EC3093D

DIGITAL SIGNAL PROCESSING LAB

DSP MINI PROJECT REPORT

NAME	ROLLNO	EMAIL
Kadagala Shiva Kumar	B200981EC	kadagala_b200981ec@nitc.ac.in
Kada Yogeswara Rao	B200980EC	kada_b200980ec@nitc.ac.in

FACE RECOGNITION -BASED – ATTENDANCE MANAGEMENT SYSTEM

INTRODUCTION:

Face recognition-based attendance management systems have gained immense popularity in various industries, educational institutions, and organizations. These systems employ sophisticated digital signal processing techniques to identify and verify the identity of individuals by analying their facial features.

A range of digital signal processing techniques are utilized in face recognition-based attendance management systems, each with its distinct advantages and limitations.

Haar Cascade Classifier: It is a machine learning object detection algorithm used to identify objects in an image or video. In this code, Haar Cascade Classifier is used to detect faces in images.

LBPH (Local Binary Patterns Histograms) Face Recognizer: It is a face recognition algorithm that extracts local binary patterns from an image and creates a histogram of them. LBPH face recognizer is used in this code to train the images and labels for the face recognition system.

Image Pre-processing Techniques: Before training the images, some image pre-processing techniques such as converting the image to grayscale, resizing, and converting to numpy array are used. These techniques help to extract features from images and prepare them for training.

Various digital signal processing techniques are employed in face recognition systems to improve their accuracy and robustness. These techniques include pre-processing, feature extraction, and classification. Pre-processing involves removing noise and enhancing the quality of the input image or video stream. Feature extraction involves identifying unique features of the face, such as the distance between the eyes, the shape of the nose, and the contour of the face. Classification involves using machine learning algorithms to identify a face and match it to a known identity.

Overall, the combination of these digital signal processing techniques enables face recognition systems to accurately and efficiently track attendance, making them a valuable tool for organizations of all sizes.

Modules used:

tkinter: Tkinter is the standard GUI (Graphical User Interface) package for Python. It provides a fast and easy way to create GUI applications that can run on any platform where Python is installed. It comes with a wide range of widgets and tools for building desktop applications, including buttons, labels, menus, text boxes, and more.

os: The os module in Python provides a way of interacting with the operating system. It allows you to perform various operations like create directories, delete files, get file attributes, and more.

cv2: OpenCV (Open Source Computer Vision Library) is a computer vision and machine learning software library. The cv2 module is a Python wrapper for the OpenCV library, which provides a wide range of image processing and computer vision functions, including object detection, facial recognition, and more.

shutil: The shutil module in Python provides a higher level interface for file operations. It provides functions for copying, moving, and deleting files and directories.

csv: The csv module in Python provides functionality to read from and write to CSV (Comma Separated Value) files. It allows you to easily store and exchange data between different applications.

numpy: NumPy is a Python library for working with arrays. It provides a high-performance multidimensional array object, and tools for working with these arrays. It is used in a wide range of applications, including scientific computing, data analysis, and machine learning.

PIL (Python Imaging Library): PIL is a library for working with images in Python. It provides tools for opening, manipulating, and saving images in a variety of formats.

pandas: Pandas is a library for working with data in Python. It provides tools for data analysis and manipulation, including data structures for efficiently storing and working with large datasets.

datetime: The datetime module in Python provides classes for working with dates and times. It allows you to perform various operations like creating a date object, formatting dates and times, and more.

time: The time module in Python provides various time-related functions. It allows you to perform operations like getting the current time, converting time between different formats, and more.

tkinter.font: The tkinter.font module provides functionality for working with fonts in Tkinter applications. It allows you to create font objects, set font properties, and more.

pyttsx3: pyttsx3 is a text-to-speech conversion library in Python. It provides a way of converting text to speech in a variety of voices and languages.

Subprocess: The subprocess module in Python allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. It provides a higher-level interface to working with processes than the os module. It provides a consistent interface to creating and interacting with additional processes, making it useful for tasks such as system administration, automation, and running external programs.

Requests: The requests module is a Python library used for making HTTP requests. It provides an easy-to-use interface for sending HTTP/1.1 requests with support for GET, POST, PUT, DELETE, and other HTTP methods, as well as handling cookies, redirects, and authentication.

Haar Cascade classifiers:

Haar Cascade classifiers are machine learning based approaches that detect objects in images. These classifiers have been trained to detect specific objects, such as faces, by extracting relevant features from sample images.
The Haar Cascade classifiers haarcascade_frontalface_alt.xml and haarcascade_frontalface_default.xml are pre-trained classifiers specifically designed to detect frontal faces. They are trained on a large number of positive and negative samples to accurately identify faces in an image.

The ***haarcascade_frontalface_alt.xml classifier*** is more accurate but slower than the ***haarcascade_frontalface_default.xml classifier.*** The haarcascade_frontalface_default.xml classifier, on the other hand, is faster but less accurate. Depending on the requirements of your face recognition project, you can choose one of these classifiers.
The advantages of using pre-trained Haar Cascade classifiers for face recognition include:

- Accuracy: The classifiers have been trained on a large number of positive and negative samples, making them highly accurate at detecting faces in an image.
- Speed: Since the classifiers are pre-trained, they can be used directly without the need for additional training, which saves time and resources.
- Availability: The classifiers are readily available as open-source software, making them easily accessible for developers.
- Flexibility: The classifiers can be used in a variety of applications, including face detection, tracking, and recognition.

The algorithm works by using a set of positive and negative sample images to train a classifier. Positive samples contain the object we want to detect (e.g. faces), while negative samples contain other images that do not contain the object.

The classifier learns to differentiate between the two sets by extracting features from each image, represented as Haar-like features. These features are calculated by determining the difference in intensity between adjacent rectangular regions in the image.

The Haar-like features are then used to train a machine learning algorithm, typically an AdaBoost classifier, which combines a series of weaker classifiers into a strong one. This strong classifier is then used to detect the object in new images by sliding a window over the image and comparing the Haar-like features within each window to the ones learned during training.

The algorithm works as follows: first, it selects a subset of the training data and trains a weak classifier on it. Then, it identifies the training examples that the classifier got wrong, and it increases their weights. The next weak classifier is then trained on the same subset of training data, but with the modified weights. This process is repeated several times, with each new classifier giving more weight to the examples that the previous classifiers got wrong.
Once all the weak classifiers have been trained, they are combined into a strong classifier by taking a weighted majority vote of their predictions.

If the features match those learned for the object, the classifier marks the region as a positive detection. The algorithm can be further optimized by using techniques such as image pyramid and sliding window with various scales, in order to improve the detection of objects at different sizes and locations.

```
detector = cv2.CascadeClassifier(haarcasecade_path)
```

We found this at:

https://github.com/opencv/opencv/tree/master/data/haarcascades

An OpenCv Public Repository.

LBPH (Local Binary Patterns Histograms) Face Recognizer:

LBPH (Local Binary Patterns Histograms) Face Recognizer is a popular algorithm used in face recognition systems. It works by dividing the face image into small regions and computing a histogram of Local Binary Patterns (LBP) for each region. LBP is a texture descriptor that summarizes the local structure of an image by comparing each pixel to its surrounding pixels.

In LBPH, for each pixel in an image, its 8 neighboring pixels are compared to it, and the result of each comparison is represented as a binary digit (0 or 1) based on whether the neighboring pixel is greater than or equal to the central pixel. This results in a binary pattern for each pixel, which can be used to compute a histogram of LBP values for the entire image or for smaller regions of the image.

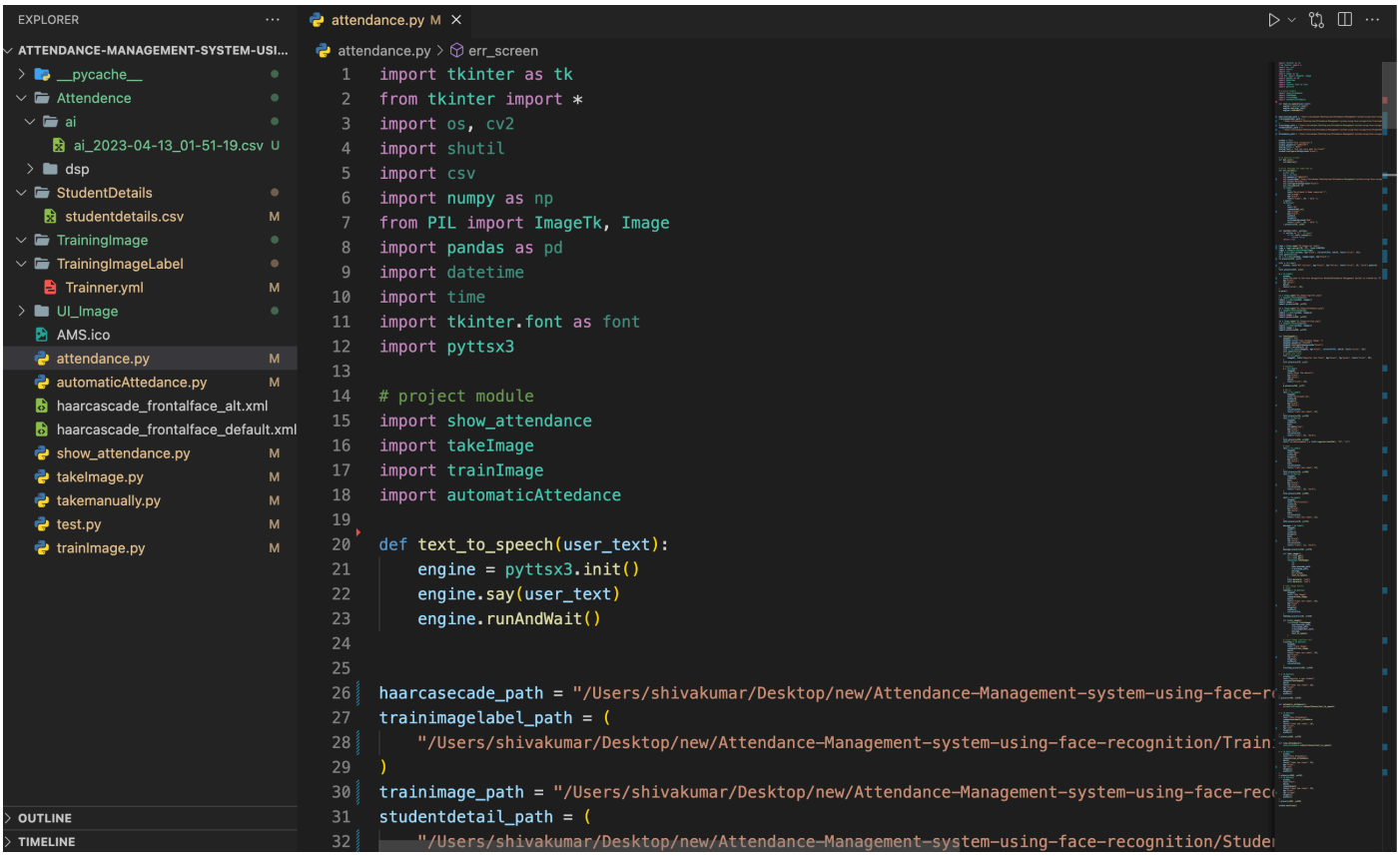
The LBPH algorithm extracts these histograms of LBP features from facial images, and uses them to train a machine learning model that can recognize faces. When recognizing a new face, the algorithm extracts LBP histograms from the input image, and compares them to the histograms of known faces in the training set to find the closest match.

LBPH is a simple and effective algorithm for face recognition, and is often used in real-world applications due to its good accuracy and speed.

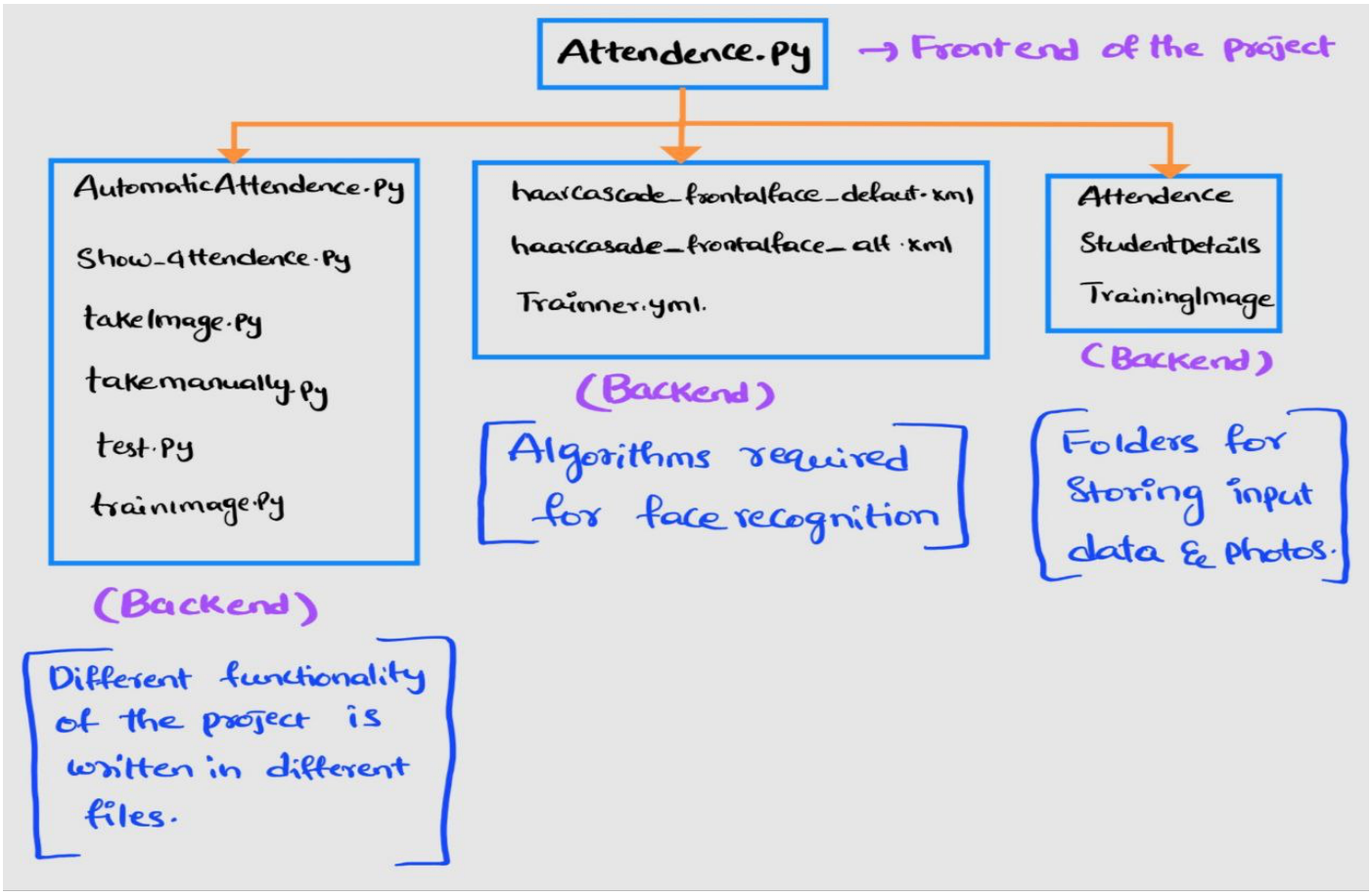
This face recognition algorithm is present in cv2 module and can be called using

```
recognizer = cv2.face.LBPHFaceRecognizer_create()
```

screenshot of all files in the project:



Project Organization:



Attendance.py file:

```
import tkinter as tk
from tkinter import *
import os, cv2
import shutil
import csv
import numpy as np
from PIL import ImageTk, Image
import pandas as pd
import datetime
import time
import tkinter.font as font
import pyttsx3

# project module
import show_attendance
import takeImage
import trainImage
import automaticAttedance

def text_to_speech(user_text):
    engine = pyttsx3.init()
    engine.say(user_text)
    engine.runAndWait()


haarcasecade_path = "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/haarcascade_frontalface_default.xml"
trainimagelabel_path = (
    "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/TrainingImageLabel/Trainner.yml"
)
trainimage_path = "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/TrainingImage"
studentdetail_path = (
    "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/StudentDetails/studentdetails.csv"
)
attendance_path = "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/Attendence"
```

```
window = Tk()
window.title("Face recognizer")
window.geometry("1280x720")
dialog_title = "QUIT"
dialog_text = "Are you sure want to close?"
window.configure(background="black")


# to destroy screen
def del_sc1():
    sc1.destroy()


# error message for name and no
def err_screen():
    global sc1
    sc1 = tk.Tk()
    sc1.geometry("400x110")
    sc1.iconbitmap("AMS.ico")
    sc1.title("Warning!!")
    sc1.configure(background="black")
    sc1.resizable(0, 0)
    tk.Label(
        sc1,
        text="Enrollment & Name required!!!",
        fg="orange",
        bg="black",
        font=("times", 20, " bold "),
    ).pack()
    tk.Button(
        sc1,
        text="OK",
        command=del_sc1,
        fg="orange",
        bg="black",
        width=9,
        height=1,
        activebackground="Red",
        font=("times", 20, " bold "),
    ).place(x=110, y=50)


def testVal(inStr, acttyp):
    if acttyp == "1":  # insert
        if not inStr.isdigit():
```

```
            return False
    return True


logo = Image.open("UI_Image/col.jpeg")
logo = logo.resize((50, 47), Image.LANCZOS)
logo1 = ImageTk.PhotoImage(logo)
titl = tk.Label(window, bg="black", relief=RIDGE, bd=10, font=("arial",
35))
titl.pack(fill=X)
l1 = tk.Label(window, image=logo1, bg="black",)
l1.place(x=470, y=15)

titl = tk.Label(
    window, text="NIT Calicut", bg="black", fg="Yellow", font=("arial", 35,
"bold"),padx=10,
)
titl.place(x=525, y=12)

a = tk.Label(
    window,
    text="Welcome to the Face Recognition Based\nAttendance Management
System \n Created by: G7 Group of EC-02 Batch",
    bg="black",
    fg="white",
    bd=10,
    font=("arial", 35),
)
a.pack()

ri = Image.open("UI_Image/register.png")
r = ImageTk.PhotoImage(ri)
label1 = Label(window, image=r)
label1.image = r
label1.place(x=100, y=270)

ai = Image.open("UI_Image/attendance.png")
a = ImageTk.PhotoImage(ai)
label2 = Label(window, image=a)
label2.image = a
label2.place(x=980, y=270)

vi = Image.open("UI_Image/verifyy.png")
v = ImageTk.PhotoImage(vi)
```

```
label3 = Label(window, image=v)
label3.image = v
label3.place(x=600, y=270)


def TakeImageUI():
    ImageUI = Tk()
    ImageUI.title("Take Student Image..")
    ImageUI.geometry("780x480")
    ImageUI.configure(background="black")
    ImageUI.resizable(0, 0)
    titl = tk.Label(ImageUI, bg="black", relief=RIDGE, bd=10,
font=("arial", 35))
    titl.pack(fill=X)
    # image and title
    titl = tk.Label(
        ImageUI, text="Register Your Face", bg="black", fg="green",
font=("arial", 30),
    )
    titl.place(x=270, y=12)

    # heading
    a = tk.Label(
        ImageUI,
        text="Enter the details",
        bg="black",
        fg="white",
        bd=10,
        font=("arial", 24),
    )
    a.place(x=280, y=75)

    # ER no
    lbl1 = tk.Label(
        ImageUI,
        text="Enrollment No",
        width=10,
        height=2,
        bg="black",
        fg="white",
        bd=5,
        relief=RIDGE,
        font=("times new roman", 12),
    )
```

```
    lbl1.place(x=120, y=130)
    txt1 = tk.Entry(
        ImageUI,
        width=17,
        bd=5,
        validate="key",
        bg="black",
        fg="white",
        relief=RIDGE,
        font=("times", 25, "bold"),
    )
    txt1.place(x=250, y=130)
    txt1["validatecommand"] = (txt1.register(testVal), "%P", "%d")

    # name
    lbl2 = tk.Label(
        ImageUI,
        text="Name",
        width=10,
        height=2,
        bg="black",
        fg="white",
        bd=5,
        relief=RIDGE,
        font=("times new roman", 12),
    )
    lbl2.place(x=120, y=200)
    txt2 = tk.Entry(
        ImageUI,
        width=17,
        bd=5,
        bg="black",
        fg="white",
        relief=RIDGE,
        font=("times", 25, "bold"),
    )
    txt2.place(x=250, y=200)

    lbl3 = tk.Label(
        ImageUI,
        text="Notification",
        width=10,
        height=2,
        bg="black",
```

```
        fg="white",
        bd=5,
        relief=RIDGE,
        font=("times new roman", 12),
    )
    lbl3.place(x=120, y=270)

    message = tk.Label(
        ImageUI,
        text="",
        width=32,
        height=2,
        bd=5,
        bg="black",
        fg="white",
        relief=RIDGE,
        font=("times", 12, "bold"),
    )
    message.place(x=250, y=270)

    def take_image():
        l1 = txt1.get()
        l2 = txt2.get()
        takeImage.TakeImage(
            l1,
            l2,
            haarcasecade_path,
            trainimage_path,
            message,
            err_screen,
            text_to_speech,
        )
        txt1.delete(0, "end")
        txt2.delete(0, "end")

    # take Image button
    # image
    takeImg = tk.Button(
        ImageUI,s
        text="Take Image",
        command=take_image,
        bd=10,
        font=("times new roman", 18),
        bg="black",
```

```
        fg="red",
        height=2,
        width=12,
        relief=RIDGE,
    )
    takeImg.place(x=130, y=350)

    def train_image():
        trainImage.TrainImage(
            haarcasecade_path,
            trainimage_path,
            trainimagelabel_path,
            message,
            text_to_speech,
        )

    # train Image function call
    trainImg = tk.Button(
        ImageUI,
        text="Train Image",
        command=train_image,
        bd=10,
        font=("times new roman", 18),
        bg="black",
        fg="red",
        height=2,
        width=12,
        relief=RIDGE,
    )
    trainImg.place(x=360, y=350)


r = tk.Button(
    window,
    text="Register a new student",
    command=TakeImageUI,
    bd=10,
    font=("times new roman", 16),
    bg="black",
    fg="red",
    height=2,
    width=17,
)
r.place(x=100, y=520)
```

```
def automatic_attedance():
    automaticAttedance.subjectChoose(text_to_speech)


r = tk.Button(
    window,
    text="Take Attendance",
    command=automatic_attedance,
    bd=10,
    font=("times new roman", 16),
    bg="black",
    fg="red",
    height=2,
    width=17,
)
r.place(x=600, y=520)


def view_attendance():
    show_attendance.subjectchoose(text_to_speech)


r = tk.Button(
    window,
    text="View Attendance",
    command=view_attendance,
    bd=10,
    font=("times new roman", 16),
    bg="black",
    fg="red",
    height=2,
    width=17,
)
r.place(x=1000, y=520)
r = tk.Button(
    window,
    text="EXIT",
    bd=10,
    command=quit,
    font=("times new roman", 16),
    bg="black",
    fg="orange",
    height=2,
```

```
    width=17,
)
r.place(x=600, y=660)

window.mainloop()
```

This is a GUI based face recognition attendance management system.
The required packages are imported at the beginning of the code. The project module is imported which contains the modules required for showing attendance, taking image, training image and automatic attendance.

- The program starts by creating an instance of tkinter as window and setting its title, geometry, and background color. The program then creates the necessary labels and buttons for the user interface.

- The TakeImageUI function creates a new window for taking an image of a student. The function first creates an instance of tkinter as ImageUI and sets its title, geometry, and background color. It then creates the necessary labels and text fields for the user interface.

- The testVal function is used to validate the user input. It checks if the input is numeric or not.

- The del_sc1 function is used to destroy a screen.

- The err_screen function displays an error message when the user input is incorrect.

- The text_to_speech function is used to convert text to speech.
- The program defines various paths where the dataset, CSV files, and XML files are stored.

This code acts as front-end interface of the project.

AutomaticAttendency.py file:

```
import tkinter as tk
from tkinter import *
import os, cv2
import shutil
```

```
import csv
import numpy as np
from PIL import ImageTk, Image
import pandas as pd
import datetime
import time
import subprocess
import tkinter.ttk as tkk
import tkinter.font as font

haarcasecade_path = "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/haarcascade_frontalface_default.xml"
trainimagelabel_path = (
    "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/TrainingImageLabel/Trainner.yml"
)
trainimage_path = "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/TrainingImage"
studentdetail_path = (
    "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/StudentDetails/studentdetails.csv"
)
attendance_path = ("/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/Attendence")
# for choose subject and fill attendance
def subjectChoose(text_to_speech):
    def FillAttendance():
        sub = tx.get()
        now = time.time()
        future = now + 20
        print(now)
        print(future)
        if sub == "":
            t = "Please enter the subject name!!!"
            text_to_speech(t)
        else:
            try:
                recognizer = cv2.face.LBPHFaceRecognizer_create()
                try:
                    recognizer.read(trainimagelabel_path)
                except:
                    e = "Model not found,please train model"
                    Notifica.configure(
                        text=e,
```

```
                    bg="black",
                    fg="white",
                    width=33,
                    font=("times", 15, "bold"),
                )
                Notifica.place(x=20, y=250)
                text_to_speech(e)
            facecasCade = cv2.CascadeClassifier(haarcasecade_path)
            df = pd.read_csv(studentdetail_path)
            cam = cv2.VideoCapture(0)
            font = cv2.FONT_HERSHEY_SIMPLEX
            col_names = ["Enrollment", "Name"]
            attendance = pd.DataFrame(columns=col_names)
            while True:
                ___, im = cam.read()
                gray = cv2.cvtColor(im, cv2.COLOR_BGR2GRAY)
                faces = facecasCade.detectMultiScale(gray, 1.2, 5)
                for (x, y, w, h) in faces:
                    global Id

             Id, conf = recognizer.predict(gray[y : y + h, x : x + w])
                    if conf < 70:
                        print(conf)
                        global Subject
                        global aa
                        global date
                        global timeStamp
                        Subject = tx.get()
                        ts = time.time()
                 date = datetime.datetime.fromtimestamp(ts).strftime(
                            "%Y-%m-%d"
                        )
                 timeStamp = datetime.datetime.fromtimestamp(ts).strftime(
                            "%H:%M:%S"
                        )
                 aa = df.loc[df["Enrollment"] == Id]["Name"].values
                        global tt
                        tt = str(Id) + "-" + aa
                        # En='1604501160'+str(Id)
                        attendance.loc[len(attendance)] = [
                            Id,
                            aa,
                        ]
                cv2.rectangle(im, (x, y), (x + w, y + h), (0, 260, 0), 4)
```

```
                cv2.putText(
                      im, str(tt), (x + h, y), font, 1, (255, 255, 0,), 4
                           )
                       else:
                           Id = "Unknown"
                           tt = str(Id)
                cv2.rectangle(im, (x, y), (x + w, y + h), (0, 25, 255), 7)
                           cv2.putText(
                             im, str(tt), (x + h, y), font, 1, (0, 25, 255), 4
                           )
                    if time.time() > future:
                        break

                    attendance = attendance.drop_duplicates(
                        ["Enrollment"], keep="first"
                    )
                    cv2.imshow("Filling Attendance...", im)
                    key = cv2.waitKey(30) & 0xFF
                    if key == 27:
                        break

                ts = time.time()
                print(aa)

                attendance[date] = 1
            date = datetime.datetime.fromtimestamp(ts).strftime("%Y-%m-%d")
                timeStamp =
datetime.datetime.fromtimestamp(ts).strftime("%H:%M:%S")
                Hour, Minute, Second = timeStamp.split(":")
                path = os.path.join(attendance_path, Subject)
                fileName = (
                    f"{path}/"
                    + Subject
                    + "_"
                    + date
                    + "_"
                    + Hour
                    + "-"
                    + Minute
                    + "-"
                    + Second
                    + ".csv"
                )
```

```
                attendance = attendance.drop_duplicates(["Enrollment"],
keep="first")
                print(attendance)
                print(fileName)
                attendance.to_csv(fileName, index=False)

                m = "Attendance Filled Successfully of " + Subject
                Notifica.configure(
                    text=m,
                    bg="black",
                    fg="white",
                    width=33,
                    relief=RIDGE,
                    bd=5,
                    font=("times", 15, "bold"),
                )
                text_to_speech(m)

                Notifica.place(x=20, y=250)

                cam.release()
                cv2.destroyAllWindows()

                import csv
                import tkinter

                root = tkinter.Tk()
                root.title("Attendance of " + Subject)
                root.configure(background="black")
                cs = os.path.join(path, fileName)
                print(cs)
                with open(cs, newline="") as file:
                    reader = csv.reader(file)
                    r = 0

                    for col in reader:
                        c = 0
                        for row in col:

                            label = tkinter.Label(
                                root,
                                width=10,
                                height=1,
                                fg="white",
```

```
                                font=("times", 15, " bold "),
                                bg="black",
                                text=row,
                                relief=tkinter.RIDGE,
                            )
                            label.grid(row=r, column=c)
                            c += 1
                        r += 1
                root.mainloop()
                print(attendance)
            except:
                f = "No Face found for attendance"
                text_to_speech(f)
                cv2.destroyAllWindows()

    ###windo is frame for subject chooser
    subject = Tk()
    # windo.iconbitmap("AMS.ico")
    subject.title("Subject...")
    subject.geometry("580x320")
    subject.resizable(0, 0)
    subject.configure(background="black")
    titl = tk.Label(subject, bg="black", relief=RIDGE, bd=10,
font=("arial", 30))
    titl.pack(fill=X)
    titl = tk.Label(
        subject,
        text="Enter the Subject Name",
        bg="black",
        fg="green",
        font=("arial", 25),
    )
    titl.place(x=160, y=12)
    Notifica = tk.Label(
        subject,
        text="Attendance filled Successfully",
        bg="white",
        fg="black",
        width=33,
        height=2,
        font=("times", 15, "bold"),
    )

    def Attf():
```

```
        sub = tx.get()
        if sub == "":
            t = "Please enter the subject name!!!"
            text_to_speech(t)
        else:
         file_path = f"/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-recognition/Attendence/{sub}"
         subprocess.run(["open", file_path])


    attf = tk.Button(
        subject,
        text="Check Sheets",
        command=Attf,
        bd=7,
        font=("times new roman", 15),
        bg="black",
        fg="red",
        height=2,
        width=10,
        relief=RIDGE,
    )
    attf.place(x=360, y=170)

    sub = tk.Label(
        subject,
        text="Enter Subject",
        width=10,
        height=2,
        bg="black",
        fg="white",
        bd=5,
        relief=RIDGE,
        font=("times new roman", 15),
    )
    sub.place(x=50, y=100)

    tx = tk.Entry(
        subject,
        width=15,
        bd=5,
        bg="black",
        fg="white",
        relief=RIDGE,
```

```
        font=("times", 30, "bold"),
    )
    tx.place(x=190, y=100)

    fill_a = tk.Button(
        subject,
        text="Fill Attendance",
        command=FillAttendance,
        bd=7,
        font=("times new roman", 15),
        bg="black",
        fg="red",
        height=2,
        width=12,
        relief=RIDGE,
    )
    fill_a.place(x=195, y=170)
    subject.mainloop()
```

The program uses OpenCV for facial recognition, and tkinter for the GUI. The program requires a pre-trained face recognition model in the form of a YAML file, which is used to recognize faces in real time.

The program has a GUI that allows the user to select a subject and fill in attendance for that subject. When the user selects a subject and clicks on the "Fill Attendance" button, the program captures images from the webcam, detects faces, and matches them with the faces in the training dataset using the pre-trained model. The program then creates a CSV file for the selected subject and fills in attendance for the students who were present in the class.

The program also has a text-to-speech feature that reads out messages to the user. The program uses several external libraries like os, cv2, shutil, csv, numpy, pandas, datetime, time, subprocess, and tkinter.ttk to implement various features. Overall, this program offers an automated way to take attendance, saving time and reducing errors associated with manual attendance taking.

takeImage.py file:

```
import csv
import os, cv2
import numpy as np
import pandas as pd
```

```
import datetime
import time

# take Image of user
def TakeImage(l1, l2, haarcasecade_path, trainimage_path, message,
err_screen,text_to_speech):
    if (l1 == "") and (l2==""):
        t='Please Enter the your Enrollment Number and Name.'
        text_to_speech(t)
    elif l1=='':
        t='Please Enter the your Enrollment Number.'
        text_to_speech(t)
    elif l2 == "":
        t='Please Enter the your Name.'
        text_to_speech(t)
    else:
        try:
            cam = cv2.VideoCapture(0)
            detector = cv2.CascadeClassifier(haarcasecade_path)
            Enrollment = l1
            Name = l2
            sampleNum = 0
            directory = Enrollment + "_" + Name
            path = os.path.join(trainimage_path, directory)
            os.mkdir(path)

            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)
                    sampleNum = sampleNum + 1
                    cv2.imwrite(
                        f"{path}/ "
                        + Name
                        + "_"
                        + Enrollment
                        + "_"
                        + str(sampleNum)
                        + ".jpg",
                        gray[y : y + h, x : x + w],
                    )
```

```
                    cv2.imshow("Frame", img)
                if cv2.waitKey(1) & 0xFF == ord("q"):
                    break
                elif sampleNum > 100:
                    break
            cam.release()
            cv2.destroyAllWindows()
            row = [Enrollment, Name]
            with open(
 "/Users/shivakumar/Desktop/new/Attendance-Management-system-using-face-
recognition/StudentDetails/studentdetails.csv",
                "a+",
            ) as csvFile:
                writer = csv.writer(csvFile, delimiter=",")
                writer.writerow(row)
                csvFile.close()
            res = "Images Saved for ER No:" + Enrollment + " Name:" + Name
            message.configure(text=res)
            text_to_speech(res)
        except FileExistsError as F:
            F = "Student Data already exists"
            text_to_speech(F)
```

This code defines a function *TakeImage()* which captures images of a user's face using the computer's webcam and saves them to a specified directory. The function takes five arguments: *l1* and *l2* which represent the user's enrollment number and name respectively, *haarcasecade_path* which is the path to the Haar Cascade classifier used for face detection, *trainimage_path* which is the path to the directory where the captured images will be saved, and *text_to_speech* which is a function used for text-to-speech output.

Inside the function, the enrollment number and name are checked to ensure they have been entered, and if not, an appropriate error message is displayed using *text_to_speech()*. The function then initializes the webcam and Haar Cascade classifier, creates a directory based on the user's enrollment number and name, and begins capturing images of the user's face. Each captured image is saved to the specified directory with a filename that includes the user's name, enrollment number, and a sample number. The function continues capturing images until either the user quits the program or 100 images have been captured.

Once image capturing is complete, the function writes the enrollment number and name to a CSV file that is used to keep track of all enrolled users. A success

message is then displayed using *message.configure(text=res)* and also output using *text_to_speech()*. If the user already exists, a FileExistsError is raised and an appropriate error message is displayed using *text_to_speech()*.

Show_attendence.py file:

```
import pandas as pd
from glob import glob
import os
import tkinter
import csv
import subprocess
import tkinter as tk
from tkinter import *

def subjectchoose(text_to_speech):
    def calculate_attendance():
        Subject = tx.get()
        if Subject=="":
            t='Please enter the subject name.'
            text_to_speech(t)
        os.chdir(
            f"/Users/shivakumar/Desktop/new/Attendance-Management-system-
using-face-recognition/Attendence/{Subject}"
        )
        filenames = glob(
            f"/Users/shivakumar/Desktop/new/Attendance-Management-system-
using-face-recognition/Attendence/{Subject}/{Subject}*.csv"
        )
        df = [pd.read_csv(f) for f in filenames]
        newdf = df[0]
        for i in range(1, len(df)):
            newdf = newdf.merge(df[i], how="outer")
        newdf.fillna(0, inplace=True)
        newdf["Attendance"] = 0
        for i in range(len(newdf)):
            newdf["Attendance"].iloc[i] = str(int(round(newdf.iloc[i, 2:-
1].mean() * 100)))+'%'
            #newdf.sort_values(by=['Enrollment'],inplace=True)
        newdf.to_csv("attendance.csv", index=False)
```

```
        root = tkinter.Tk()
        root.title("Attendance of "+Subject)
        root.configure(background="black")
        cs = f"/Users/shivakumar/Desktop/new/Attendance-Management-system-
using-face-recognition/Attendence/{Subject}/attendance.csv"
        with open(cs) as file:
            reader = csv.reader(file)
            r = 0

            for col in reader:
                c = 0
                for row in col:

                    label = tkinter.Label(
                        root,
                        width=10,
                        height=1,
                        fg="white",
                        font=("times", 15, " bold "),
                        bg="black",
                        text=row,
                        relief=tkinter.RIDGE,
                    )
                    label.grid(row=r, column=c)
                    c += 1
                r += 1
        root.mainloop()
        print(newdf)

    subject = Tk()
    # windo.iconbitmap("AMS.ico")
    subject.title("Subject...")
    subject.geometry("580x320")
    subject.resizable(0, 0)
    subject.configure(background="black")
    titl = tk.Label(subject, bg="black", relief=RIDGE, bd=10,
font=("arial", 30))
    titl.pack(fill=X)
    titl = tk.Label(
        subject,
        text="Which Subject of Attendance?",
        bg="black",
        fg="green",
        font=("arial", 25),
```

```
    )
    titl.place(x=100, y=12)

    def Attf():
     sub = tx.get()
     if sub == "":
        t="Please enter the subject name!!!"
        text_to_speech(t)

     else:
        file_path = f"/Users/shivakumar/Desktop/new/Attendance-Management-
system-using-face-recognition/Attendence/{sub}"
        subprocess.run(["open", file_path])

    attf = tk.Button(
        subject,
        text="Check Sheets",
        command=Attf,
        bd=7,
        font=("times new roman", 15),
        bg="black",
        fg="red",
        height=2,
        width=10,
        relief=RIDGE,
    )
    attf.place(x=360, y=170)

    sub = tk.Label(
        subject,
        text="Enter Subject",
        width=10,
        height=2,
        bg="black",
        fg="white",
        bd=5,
        relief=RIDGE,
        font=("times new roman", 15),
    )
    sub.place(x=50, y=100)

    tx = tk.Entry(
        subject,
        width=15,
```

```
        bd=5,
        bg="black",
        fg="white",
        relief=RIDGE,
        font=("times", 30, "bold"),
    )
    tx.place(x=190, y=100)

    fill_a = tk.Button(
        subject,
        text="View Attendance",
        command=calculate_attendance,
        bd=7,
        font=("times new roman", 15),
        bg="black",
        fg="red",
        height=2,
        width=12,
        relief=RIDGE,
    )
    fill_a.place(x=195, y=170)
    subject.mainloop()
```

The script uses the pandas library to read CSV files and perform data manipulation, and also uses tkinter library for creating the GUI. The script begins by defining a function named *subjectchoose* that prompts the user to enter a subject and then calculates attendance based on the corresponding CSV files. The attendance data is then displayed in a tkinter window.

The function subjectchoose contains a nested function called *calculate_attendance* which performs the actual calculation of attendance. It takes the user input subject, navigates to the corresponding directory using the os module, reads all the CSV files present in the directory using the glob module and merges them into a single DataFrame using the pandas library. The merged DataFrame is then used to calculate the attendance percentage of each student and create a new CSV file.

The attendance percentage data is displayed in a new tkinter window using a nested for loop that reads the data from the CSV file and creates tkinter labels for each row and column of the attendance data.
The tkinter window also contains buttons for checking attendance sheets and viewing attendance data.

Takemanually.py file:

```
import tkinter as tk
from tkinter import Message, Text
import os, cv2
import shutil
import csv
import numpy as np
from PIL import ImageTk, Image
import pandas as pd
import datetime
import time
import tkinter.ttk as tkk
import tkinter.font as font

ts = time.time()
Date = datetime.datetime.fromtimestamp(ts).strftime("%Y_%m_%d")
timeStamp = datetime.datetime.fromtimestamp(ts).strftime("%H:%M:%S")
Time = datetime.datetime.fromtimestamp(ts).strftime("%H:%M:%S")
Hour, Minute, Second = timeStamp.split(":")
d = {}
index = 0
####GUI for manually fill attendance
def manually_fill():
    global sb
    sb = tk.Tk()
    sb.iconbitmap("/Users/shivakumar/Desktop/new/Attendance-Management-
system-using-face-recognition/AMS.ico")
    sb.title("Enter subject name...")
    sb.geometry("580x320")
    sb.configure(background="snow")

    def err_screen_for_subject():
        def ec_delete():
            ec.destroy()

        global ec
        ec = tk.Tk()
        ec.geometry("300x100")
        ec.iconbitmap("/Users/shivakumar/Desktop/new/Attendance-Management-
system-using-face-recognition/AMS.ico")
        ec.title("Warning!!")
        ec.configure(background="snow")
        tk.Label(
```

```
            ec,
            text="Please enter subject name!!!",
            fg="red",
            bg="white",
            font=("times", 16, " bold "),
        ).pack()
        tk.Button(
            ec,
            text="OK",
            command=ec_delete,
            fg="black",
            bg="lawn green",
            width=9,
            height=1,
            activebackground="Red",
            font=("times", 15, " bold "),
        ).place(x=90, y=50)

    def fill_attendance():

        ##Create table for Attendance
        global subb
        subb = SUB_ENTRY.get()

        if subb == "":
            err_screen_for_subject()
        else:
            sb.destroy()
            MFW = tk.Tk()
            MFW.iconbitmap("/Users/shivakumar/Desktop/new/Attendance-
Management-system-using-face-recognition/AMS.ico")
            MFW.title("Manually attendance of " + str(subb))
            MFW.geometry("880x470")
            MFW.configure(background="snow")

            def del_errsc2():
                errsc2.destroy()

            def err_screen1():
                global errsc2
                errsc2 = tk.Tk()
                errsc2.geometry("330x100")
```

```
errsc2.iconbitmap("/Users/shivakumar/Desktop/new/Attendance-Management-
system-using-face-recognition/AMS.ico")
                errsc2.title("Warning!!")
                errsc2.configure(background="snow")
                tk.Label(
                    errsc2,
                    text="Please enter Student & Enrollment!!!",
                    fg="red",
                    bg="white",
                    font=("times", 16, " bold "),
                ).pack()
                tk.Button(
                    errsc2,
                    text="OK",
                    command=del_errsc2,
                    fg="black",
                    bg="lawn green",
                    width=9,
                    height=1,
                    activebackground="Red",
                    font=("times", 15, " bold "),
                ).place(x=90, y=50)

            def testVal(inStr, acttyp):
                if acttyp == "1":  # insert
                    if not inStr.isdigit():
                        return False
                return True

            ENR = tk.Label(
                MFW,
                text="Enter Enrollment",
                width=15,
                height=2,
                fg="white",
                bg="blue2",
                font=("times", 15, " bold "),
            )
            ENR.place(x=30, y=100)

            STU_NAME = tk.Label(
                MFW,
                text="Enter Student name",
```

```
            width=15,
            height=2,
            fg="white",
            bg="blue2",
            font=("times", 15, " bold "),
        )
        STU_NAME.place(x=30, y=200)

        global ENR_ENTRY
        ENR_ENTRY = tk.Entry(
            MFW,
            width=20,
            validate="key",
            bg="white",
            fg="red",
            font=("times", 23, " bold "),
        )
        ENR_ENTRY["validatecommand"] = (ENR_ENTRY.register(testVal),
"%P", "%d")
        ENR_ENTRY.place(x=290, y=105)

        def remove_enr():
            ENR_ENTRY.delete(first=0, last=22)

        STUDENT_ENTRY = tk.Entry(
            MFW, width=20, bg="white", fg="red", font=("times", 23, "
bold ")
        )
        STUDENT_ENTRY.place(x=290, y=205)

        def remove_student():
            STUDENT_ENTRY.delete(first=0, last=22)

        ####get important variable

        def enter_data_DB():
            global index
            global d
            ENROLLMENT = ENR_ENTRY.get()
            STUDENT = STUDENT_ENTRY.get()
            if ENROLLMENT == "":
                err_screen1()
            elif STUDENT == "":
                err_screen1()
```

```
                else:
                    if index == 0:
                        d = {
                            index: {"Enrollment": ENROLLMENT, "Name":
STUDENT, Date: 1}
                        }
                        index += 1
                        ENR_ENTRY.delete(0, "end")
                        STUDENT_ENTRY.delete(0, "end")
                    else:
                        d[index] = {"Enrollment": ENROLLMENT, "Name":
STUDENT, Date: 1}
                        index += 1
                        ENR_ENTRY.delete(0, "end")
                        STUDENT_ENTRY.delete(0, "end")
                    # TODO implement CSV code
                print(d)

            def create_csv():
                df = pd.DataFrame(d)
                csv_name = (
                    "Attendance-Management-system-using-face-
recognition/Attendence/"
                    + subb
                    + "_"
                    + Date
                    + "_"
                    + Hour
                    + "-"
                    + Minute
                    + "-"
                    + Second
                    + ".csv"
                )
                df.to_csv(csv_name)
                O = "CSV created Successfully"
                Notifi.configure(
                    text=O,
                    bg="Green",
                    fg="white",
                    width=33,
                    font=("times", 19, "bold"),
                )
                Notifi.place(x=180, y=380)
```

```
        Notifi = tk.Label(
            MFW,
            text="CSV created Successfully",
            bg="Green",
            fg="white",
            width=33,
            height=2,
            font=("times", 19, "bold"),
        )

        c1ear_enroll = tk.Button(
            MFW,
            text="Clear",
            command=remove_enr,
            fg="black",
            bg="deep pink",
            width=10,
            height=1,
            activebackground="Red",
            font=("times", 15, " bold "),
        )
        c1ear_enroll.place(x=690, y=100)

        c1ear_student = tk.Button(
            MFW,
            text="Clear",
            command=remove_student,
            fg="black",
            bg="deep pink",
            width=10,
            height=1,
            activebackground="Red",
            font=("times", 15, " bold "),
        )
        c1ear_student.place(x=690, y=200)

        DATA_SUB = tk.Button(
            MFW,
            text="Enter Data",
            command=enter_data_DB,
            fg="black",
            bg="lime green",
            width=20,
```

```
            height=2,
            activebackground="Red",
            font=("times", 15, " bold "),
        )
        DATA_SUB.place(x=170, y=300)

        MAKE_CSV = tk.Button(
            MFW,
            text="Convert to CSV",
            command=create_csv,
            fg="black",
            bg="red",
            width=20,
            height=2,
            activebackground="Red",
            font=("times", 15, " bold "),
        )
        MAKE_CSV.place(x=570, y=300)
        # TODO remove check sheet
        def attf():
            import subprocess

            subprocess.Popen(
                "/Users/shivakumar/Desktop/new/Attendance-Management-
system-using-face-recognition/Attendence"
            )

        attf = tk.Button(
            MFW,
            text="Check Sheets",
            command=attf,
            fg="black",
            bg="lawn green",
            width=12,
            height=1,
            activebackground="Red",
            font=("times", 14, " bold "),
        )
        attf.place(x=730, y=410)

        MFW.mainloop()

    SUB = tk.Label(
        sb,
```

```
        text="Enter Subject",
        width=15,
        height=2,
        fg="white",
        bg="blue2",
        font=("times", 15, " bold "),
    )
    SUB.place(x=30, y=100)

    global SUB_ENTRY

    SUB_ENTRY = tk.Entry(
        sb, width=20, bg="white", fg="red", font=("times", 23, " bold ")
    )
    SUB_ENTRY.place(x=250, y=105)

    fill_manual_attendance = tk.Button(
        sb,
        text="Fill Attendance",
        command=fill_attendance,
        fg="white",
        bg="deep pink",
        width=20,
        height=2,
        activebackground="Red",
        font=("times", 15, " bold "),
    )
    fill_manual_attendance.place(x=250, y=160)
    sb.mainloop()
```

The code appears to be a graphical user interface (GUI) application for managing attendance records of students. The application uses the Python library tkinter for its GUI and relies on other libraries such as cv2, os, shutil, csv, numpy, pandas, datetime, and time for other tasks. The code is divided into two major sections:

Graphical User Interface (GUI):

This section defines the graphical interface of the application and contains two functions *manually_fill()* and *enter_data_DB()*. The first function manually_fill() creates a GUI for manually filling in attendance records while the second function *enter_data_DB()* handles the submission of data entered manually.

The GUI interface created by *manually_fill()* function contains three windows, namely *sb, ec,* and *MFW*. The first window *(sb)* prompts the user to enter the name of the subject for which attendance is being taken, while the second window *(ec)* issues a warning message to the user to enter the subject name. The third window *(MFW)* is where the actual attendance is taken. It prompts the user to enter the enrollment number and name of the student and displays a warning message if either of these fields is left blank. It also provides a button to create a CSV file for storing the attendance records.

Attendance Management:

This section of the code handles the actual management of attendance records. It defines variables, creates a dictionary for storing attendance data, and creates a CSV file for storing attendance records.
The variables defined include *ts, Date, timeStamp, Time, Hour, Minute, Second, d,* and *index*. These variables are used for timestamping attendance records.
The dictionary d is used for storing attendance data, while the index index is used for indexing the attendance data. The *enter_data_DB()* function updates the attendance data by adding new entries to the dictionary.
The *create_csv()* function converts the attendance data in the dictionary to a pandas dataframe and saves it as a CSV file. The CSV file is stored with the name "Attendance_date.csv", where date is the current date.

Test.py file:

```
import requests
import cv2
import numpy as np

url = "http://192.168.0.6:8080/shot.jpg"

while True:
    cam = requests.get(url)
    imgNp = np.array(bytearray(cam.content), dtype=np.uint8)
    img = cv2.imdecode(imgNp, -1)
    cv2.imshow("cam", img)

    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
```

This code uses the *requests* library to access a URL that serves an image from a camera, and then uses OpenCV *(cv2)* to display the image.

The URL is defined as a string and points to the address of the camera that serves the image. This URL may need to be adjusted to match the address of the camera being used.

The code then enters a while loop that continuously fetches the image from the camera using the URL. It does this by sending an HTTP GET request using the *requests* library, which returns a response object containing the image data. The image data is then converted to a NumPy array using *np.array* with *dtype=np.uint8.* The *dtype* argument specifies the data type of the array, which in this case is unsigned 8-bit integers.

The NumPy array is then decoded using *cv2.imdecode* to create an image that can be displayed using OpenCV. The *imdecode* function decodes the image from memory and returns a NumPy array representing the image.

The *cv2.imshow* function displays the image in a window with the title "cam". The image will be continuously updated in the window as the while loop iterates. The while loop will continue to run until the user presses the "q" key. When the "q" key is pressed, the loop will exit and the program will terminate.

Overall, this code retrieves an image from a camera using a URL, converts the image data to a NumPy array, decodes the array into an image, and displays the image using OpenCV.

Trainimage.py:

```
import csv
import os
import cv2
import numpy as np
import pandas as pd
import datetime
import time
from PIL import ImageTk, Image

# Train Image
def TrainImage(haarcasecade_path, trainimage_path, trainimagelabel_path,
message,text_to_speech):
    recognizer = cv2.face.LBPHFaceRecognizer_create()
    detector = cv2.CascadeClassifier(haarcasecade_path)
    faces, Id = getImagesAndLables(trainimage_path)
    recognizer.train(faces, np.array(Id))
```

```
    recognizer.save(trainimagelabel_path)
    res = "Image Trained successfully"  # +",".join(str(f) for f in Id)
    message.configure(text=res)
    text_to_speech(res)

def getImagesAndLables(path):
    faces = []
    Ids = []
    imagePath = []
    for foldername in os.listdir(path):
        folderpath = os.path.join(path, foldername)
        if not os.path.isdir(folderpath):
            continue
        for filename in os.listdir(folderpath):
            if filename.endswith('.jpg'):
                filepath = os.path.join(folderpath, filename)
                pilImage = Image.open(filepath).convert("L")
                imageNp = np.array(pilImage, "uint8")
                Id = int(os.path.split(filepath)[-1].split("_")[1])
                imagePath.append(filepath)
                faces.append(imageNp)
                Ids.append(Id)
    return faces, Ids
```

This code defines a function called TrainImage, which trains an image recognizer using the LBPH algorithm from OpenCV. The function takes in four parameters:

- haarcasecade_path: a string that represents the path to the Haar cascade classifier used to detect faces in images.
- trainimage_path: a string that represents the path to the folder containing the training images.
- trainimagelabel_path: a string that represents the path to save the trained recognizer.
- message: a tkinter label object that will display a message after training.

The *TrainImage* function uses the *getImagesAndLables* function to get the training data. The *getImagesAndLables* function loops over all the folders in the *trainimage_path* directory, and for each folder, it loops over all the *.jpg* files in the folder. It opens each file as a grayscale image, converts it to a numpy array, and extracts the label (person's ID) from the filename. The function then returns two arrays: *faces* contains the numpy arrays of each image, and *Ids* contains the label for each image.

The *TrainImage* function then initializes a recognizer and a detector object. It passes the training data to the recognizer object using the *train* method, and saves the trained recognizer to *trainimagelabel_path.* Finally, it updates the message label to indicate that training was successful and plays the message using the *text_to_speech* function.

Overall, this code is designed to train an image recognizer to identify people from a set of training images using the LBPH algorithm from OpenCV.

Trainer.yml:

Sample:

```
%YAML:1.0
---
opencv_lbphfaces:
   threshold: 1.7976931348623157e+308
   radius: 1
   neighbors: 8
   grid_x: 8
   grid_y: 8
   histograms:
      - !!opencv-matrix
         rows: 1
         cols: 16384
         dt: f
         data: [ 1.53787003e-03, 3.84467514e-03, 0., 0., 4.22914280e-03,
             3.84467508e-04, 3.84467508e-04, 1.80699732e-02, 0., 0., 0.,
             0., 1.15340250e-03, 0., 3.84467508e-04, 3.84467508e-04,
             8.45828559e-03, 1.53787003e-03, 0., 3.84467508e-04,
             4.99807764e-03, 0., 0., 4.61360998e-03, 1.53787003e-03,
             3.84467508e-04, 0., 0., 6.92041516e-02, 2.30680499e-03,
             1.11495573e-02, 4.26758938e-02, 0., 0., 0., 0.,
             3.84467508e-04, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             7.68935017e-04, 0., 0., 0., 0., 0., 0., 1.53787003e-03,
             0., 0., 0., 1.06497496e-01, 2.69127265e-03, 2.80661285e-02,
             1.34563632e-02, 3.84467508e-04, 0., 0., 0., 2.30680499e-03,
             4.61360998e-03, 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
             0., 0., 0., 0., 0., 6.92041498e-03, 1.92233757e-03, 0., 0.,
             5.38254529e-03, 0., 0., 2.30680499e-03, 2.69127265e-03,
             3.84467508e-04, 0., 0., 1.09957710e-01, 4.22914280e-03,…….
```

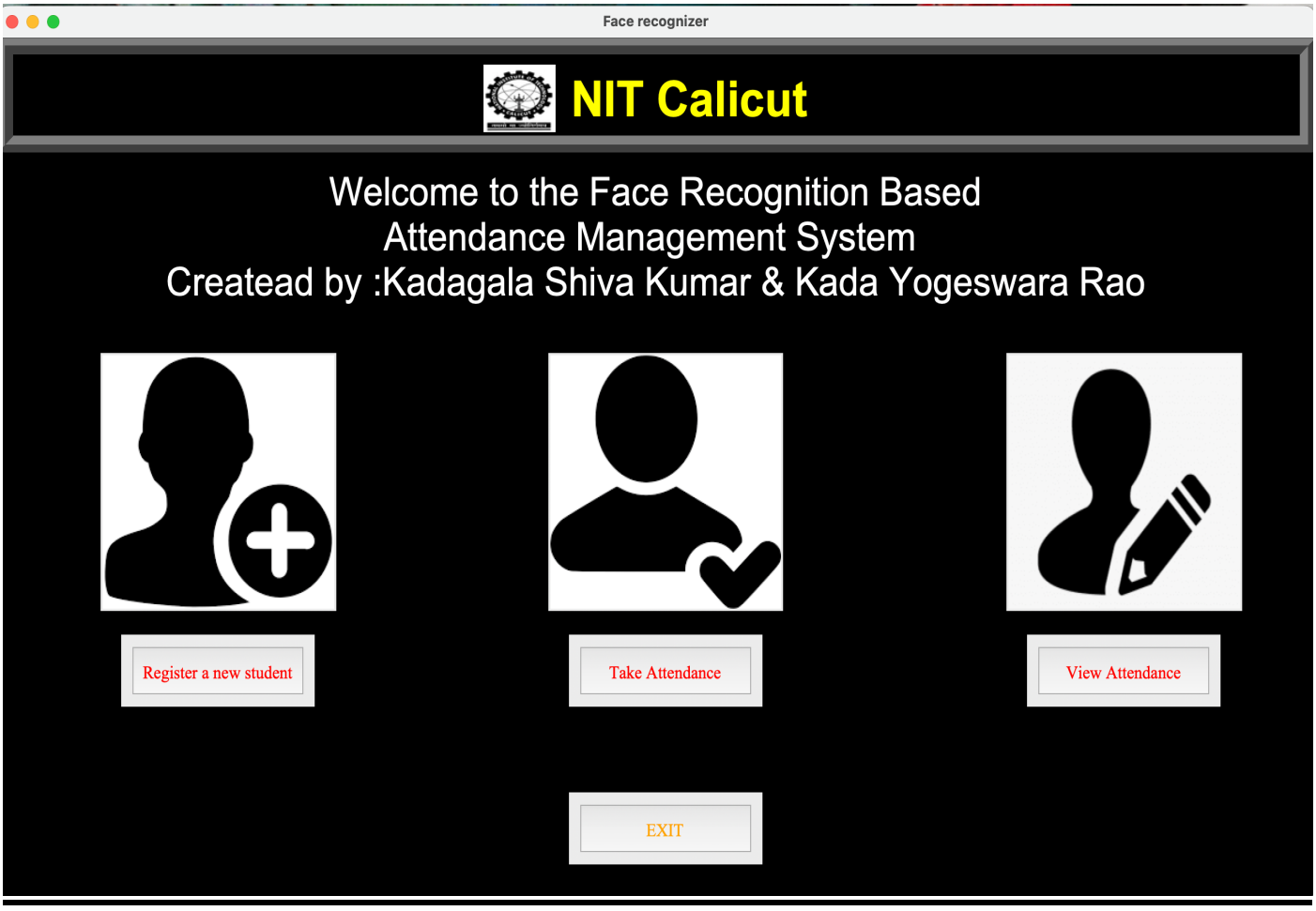
In face recognition projects, *trainer.yml* is a file that stores a trained machine learning model for recognizing faces. It is generated using training data, typically consisting of images of faces of different individuals along with their corresponding labels.

The training process involves feeding the images into a machine learning algorithm, which then learns to recognize the unique features of each individual face. This information is then used to create a model that can recognize faces with a certain level of accuracy.

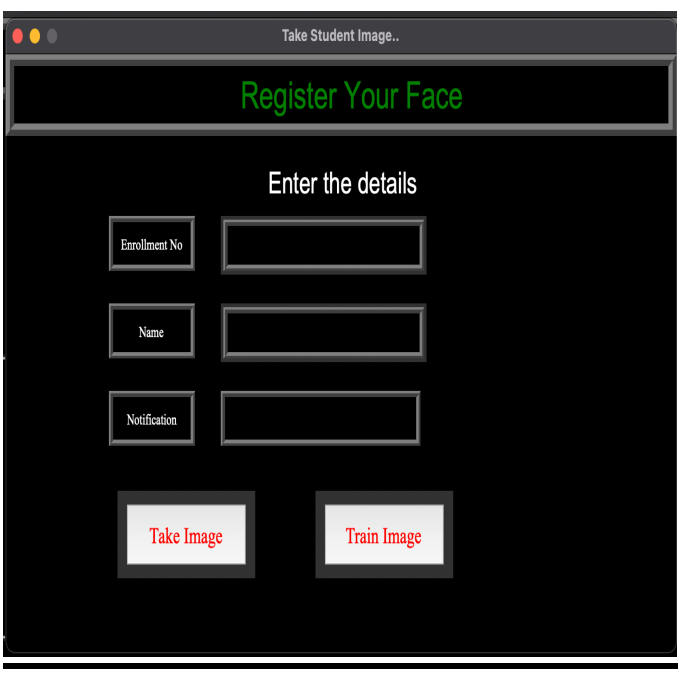
Once the model is trained and saved as *trainer.yml*, it can be loaded and used for recognizing faces in real-time applications. The model typically takes an input image, processes it, and generates a unique output vector for each face it recognizes. These output vectors can then be compared to previously saved vectors in a database to identify the individual.

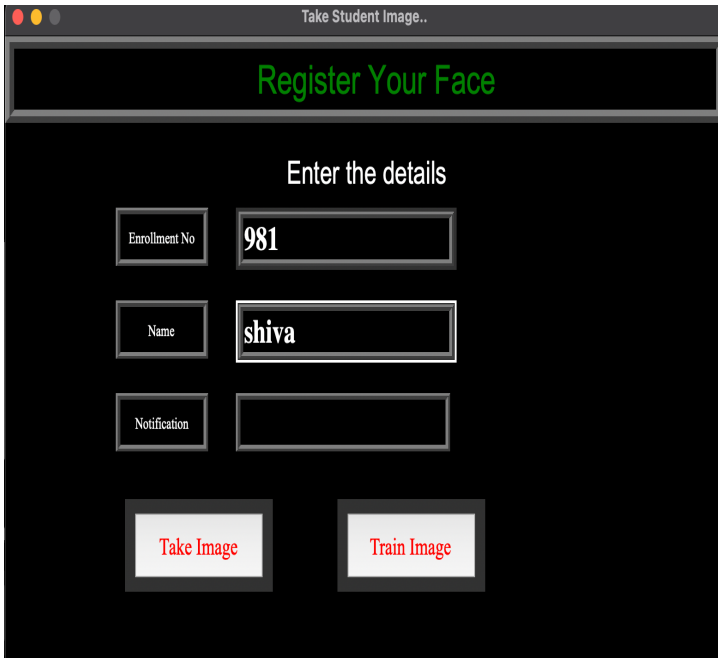
PROJECT OUTPUT SNAPS:

Running attendance.py:



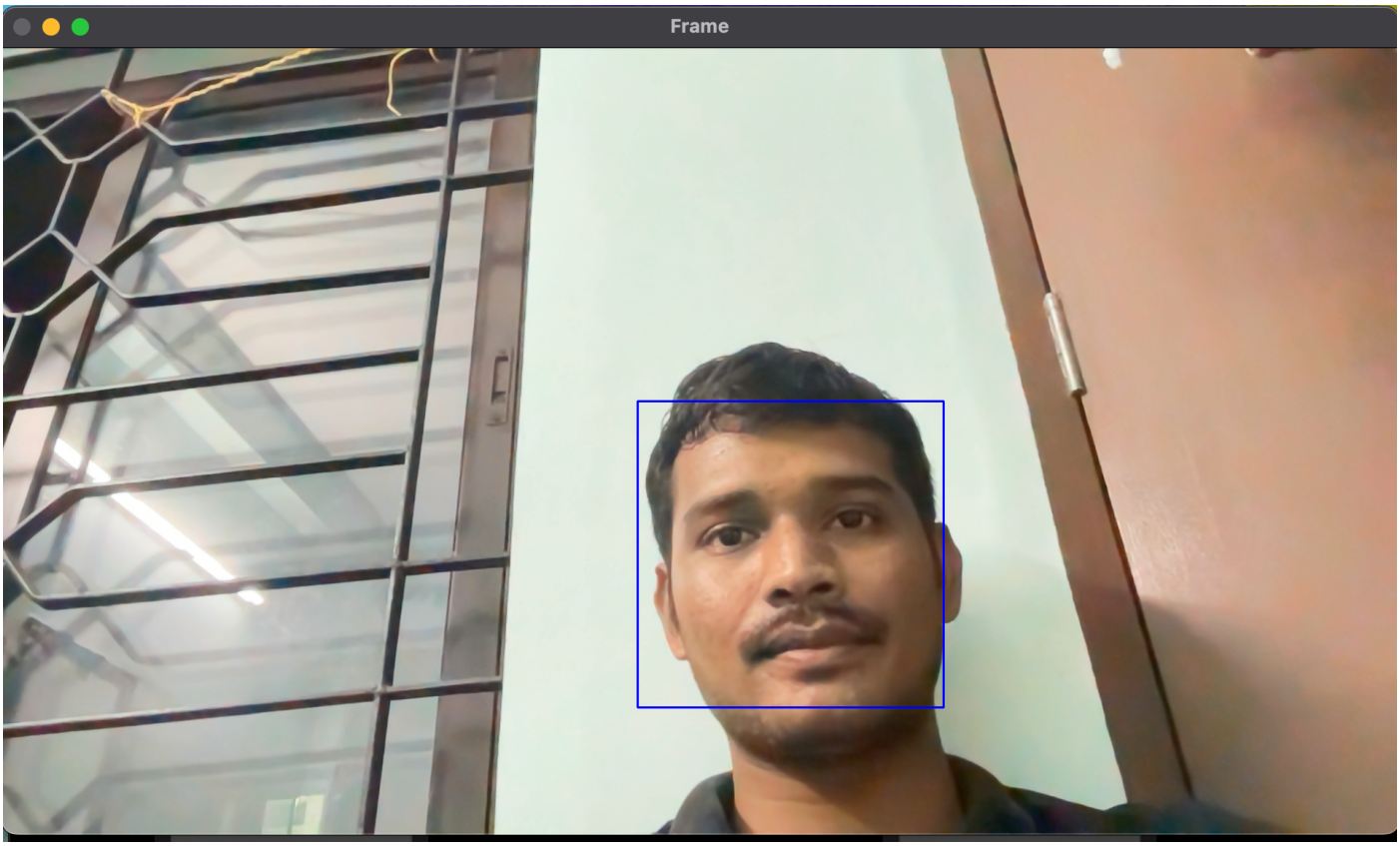
Clicking on Register a new student:



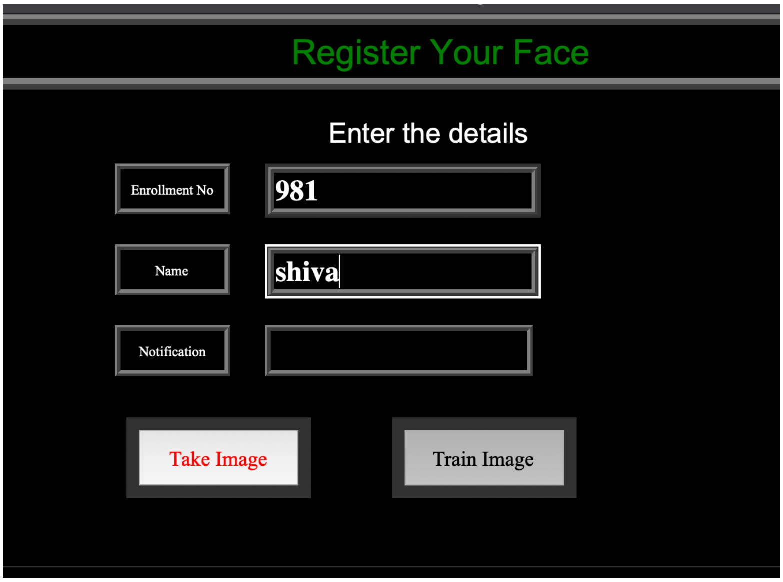


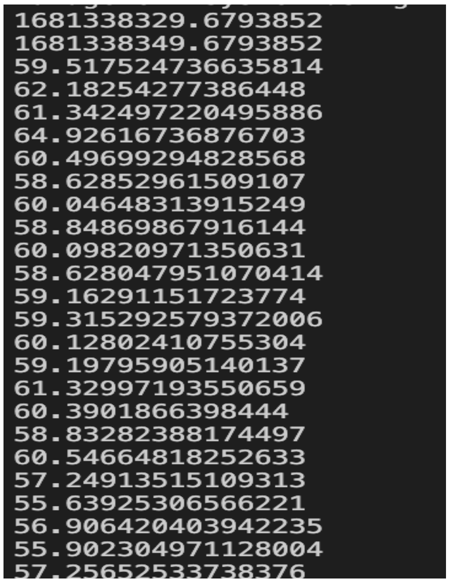
After entering Enrollment No & Name:

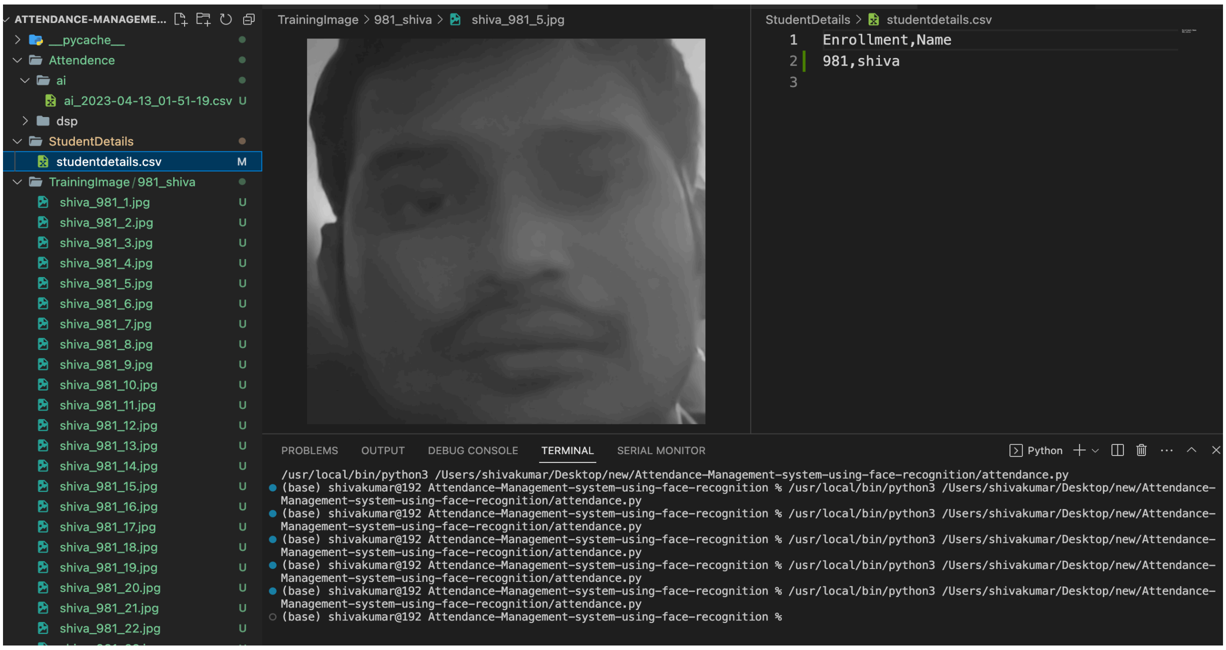
Takes 101 photos of us and creates a folder using the inputs of enrolment number and name and stores it:



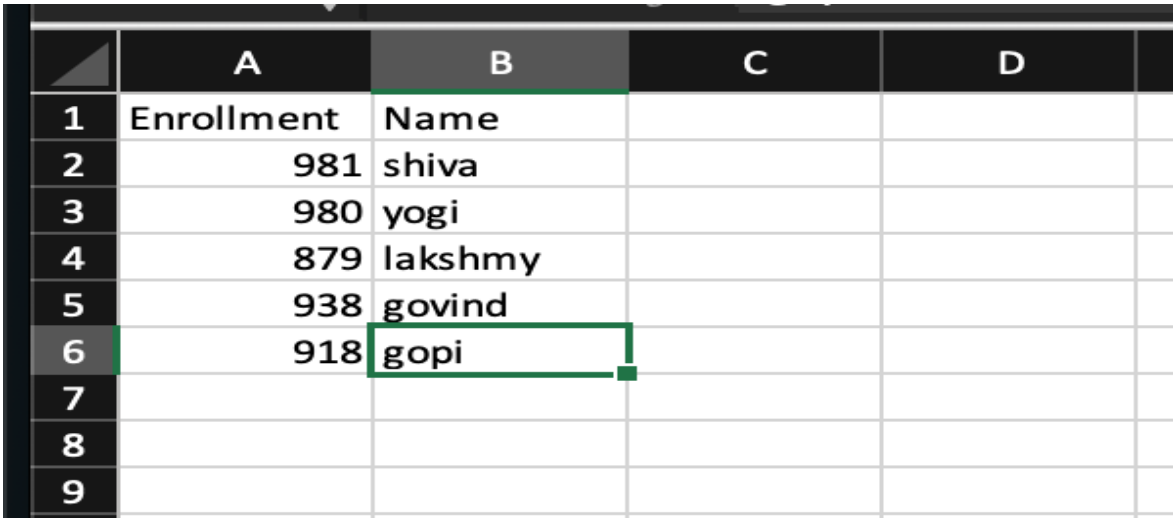
Clicking on train image takes the 101 images and endcodes them and stores them in trainer.yml:



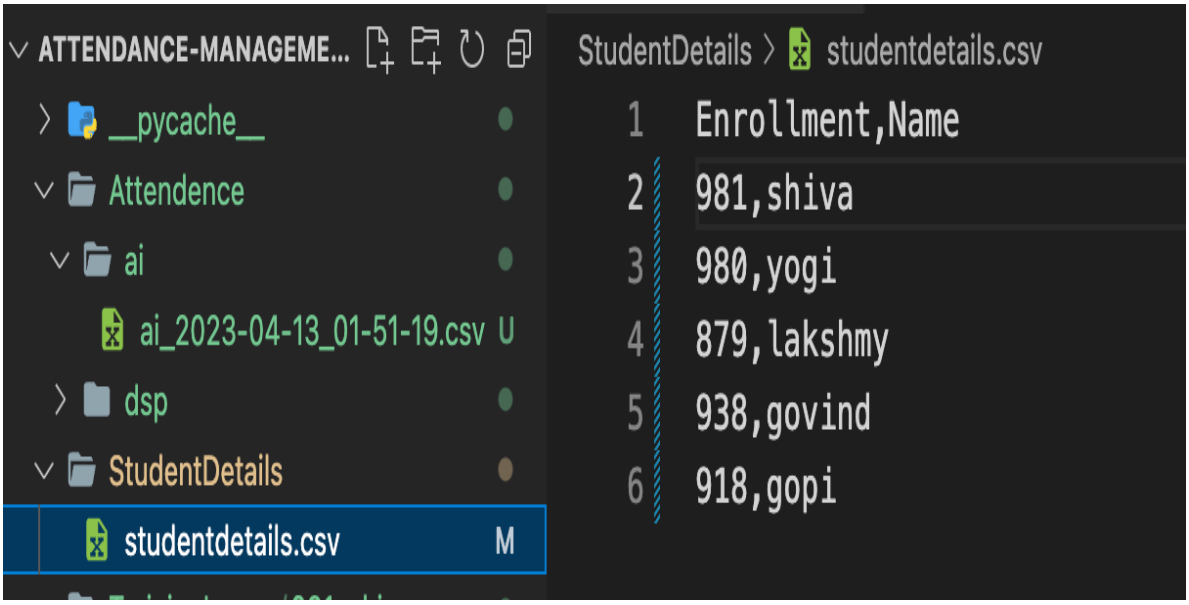




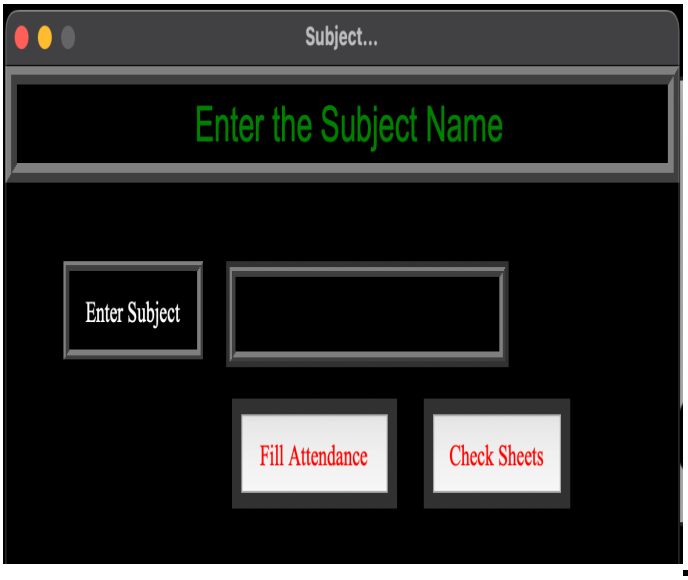
Csv automatically storing student database:

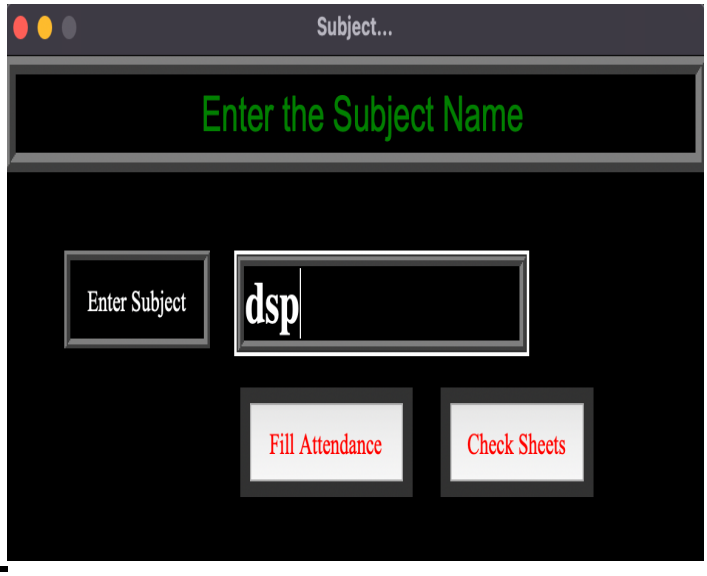


	A	B	C	D
1	Enrollment	Name		
2	981	shiva		
3	980	yogi		
4	879	lakshmy		
5	938	govind		
6	918	gopi		
7				
8				
9				

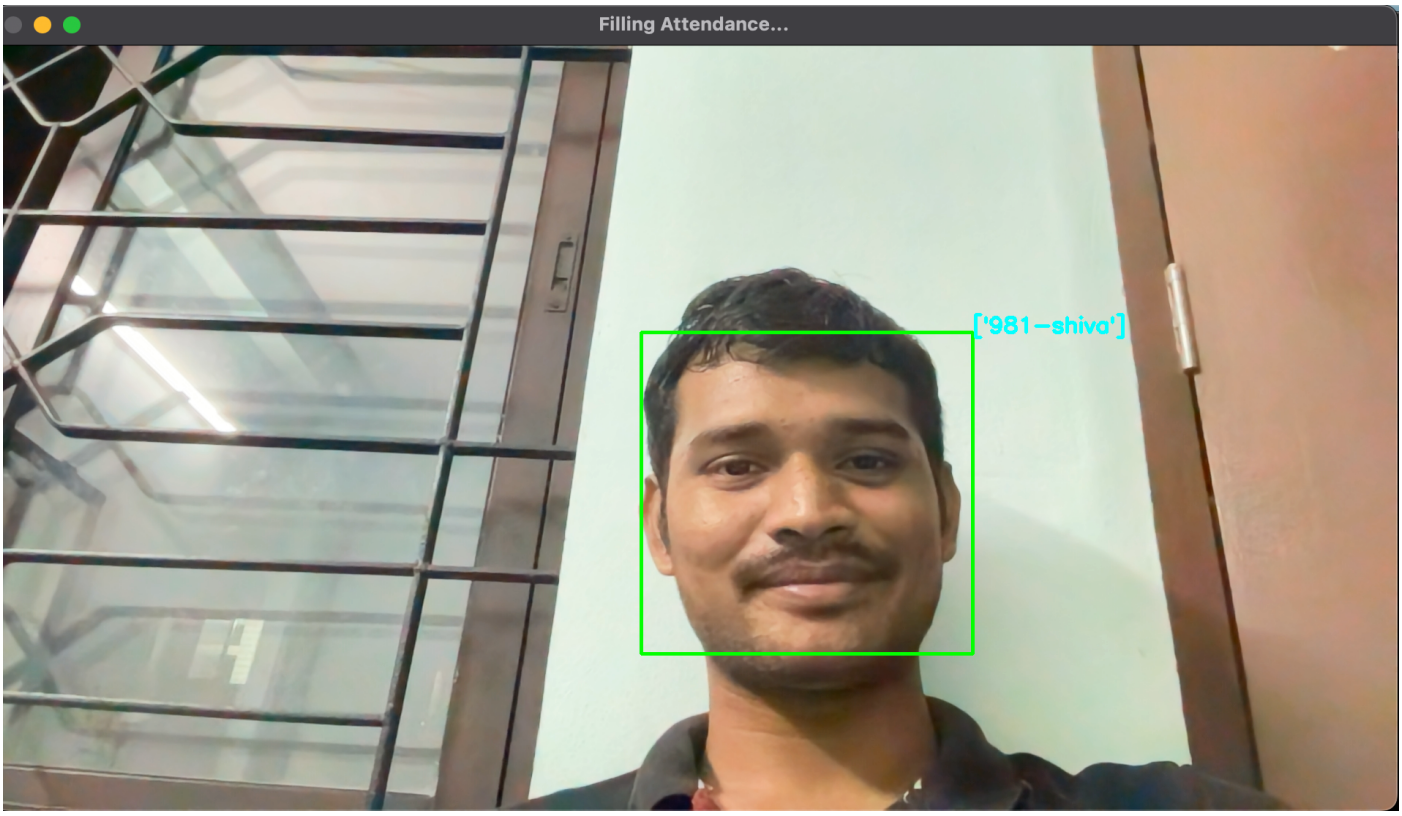


Clicking on Fill Ateendence and Entring Subject name allows you to take Attendence:

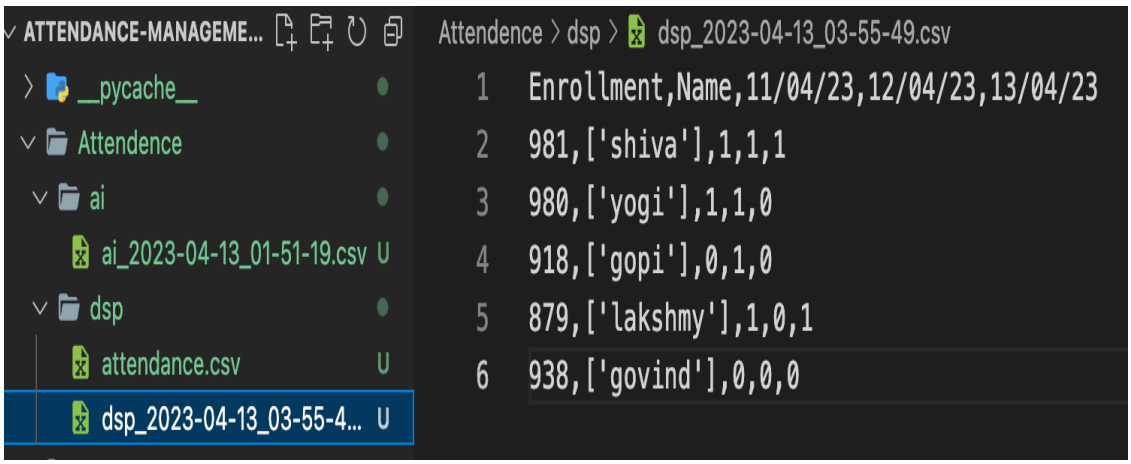




Taking Attendance:



	A	B	C	D	E	F
1	Enrollment	Name	11/04/23	12/04/23	13/04/23	
2	981	['shiva']	1	1	1	
3	980	['yogi']	1	1	0	
4	918	['gopi']	0	1	0	
5	879	['lakshmy']	1	0	1	
6	938	['govind']	0	0	0	
7						



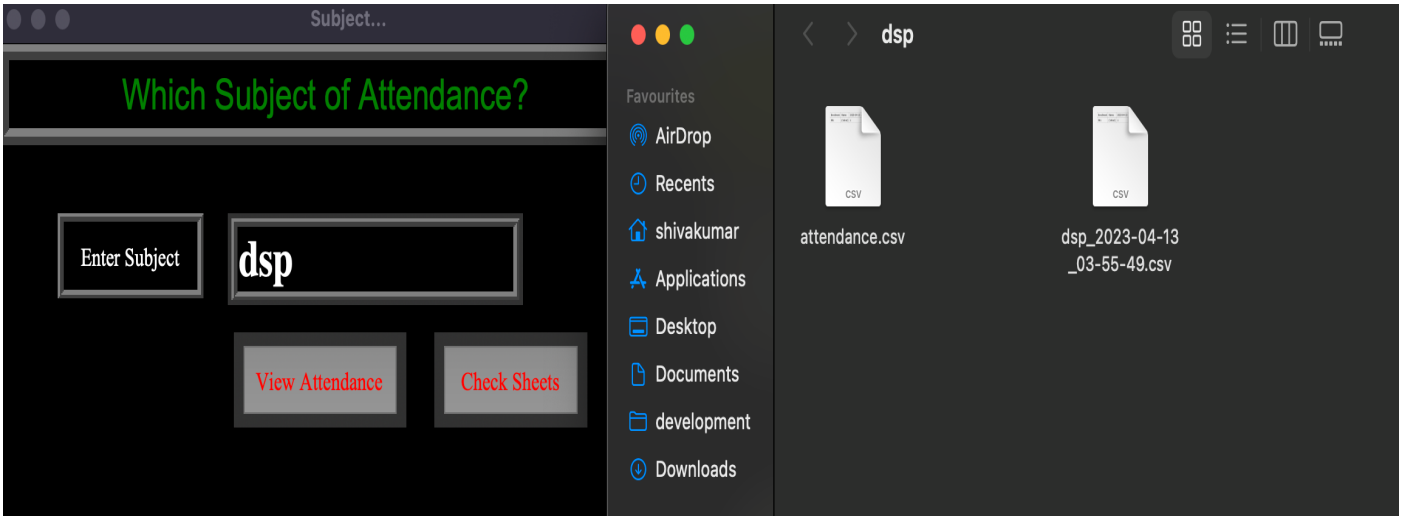
Checking Attendance:

Entering subject name Displays Attendence:

Attendance of dsp

Enrollment	Name	11/04/23	12/04/23	13/04/23	Attendance
981	['shiva']	1	1	1	100%
980	['yogi']	1	1	0	67%
918	['gopi']	0	1	0	33%
879	['lakshmy']	1	0	1	67%
938	['govind']	0	0	0	0%

The Check Sheets feature enables you to verify if the subject is present in the database:



Code Editor /IDE Used: Visual Studio Code

How to run the project:

Please follow the following steps to run the project:

- Download or clone the Repository to your device.
- Open the command prompt and type "**pip install -r requirements.txt**" to install the required packages for the project.
- Create folders named as “TrainingImage” and “Attendence” respectively.
- Open "**attendance.py**" and "**automaticAttendance.py**" and change all the path according to your system.
- Create the folders for subjects we require to take attendance the "Attendence" folder.
- Run "**attendance.py**" file to register your face so that the system can identify you.

The project workflow can be described as follows:

- Click on "Register New Student" and a small window will pop up where you have to enter your ID and name. Then click on the "Take Image" button to capture your face. A camera window will pop up and take up to 100 images (you can change the number of images it can take) and store them in the "**TrainingImage**" folder.

- Click on the "Train Image" button to train the model and convert all the images into a numeric format so that the computer can understand them. The system will take some time to complete the training, depending on your system.

- After training the model, click on "**Automatic Attendance**". Enter the subject name, and the system will fill in attendance by recognizing your face using the trained model. It will create a separate .**csv** file for every subject you enter.

- You can view the attendance record in tabular format by clicking on the "**View Attendance"** button.

Requirements of the system:

- Python 3.6+

- Ram minimum : 8gb

- 4gb Graphic Card

- **your Operating System should Support the required modules.**

    1. Opencv (pip install opencv-python)
    2. Tkinter (Available in python)
    3. PIL (pip install Pillow)
    4. Pandas. (pip install pandas)
    5. Numpy. (pip install numpy)
    6. Pillow (pip install Pillow)

- haarcascade_frontalface_default.xml

1. The folder "haarcascades" contains pre-trained classifiers for detecting specific types of objects such as faces (frontal and profile) and pedestrians.

Some of the classifiers may have unique licenses that require further investigation.
https://github.com/opencv/opencv/tree/master/data/haarcascades

- The quality of images is crucial as it can impact the accuracy of the system by introducing noise.