and online settings. Moreover, engagement detection practices, such as drowsiness detection and gaze detection to maintain the sanctity of a classroom have also been proposed.

Introduction

Studies show that student's attendance has a strong positive correlation with their performance in classrooms [1], however, mid-class attendance along with factors such as inattentiveness of students often disturb a classroom. A considerable part of each lecture is spent recording students' attendance and our proposed system circumvents this limitation and allows for a more productive learning environment.

We plan to create an Artificial Intelligence based system that uses the face recognition algorithm *Local Binary Patterns Histogram* [2], to detect faces in a classroom setting – both offline and virtually. The student's presence/absence will then be updated in this database depending on it

Apart from real-time attendance, we also plan to incorporate student engagement detection in the classroom – using drowsiness detection, gaze detection and mobile-phone detection. For drowsiness detection, we aim to use facial landmark detection to detect eyes' coordinates, which can be used to compute the metric – Eye Aspect Ratio to tell us whether the eye is open or closed.

Testing of the system has been done on both, freely available datasets [3] as well as real-world data which we our system collects using webcam live streams.

As for the project timeline, we plan to incorporate real-time attendance and drowsiness detection in this semester as a part of the minor project and then extend the project by adding gaze and mobile phone detection, along with creation of a web interface for our major project in the next semester.

Problem Statement

Real-time Attendance and Engagement Detection in Classrooms using Deep Learning and Computer Vision

Methodology and Algorithms Formulated

Smart Attendance System

The architecture of the Smart Attendance System is as follows:

1. A camera – webcam or CCTV should be installed in front of the students so that it can capture the face of the student.
2. After the image has been captured; the captured image is transferred into the system as an input.
3. To circumvent the variability in the brightness of the images received, the images are converted to grayscale.
4. Futher, OpenCV's haarcascade model is used to detect faces and then extracting these facial images for the face recognition systems.
5. If the user is not already registered, they can at this point use the system to capture images to be used for recognition and register themselves with our system.
6. In the next step, the LBPH algorithm has been used to recognise faces of the registered users.
7. Finally, the attendance of the users detected i.e., their name, ID and the current date and time are exported and saved as a CSV file.

Local Binary Patterns Histogram (LBPH) Algorithm

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. When LBP is joined with histograms of oriented gradients (HOG) descriptor, it improves the efficiency and accuracy considerably for some datasets. Moreover, utilizing LBP joined with histograms allows us to represent facial images using simple data vectors, thus making it more suitable for face detection tasks.

The LBPH algorithm works in 5 steps as given below:

a) Parameters: LBPH uses 4 parameters:

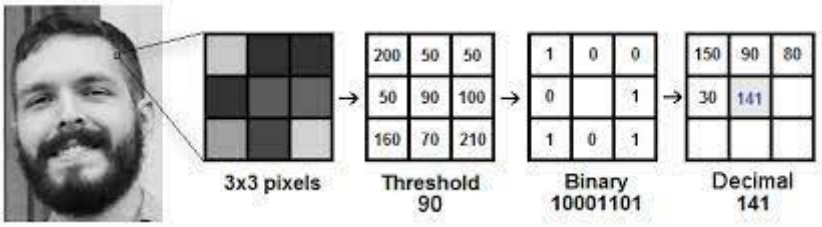
- Radius: the radius is used to create a local binary pattern and represents the radius around the centre pixel. The frequency is set to 1.
- Neighbours: the number of sample points to form the circular binary pattern. Higher the number of neighbours, the higher the computational cost. The frequency has been set to 8.
- Grid X: the number of cells in the horizontal direction. Using larger values for Grid X i.e., a more precise and finer grid allows higher dimensionality for the resulting vector. The frequency is set to 8.
- Grid Y - the number of cells in the vertical direction. Using larger values for Grid Y i.e., a more precise and finer grid allows higher dimensionality for the resulting vector. The frequency is set to 8.

b) Creating the dataset

The first step is to train the model. To do so, we use a database that contains images of people who are to be recognized i.e., our training data. Next, an identifier, such as a name or an identification number, is added to all the images to allow the algorithm to identify the input image. Multiple photos of the same person must have the same ID.

c) Applying the LBP operation

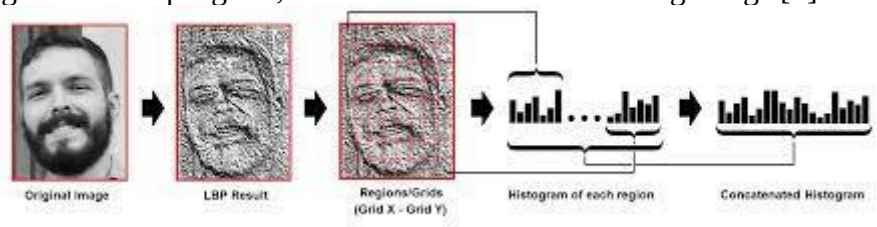
Next thing is to create an intermediate image that highlights the facial features of the original image. To do so, the algorithm law uses a sliding window which varies with change in the the parameters – radius and neighbours.



- We have a grayscale face image [5].
- Get a 3x3 pixel window from the image, with each number in the matrix showing the intensity of the corresponding pixel from 0-255.
- Find the central value of the matrix and use it as a threshold.
- Reassign the values of the remaining 8 neighbours – value=1 for pixels with equal or higher intensity than the threshold and value=0 for the others.
- Now, the matrix will contain only binary values. Concatenate the 8 binary values together to form a new 8-bit binary value. (e.g. 10001101).
- After that, we convert this binary number to a decimal value and set it to the center value of the matrix, which is a pixel from the first image.
- At the end of this process (LBP process), we have a new image that better represents the features of the original image.

d) Recording Histograms:

Now, using the image created in the last step, we can use the Grid X and Grid Y parameters to split the image into multiple grids, as can be seen in the following image [5]:



- Since we have a grayscale image, each histogram (from each grid) will only contain 256 (0 ~ 255) positions representing each pixel intensity.
- Further, a larger histogram is created by concatenating each histogram. If we assume we have 8x8 grids, we will have 8x8x256 = 16.384 positions in the final histogram. The final histogram represents the features of the first image.

e) Performing face recognition:

At this point, the algorithm is already trained. Each created histogram is used to represent each image from the training database. Whenever an image is input to the system, the following steps are retraced and a histogram is created for the given image.

To find the most similar image from the database to the input image, we just need to compare the two histograms and return the image with the nearest histogram.

We can use various methods to compare histograms (calculating the distance between two histograms), for example, Euclidean distance, square-chi, total value, etc. In our implementation, we use the Euclidean distance based on the following formula:

$$D = \sqrt{\sum_{i=1}^{n} (\text{hist } 1_i - \text{hist } 2_i)^2}$$

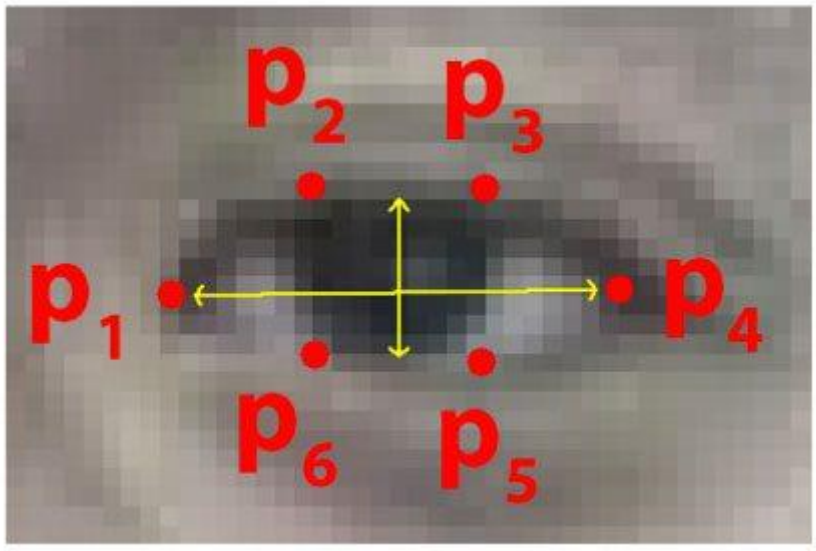
Distance Between two histograms

So, the calculation output is an ID from a picture with a close-by histogram. The calculated distance can be utilized as a measure of 'confidence'.

Detecting drowsiness

Using facial landmark detection to localize important regions of the face, including eyes, eyebrows, nose, ears, and mouth allows us the leverage to extract *specific facial structures* by knowing the indexes of the particular face parts: In terms of drowsiness detection, we are only interested in two sets of facial structures — *the eyes.*

Each eye is represented by 6 *(x, y)*-coordinates, starting at the left-corner of the eye and then working clockwise around the remainder of the region:



The 6 facial landmarks associated with the eye.

Based on the work by Soukupová and Čech in their 2016 paper, *Real-Time Eye Blink Detection using Facial Landmarks*,[4] we can then derive an equation that reflects the relation between the width and height of the eye, called the *eye aspect ratio* (EAR):

$$\text{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2\|p_1 - p_4\|}$$

The eye aspect ratio equation.

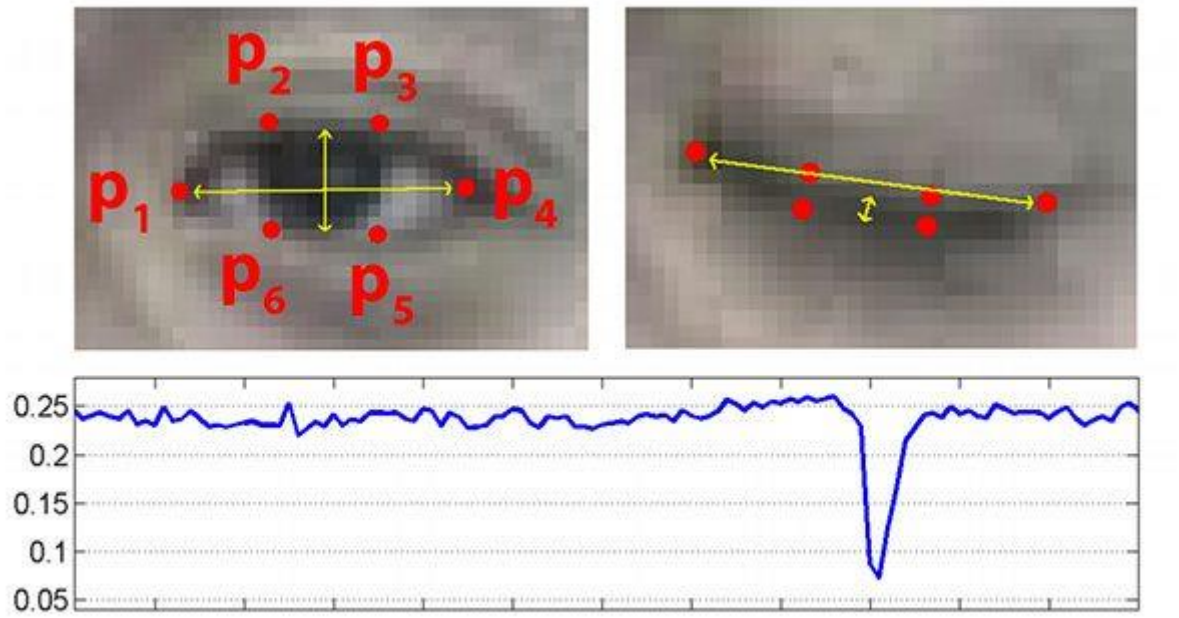
Where *p1, …, p6* are 2D facial landmark locations.

The numerator of this equation computes the distance between the vertical eye landmarks while the denominator computes the distance between horizontal eye landmarks, weighing the denominator appropriately since there is only *one* set of horizontal points but *two* sets of vertical points.

The eye aspect ratio is approximately constant while the eye is open but will rapidly fall to zero when a blink is taking place.

Using this simple equation, we can *avoid image processing techniques* and simply rely on the *ratio of eye landmark distances* to determine if a person is blinking.

To make this clear, consider the following figure from Soukupová and Čech:



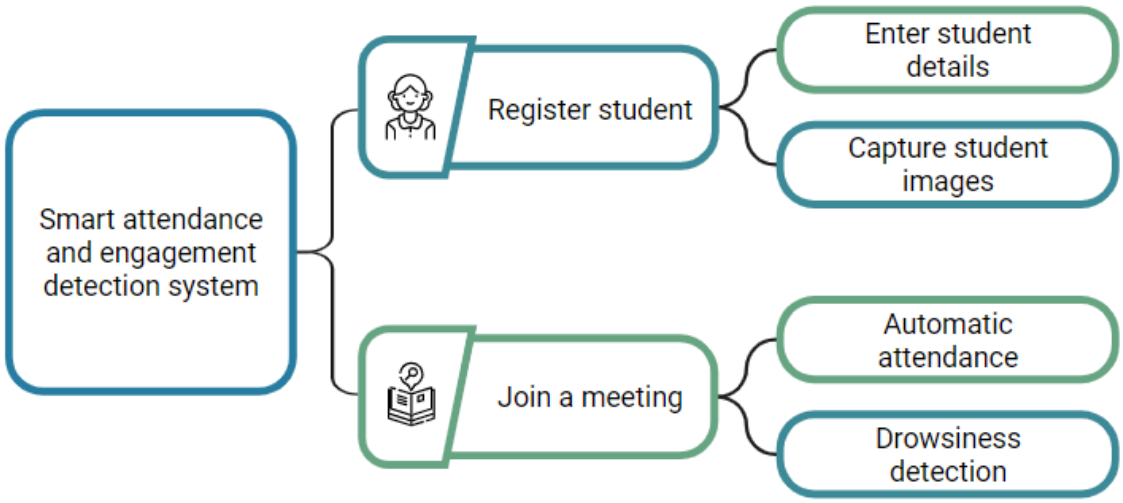
Top-left: A visualization of eye landmarks when then the eye is open. Top-right: Eye landmarks when the eye is closed. Bottom: Plotting the eye aspect ratio over time. The dip in the eye aspect ratio indicates a blink

On the *top-left* we have an eye that is fully open — the eye aspect ratio here would be large(r) and relatively constant over time.

However, once the person blinks (*top-right*) the eye aspect ratio decreases dramatically, approaching zero.

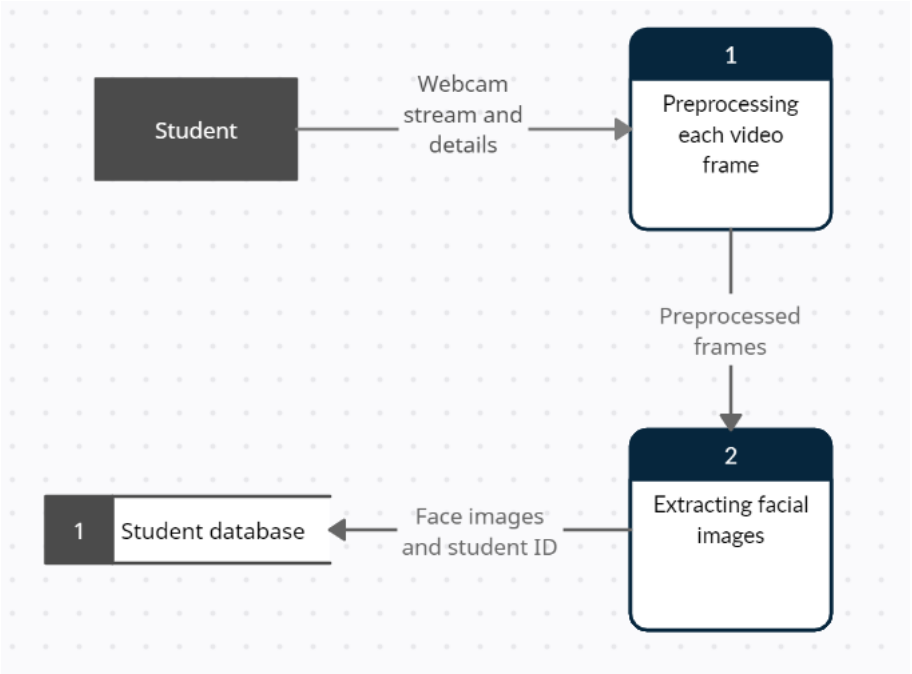
The *bottom* figure plots a graph of the eye aspect ratio over time for a video clip. As we can see, the eye aspect ratio is constant, then rapidly drops close to zero, then increases again, indicating a single blink has taken place.

Project Workflow

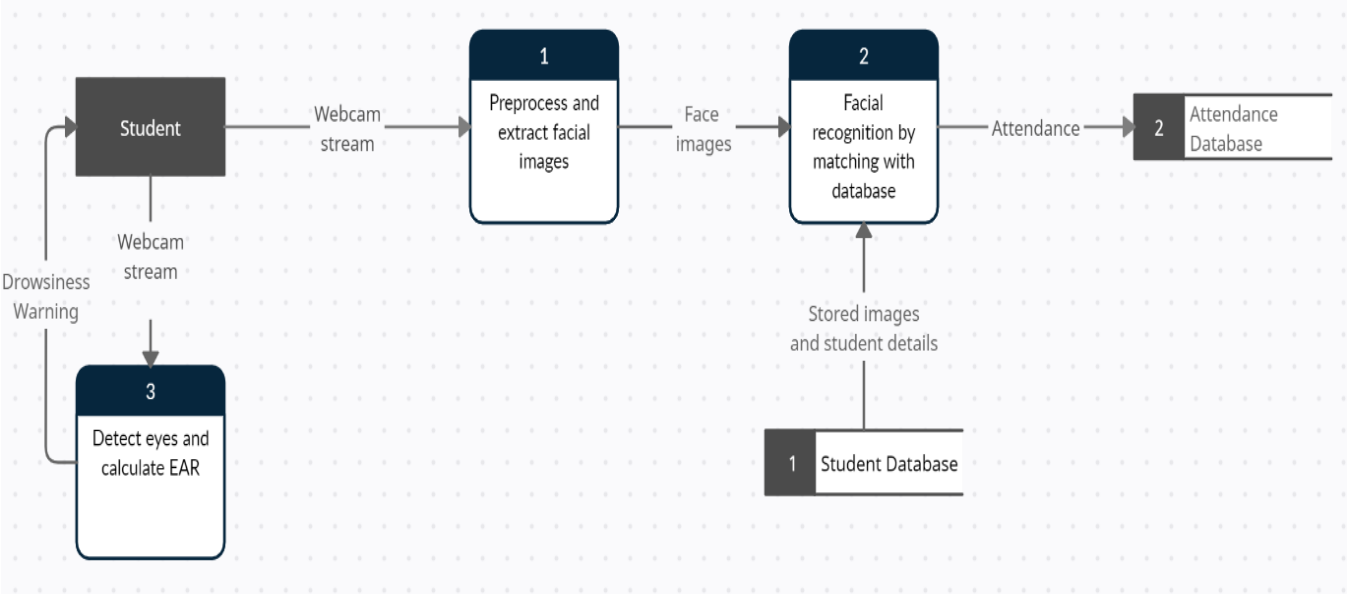


Dataflow Diagrams

Registering a student



Automatic attendance and drowsiness detection



Hardware and Software Requirements

Hardware Requirements:

1. CCTV cameras - for capturing students' images to be used for attendance and engagement detection
2. Webcam – to accomplish the same tasks in an online setting.

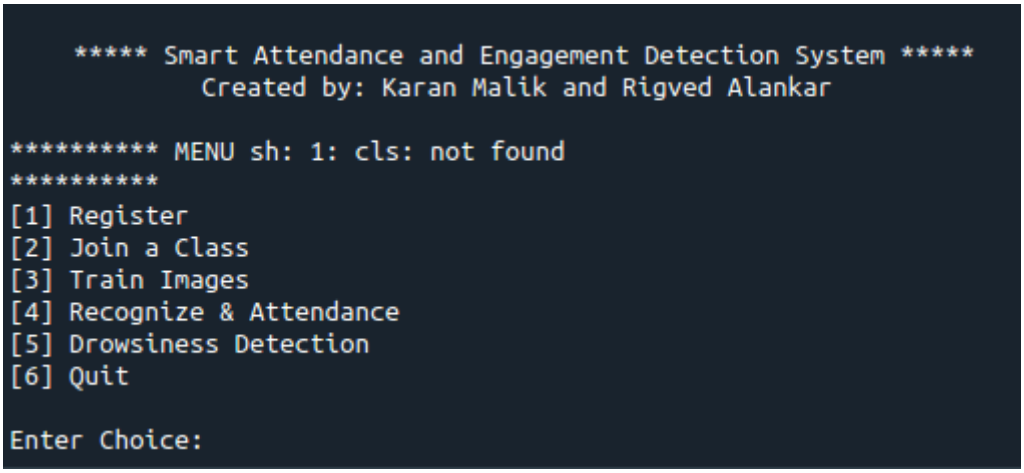
Software Requirements:

1. Programming language to be used: Python (for developing the system) and HTML/CSS (for developing the web interface)
2. Frameworks: Scikit-Learn, Tensorflow, Keras, OpenCV, NumPy etc.
3. Hosting the system on local host: Flask Web Development

Working and Code

In our model, we will be providing the users with the options to choose a given function from the above possible functions, the pseudocode for the same is given below:

1. Register
2. Join a Class
3. Train Images
4. Recognize & Attendance
5. Drowsiness Detection
6. Quit



The main menu of the working model

Pseudocode:

Begin

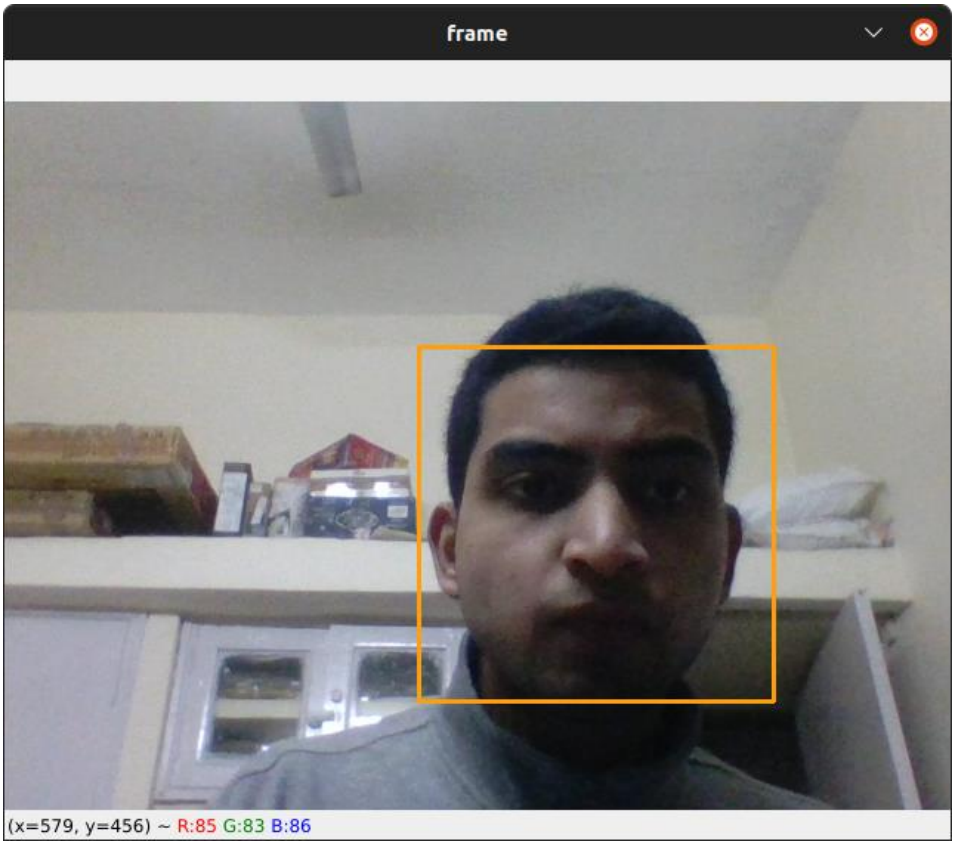
- if choice = 1
  call the function to register a new student and retrain the LBPH model
- elif choice = 2
  call the function which uses face recognition to identify the person and then mark attendance in a CSV file while also detecting for signs of drowsiness and warning the student if required
- elif choice = 3
  call the function which manually retrains images
- elif choice = 4
  call the function which uses face recognition to identify the person and then mark attendance in a CSV file
- elif choice = 5
  call the function that detects drowsiness
- else exit

End

Face recognition steps and working:

Step - 1: Finding all the Faces

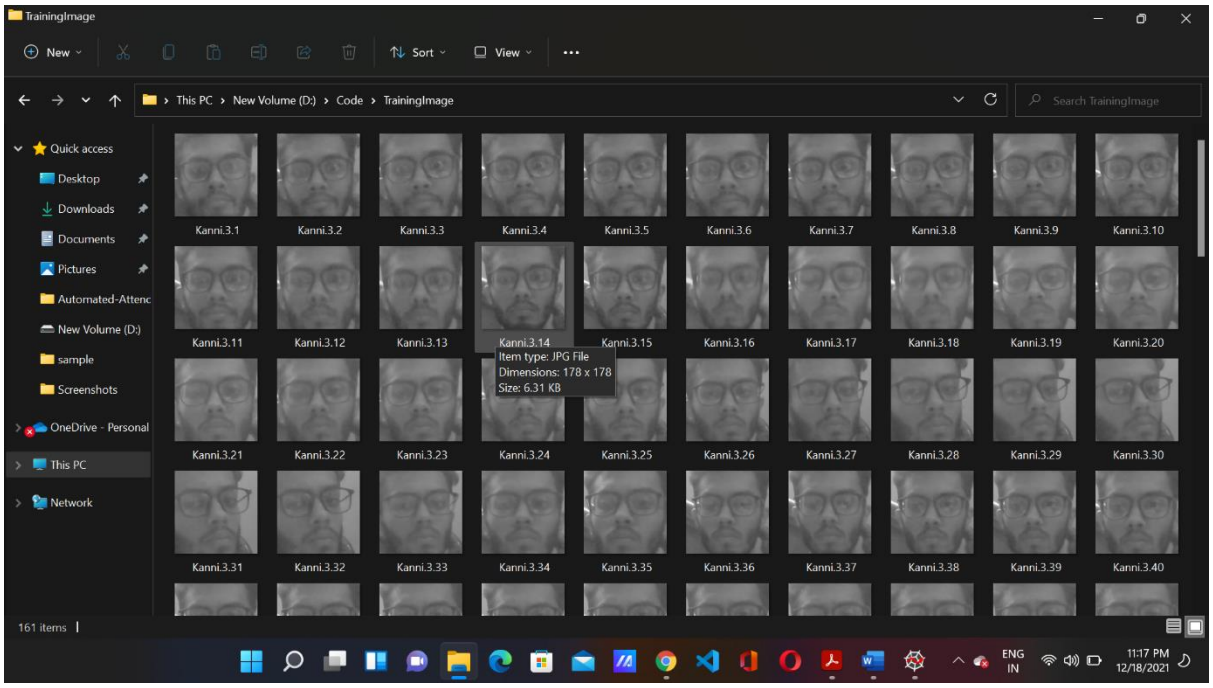
We start by looking out for faces in an image. Initially the image is converted to grayscale as colour data isn't required to identify or recognize faces and this allows our model to reduce its computational requirements. Then the pretrained model, haarcascade, provided by OpenCV has been used to detect all the faces in the input image.



Face getting detected by the model

Step 2: Training the model

In this step, the face image is encoded and the LBPH model trainer is used and the model is trained using the images in the dataset.



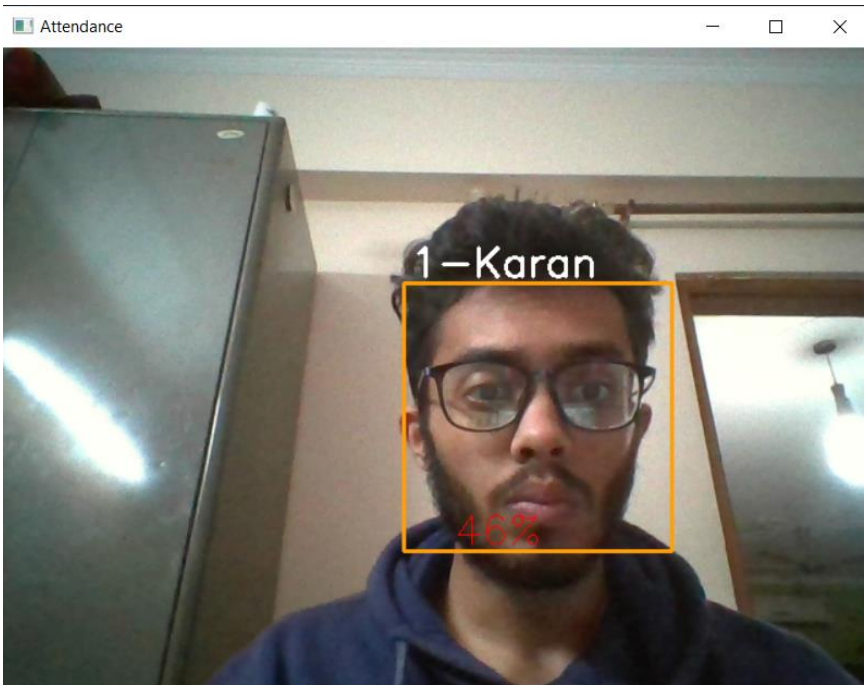
Snapshots captured while training the faces

From the figure above, it can be noticed that ~30-40 images get captured and stored in the database/ local storage. These images are then used to detect and recognize the person to mark the attendance.

```
def TrainImages():
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    harcascadePath = "haarcascade_default.xml"
    detector = cv2.CascadeClassifier(harcascadePath)
    faces, Id = getImagesAndLabels("TrainingImage")
    Thread(target = recognizer.train(faces, np.array(Id))).start()
    Thread(target = counter_img("TrainingImage")).start()
    recognizer.save("TrainingImageLabel"+os.sep+"Trainner.yml")
    print("All Images")
```

Function to train images in Python

Step 3: Compare against known faces
In this step, the model trained and stored in the previous step is used. An image which is input to the model is preprocessed and face images are extracted. Then the LBPH predict function is used to identify the facial image from the dataset closest to the image which is received.



Predicting face using LBPH model

Step 4: Marking attendance
The face is recognized, and the attendance is marked for the recognized person, along with a timestamp. This attendance is then stored in a CSV file

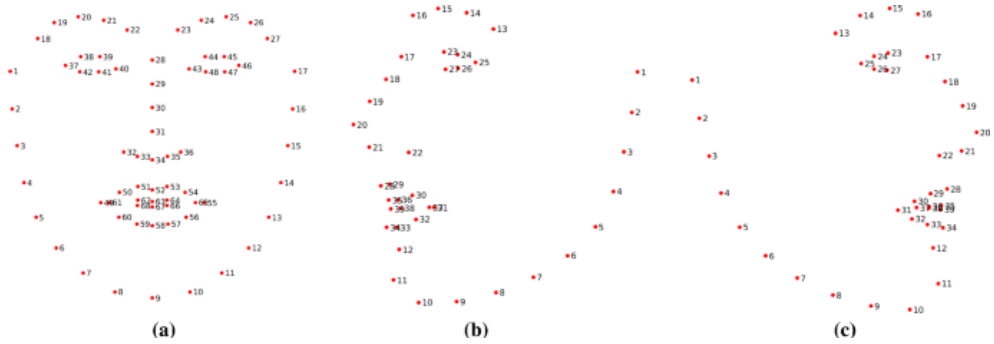
```
In [24]: recognize.recognize_attendence()
  Id    Name        Date       Time
0  1  Karan  2021-12-18  17:07:29
Attendance Successful
```

A	B	C	D
Id	Name	Date	Time
1	Karan	18-12-2021	17:07:29

Fig. Marked attendance and CSV file

Drowsiness detection steps and working:

Step - 1: Finding facial landmarks
For this, we use Dlib's face landmark estimation. The library allows us to return sixty-eight specific points (landmarks) including the upper part of the chin, the skin fringe of every eye, the inner fringe of the eyebrow, etc.



Front and side poses

This allows us to locate the eyes.

Step - 2: Calculating the eye aspect ratio
The above detected landmarks are then used to calculate the eye aspect ratio.

```
def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    C = dist.euclidean(eye[0], eye[3])
    ear = (A + B) / (2.0 * C)
    return ear
```

Calculating EAR

Step - 3: Detecting drowsiness
If the calculated EAR is below a given threshold, a warning is displayed on the screen.

Detecting drowsiness

Tasks Completed

We have created the models for real-time attendance and for detecting drowsiness from the captured images.

All our code is written in Python. Below are the important python codes which are the core of our implementation of this software engineering project.

1. Dataset: Where all the face images are saved.
2. main.py: Main program file to run the program.
3. train_image.py: Train the captured images and work on datasets.
4. recognize.py: To recognize and mark attendance
5. detect_drowsiness.py: To detect drowsiness in the captured images

a) main.py
This is the main python code for our project that includes functions from other files and has all the options which are allowed in the user interface. Once this file is run, the user gets to choose whatever action has to be taken place, say whether it is to capture an image or to train an image, etc. The main function calls the other functions and specifies options to the user.

```
import os
#import check_camera
import capture_image
import train_image
import recognize
import detect_drowsiness

def title_bar():
    os.system('cls')
    print("\n\n\t***** Smart Attendance and Engagement Detection System *****")
    print("\t\t\tCreated by: Karan Malik and Rigved Alankar")


def mainMenu():
    title_bar()
    print()
    print(10 * "*", "MENU", 10 * "*")
    print("[1] Register")
    print("[2] Join a Class")
    print("[3] Train Images")
    print("[4] Recognize & Attendance")
    print("[5] Drowsiness Detection")
    print("[6] Quit")
    while True:
        try:
            choice = int(input("Enter Choice: "))
            if choice == 1:
                CaptureFaces()
                Trainimages()
                break
            elif choice == 2:
                CaptureFaces()
                break
            elif choice == 3:
                Trainimages()
                break
```

```
            elif choice == 4:
                recognizeFaces()
                break
            elif choice == 5:
                drowsinessDetect()
                break
            elif choice == 6:
                print("Thank You")
                break
            else:
                print("Invalid Choice. Enter 1-4")
                mainMenu()
        except ValueError:
            print("Invalid Choice. Enter 1-4\n Try Again")
    exit



def CaptureFaces():
    capture_image.takeImages()
    print('\nStudent registered\n')


def Trainimages():
    train_image.TrainImages()
    print('**Model retrained**')
    key = input("Enter any key to return main menu")
    mainMenu()

def recognizeFaces():
    recognize.recognize_attendence()
    key = input("Enter any key to return main menu")
    mainMenu()
```

```
def drowsinessDetect():
    detect_drowsiness.detect_drowsy()
    key = input("Enter any key to return main menu")
    mainMenu()


mainMenu()
```

b) train_image.py

In this step, the image gets trained with database images and machine learning techniques. This is done by making use of a haarcascade classifier. Once the images are trained and the model is ready for the next step, the "image trained" message appears on the screen.

```
import os
import time
import cv2
import numpy as np
from PIL import Image
from threading import Thread


def getImagesAndLabels(path):
    imagePaths = [os.path.join(path, f) for f in os.listdir(path)]
    faces = []
    Ids = []
    for imagePath in imagePaths:
        pilImage = Image.open(imagePath).convert('L')
        imageNp = np.array(pilImage, 'uint8')
        Id = int(os.path.split(imagePath)[-1].split(".")[1])
        faces.append(imageNp)
        Ids.append(Id)
    return faces, Ids


def TrainImages():
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    harcascadePath = "haarcascade_default.xml"
    detector = cv2.CascadeClassifier(harcascadePath)
    faces, Id = getImagesAndLabels("TrainingImage")
    Thread(target = recognizer.train(faces, np.array(Id))).start()
    Thread(target = counter_img("TrainingImage")).start()
    recognizer.save("TrainingImageLabel"+os.sep+"Trainner.yml")
    print("All Images")


def counter_img(path):
    imgcounter = 1
    imagePaths = [os.path.join(path, f) for f in os.listdir(path)]
    for imagePath in imagePaths:
        print(str(imgcounter) + " Images Trained", end="\r")
        time.sleep(0.008)
        imgcounter += 1
```

c) recognize.py

In this step, the faces are recognized with the help of the LBPH Face recognizer function. Along with this, the time and date also get recorded. If the function works successfully then the attendance is marked for the recognized face.

```
def recognize_attendence():
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    recognizer.read("TrainingImageLabel"+os.sep+"Trainner.yml")
    harcascadePath = "haarcascade_default.xml"
    faceCascade = cv2.CascadeClassifier(harcascadePath)
    df = pd.read_csv("StudentDetails"+os.sep+"StudentDetails.csv",header=None)
    df.columns=['Id','Name']
    font = cv2.FONT_HERSHEY_SIMPLEX
    col_names = ['Id', 'Name', 'Date', 'Time']
    attendance = pd.DataFrame(columns=col_names)

    cam = cv2.VideoCapture(0, cv2.CAP_DSHOW)
    cam.set(3, 640)
    cam.set(4, 480)
    minW = 0.1 * cam.get(3)
    minH = 0.1 * cam.get(4)

    while True:
        ret, im = cam.read()
        gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
        faces = faceCascade.detectMultiScale(gray, 1.2, 5,
                minSize = (int(minW), int(minH)),flags = cv2.CASCADE_SCALE_IMAGE)
        for(x, y, w, h) in faces:
            cv2.rectangle(im, (x, y), (x+w, y+h), (10, 159, 255), 2)
            Id, conf = recognizer.predict(gray[y:y+h, x:x+w])
            if conf < 100:
                #print(df)
                aa = df.loc[df['Id'] == Id]['Name'].values
                confstr = "  {0}%".format(round(100 - conf))
```

```
            if conf < 100:
                #print(df)
                aa = df.loc[df['Id'] == Id]['Name'].values
                confstr = "  {0}%".format(round(100 - conf))
                tt = str(Id)+"-"+aa
            else:
                Id = '  Unknown  '
                tt = str(Id)
                confstr = "  {0}%".format(round(100 - conf))

            if (conf) > 55:
                ts = time.time()
                date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
                timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
                aa = str(aa)[2:-2]
                attendance.loc[len(attendance)] = [Id, aa, date, timeStamp]

            tt = str(tt)[2:-2]
            if(100-conf) > 67:
                tt = tt + " [Pass]"
                cv2.putText(im, str(tt), (x+5,y-5), font, 1, (255, 255, 255), 2)
            else:
                cv2.putText(im, str(tt), (x + 5, y - 5), font, 1, (255, 255, 255), 2)

            if (100-conf) > 67:
                cv2.putText(im, str(confstr), (x + 5, y + h - 5), font,1, (0, 255, 0),1 )
            elif (100-conf) > 50:
                cv2.putText(im, str(confstr), (x + 5, y + h - 5), font, 1, (0, 255, 255), 1)
            else:
                cv2.putText(im, str(confstr), (x + 5, y + h - 5), font, 1, (0, 0, 255), 1)


        attendance = attendance.drop_duplicates(subset=['Id'], keep='first')

        cv2.imshow('Attendance', im)
        if (cv2.waitKey(1) == ord('q')):
            break

    print(attendance)
    ts = time.time()
    date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
```

```
print(attendance)
ts = time.time()
date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
timeStamp = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
Hour, Minute, Second = timeStamp.split(":")
fileName = "Attendance"+os.sep+"Attendance_"+date+"_"+Hour+"-"+Minute+"-"+Second+".csv"
attendance.to_csv(fileName, index=False)
print("Attendance Successful")
cam.release()
cv2.destroyAllWindows()
```

d) detect_drowsiness.py

```
from scipy.spatial import distance as dist
from imutils.video import VideoStream
from imutils import face_utils
from threading import Thread
import numpy as np
import playsound
import argparse
import imutils
import time
import dlib
import cv2


def eye_aspect_ratio(eye):
    A = dist.euclidean(eye[1], eye[5])
    B = dist.euclidean(eye[2], eye[4])
    C = dist.euclidean(eye[0], eye[3])
    ear = (A + B) / (2.0 * C)
    return ear
```

```
def detect_drowsy():

    EYE_AR_THRESH = 0.3
    EYE_AR_CONSEC_FRAMES = 48

    COUNTER = 0
    detector = dlib.get_frontal_face_detector()
    predictor = dlib.shape_predictor('shape_predictor_68_face_landmarks.dat')

    (lStart, lEnd) = face_utils.FACIAL_LANDMARKS_IDXS["left_eye"]
    (rStart, rEnd) = face_utils.FACIAL_LANDMARKS_IDXS["right_eye"]

    vs = cv2.VideoCapture(0, cv2.CAP_DSHOW)
    time.sleep(1.0)
```

```
while True:
    _,frame = vs.read()
    frame = imutils.resize(frame, width=450)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    rects = detector(gray, 0)

    for rect in rects:
        shape = predictor(gray, rect)
        shape = face_utils.shape_to_np(shape)
        leftEye = shape[lStart:lEnd]
        rightEye = shape[rStart:rEnd]
        leftEAR = eye_aspect_ratio(leftEye)
        rightEAR = eye_aspect_ratio(rightEye)

        ear = (leftEAR + rightEAR) / 2.0

        leftEyeHull = cv2.convexHull(leftEye)
        rightEyeHull = cv2.convexHull(rightEye)
        cv2.drawContours(frame, [leftEyeHull], -1, (0, 255, 0), 1)
        cv2.drawContours(frame, [rightEyeHull], -1, (0, 255, 0), 1)

        if ear < EYE_AR_THRESH:
            COUNTER += 1

            if COUNTER >= EYE_AR_CONSEC_FRAMES:
                cv2.putText(frame, "DROWSINESS ALERT!", (10, 30),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
        else:
            COUNTER = 0

        cv2.putText(frame, "EAR: {:.2f}".format(ear), (300, 30),
            cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)

    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break
cv2.destroyAllWindows()
vs.release()
```

References

1. T. Godson '*Effects of Classroom Attendance and Learning Strategies on the Learning Outcome'* Journal of International Education in Business (2018).
2. 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.
3. P.E. Kekong, I.A. Ajah & U. Chidiebere, 'Real Time Drowsy Driver Monitoring and Detection System Using Deep Learning Based Behavioural Approach' International Journal of Computer Sciences and Engineering (2021).
4. Soukupová, Tereza and Jan Cech. "*Real-Time Eye Blink Detection using Facial Landmarks*." (2016).
5. https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b