STUDENT ATTENDANCE SYSTEM

DISSERTATION SUBMITTED
IN
PARTIAL FULFILLMENT OF THE REQUIREMENTS
OF THE DEGREE OF

BACHELOR OF TECHNOLOGY

IN
COMPUTER SCIENCE & ENGINEERING

BY

ANUSIKA RANI – AJU/201336
UPLOW KUMAR PANDEY - AJU/200274
RIYA KUMARI - AJU/200597

Under the supervision of
Dr. Nidhi Dua

School of Engineering & I.T,
ARKA JAIN University, Jharkhand
(2020-2024)

ARKA JAIN University
Jharkhand

NAAC GRADE A
ACCREDITED UNIVERSITY

CERTIFICATE

15th April 2024

This is to certify that the project entitled "**Student Attendance System**" has been submitted to the Department of Computer Science & IT, ARKA JAIN UNIVERSITY, Jharkhand for the fulfillment of the requirement for the award of the degree of "Bachelor of Technology in Computer Science & Engineering" by following student of final year B. Tech (Computer Science & Engineering).

Anusika Rani – AJU/201336
Uplow Kumar Pandey – AJU/200274
Riya Kumari – AJU/200597

Dr. Nidhi Dua
(Project Guide)

Dr. Ashwini Kumar
(Asst. Dean)

ARKA JAIN UNIVERSITY JHARKHAND

DECLARATION BY THE CANDIDATE

I hereby declare that the project report entitled "**Student Attendance System**" submitted by us to **ARKA JAIN University**, Jharkhand in partial fulfillment of the requirement for the award of the degree of **Bachelor of Technology** in **Computer Science & Engineering** is a record of bonafide project work carried out by us under the guidance of Dr. Nidhi Dua, I further declare that the work reported in this project has not been submitted and will not be submitted, either in part or in full, for the award of any other degree in this university or any other institute or university. We will be solely responsible if any kind of plagiarism is found.

Date: - 15/04/24

Arka Jain University

Anusika Rani – AJU/201336

Uplow Kumar Pandey – AJU/200274

Riya Kumari – AJU/200597

iii

ACKNOWLEDGMENT

We would like to share our sincere gratitude to all those who helped us in the completion of this project. During the work, we faced many challenges due to our lack of knowledge and experience but these people helped us to get over all the difficulties and in completion of our idea of a shaped sculpture.

We would like to thank Dr. Ashwini Kumar sir for his governance and guidance, because of which our whole team was able to learn the minute aspects of a project work.

We would also like to thank our Project Coordinator Dr. Nidhi Dua for her continuous support and monitoring during the project work.

Finally, we would like to thank the management of ARKA JAIN University for providing us with such an opportunity to learn from their experiences.

All of our team is thankful to all the faculties and staff of the Department of Computer Science & IT, AJU, Jharkhand, for their help and support towards this project and our team.

We are also thankful to our whole class and to our parents who have inspired us to face all the challenges and win all the hurdles in life.

April 15th, 2024
Arka Jain University

Anusika Rani - (AJU/201336)
Uplow Kumar -(AJU/200274)
Riya Kumari - (AJU/200597)

ABSTRACT

In this comprehensive review paper, an in-depth exploration of existing student attendance systems powered by Python and integrated with face recognition technology is undertaken. The significance of efficient attendance management in educational institutions, ranging from academic to administrative spheres, necessitates the adoption of innovative and automated solutions.

Traditionally, attendance tracking has been a manual and time-consuming process, often prone to errors and inconsistencies. The advent of Python-powered face recognition systems has revolutionized this landscape by offering a reliable and automated alternative. The utilization of facial recognition technology not only enhances accuracy but also streamlines the overall attendance recording process.

The student attendance system's architecture and functionality are meticulously studied in this review, delving into the intricacies of Python programming and its integration with facial recognition algorithms. Considerations such as system sizing, user requirements, and the contextual relevance of the technology are thoroughly examined to provide a comprehensive understanding of the design parameters influencing the effectiveness of the student attendance system. The existing literature is critically reviewed, offering valuable insights into the evolution, implementation, and prospects of Python-powered face recognition systems for automated student attendance.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

1.1 Introduction:

Our Attendance System, powered by Python, stands as a solution to streamline the attendance tracking process. By seamlessly integrating Python's capabilities, our system revolutionizes how organizations manage attendance, offering a sophisticated approach to conventional methods.

Utilizing the camera of any device, our system takes attendance tracking to the next level by providing not only smooth but also highly accurate monitoring of individuals in various settings. This ensures that attendance data is not only reliable but also easily accessible, creating a seamless experience for users and administrators alike.

Automation takes center stage in our design philosophy, allowing organizations to effortlessly manage attendance records. This means less manual effort and more efficiency, as our system takes care of the tedious tasks, freeing up valuable time for administrators to focus on other essential responsibilities.

Our user-friendly interface is the gateway to a world of streamlined attendance management. Paired with strong backend algorithms, it guarantees real-time data collection, effectively minimizing the administrative workload. This ensures that administrators can make data-driven decisions with the latest and most accurate attendance information at their convenience.

1. SOFTWARE REQUIRED

2.1 Software required for development

System Requirement

- RAM: - 16.0 GB,
- PROCESSOR: - 12th Gen Intel® Core™ i5-1235U 1.30GHz
- SYSTEM TYPE: - 64-bit Operating System,
- EDITION: - Windows 11
- PLATFORM: - Python 3.8v, Xampp, Visual Studio Code

Algorithm

Face recognition algorithm based on a deep learning CNN module (convolutional neural network), is used for face recognition, verification, and clustering.

Front End

To design the interface **HTML** is used. Giving static websites a design to obtain a list of students' present date-wise.

Back End

At the backend, to store entered data, a **database** in **SQL Lite** is created locally, using **SQL** queries.

Python Libraries used

1. NumPy: Numerical computing library.
- Enables efficient array operations and mathematical functions.
- Widely used in scientific computing, machine learning, and data analysis.

2. Pandas: Data manipulation and analysis.
- Provides Data Frame structure for easy handling of structured data.
- Essential for tasks such as data cleaning, exploration, and transformation.

3. Dlib: Toolkit for machine learning and computer vision.
- Includes facial recognition and object detection capabilities.

- Utilized for developing applications in image and video processing.

4. os: Operating system interface.

- Facilitates interaction with the underlying operating system.
- Used for tasks like file and directory manipulation, environment variables, etc.

5. cv2 (OpenCV): Computer vision library.

- Offers a vast array of tools for image and video processing.
- Applied in image filtering, feature detection, and video analysis.

6. time: Time-related functions.

- Used for measuring code execution time and creating time delays.
- Enables working with timestamps and measuring elapsed time.

7. logging: Logging messages from applications.

- Facilitates systematic recording of events for debugging and analysis.
- Configurable to capture messages at different severity levels.

8. sqlite3: SQLite database interface.

- Provides a lightweight, embedded database solution.
- Enables the creation and management of local databases.

9. datetime: Date and time manipulation.

- Offers classes for representing dates, times, and intervals.
- Supports formatting and parsing of dates and times.

10. shutil: File operations.

- Simplifies file operations like copying, moving, and deleting files.
- Useful for managing directories and handling file-related tasks.

11. tkinter: GUI (Graphical User Interface) toolkit.

- Enables the creation of desktop GUI applications.
- Includes a variety of widgets for building interactive user interfaces.

2.2 User requirement for using the application:

Since all the codebase and database is done locally on the system, we insist on using the system with similar or higher specification as the system we are using originally for development.

System Requirement

- RAM: - 16.0 GB,
- PROCESSOR: - 12th Gen Intel® Core™ i5-1235U 1.30GHz
- SYSTEM TYPE: - 64-bit Operating System,
- EDITION: - Windows 11
- PLATFORM: - Python 3.8v, Xampp, Visual Studio Code

2. OBJECTIVE

3.1 Objective: -

Automating Attendance Tracking: Implementing a system that automates the attendance tracking process using advanced technologies. Utilizing CNN and facial recognition to accurately and efficiently identify and record student presence.

Increasing Accuracy and Reliability: Enhancing accuracy by leveraging facial recognition to minimize errors in attendance records. Ensuring reliable attendance data for administrative and academic purposes.

Streamlining Administrative Processes: Providing administrators with a streamlined system for managing attendance data. Automating attendance reporting and reducing the administrative burden on faculty.

Improving Time Efficiency: Implementing a time-efficient system that reduces the time taken for manual attendance taking. Enabling quick and real-time updates on student attendance for teachers and administrators.

Enhancing Security Measures: Integrating facial recognition technology to enhance the security of the attendance system. Implementing measures to protect student information and attendance records.

In summary, the student attendance system aims to revolutionize traditional attendance tracking by combining cutting-edge technologies, ultimately addressing challenges related to accuracy, efficiency, and security within the educational landscape.

3.2Feasibility study:

Feasibility Study for the Student Attendance System involves evaluating the viability of the project in terms of its technical, economic, and operational aspects. Here are some key considerations for each of these aspects:

Technical Feasibility:

The technical feasibility of the Student Attendance System involves assessing its capacity to utilize advanced technologies, specifically Convolutional Neural Networks (CNN) and facial recognition. The system must be capable of accurately and efficiently tracking student attendance, integrate seamlessly with existing educational technology infrastructure, and ensure data security. Key considerations include the ability to process facial recognition data, implement real-time updates, and provide a user-friendly interface.

Economic Feasibility:

Economically, the project involves evaluating the financial viability of implementing the Student Attendance System. Initial investment requirements include software development, integration of facial recognition technology, and operational expenses. The system is expected to generate revenue by reducing administrative costs associated with manual attendance tracking.

Operational Feasibility:

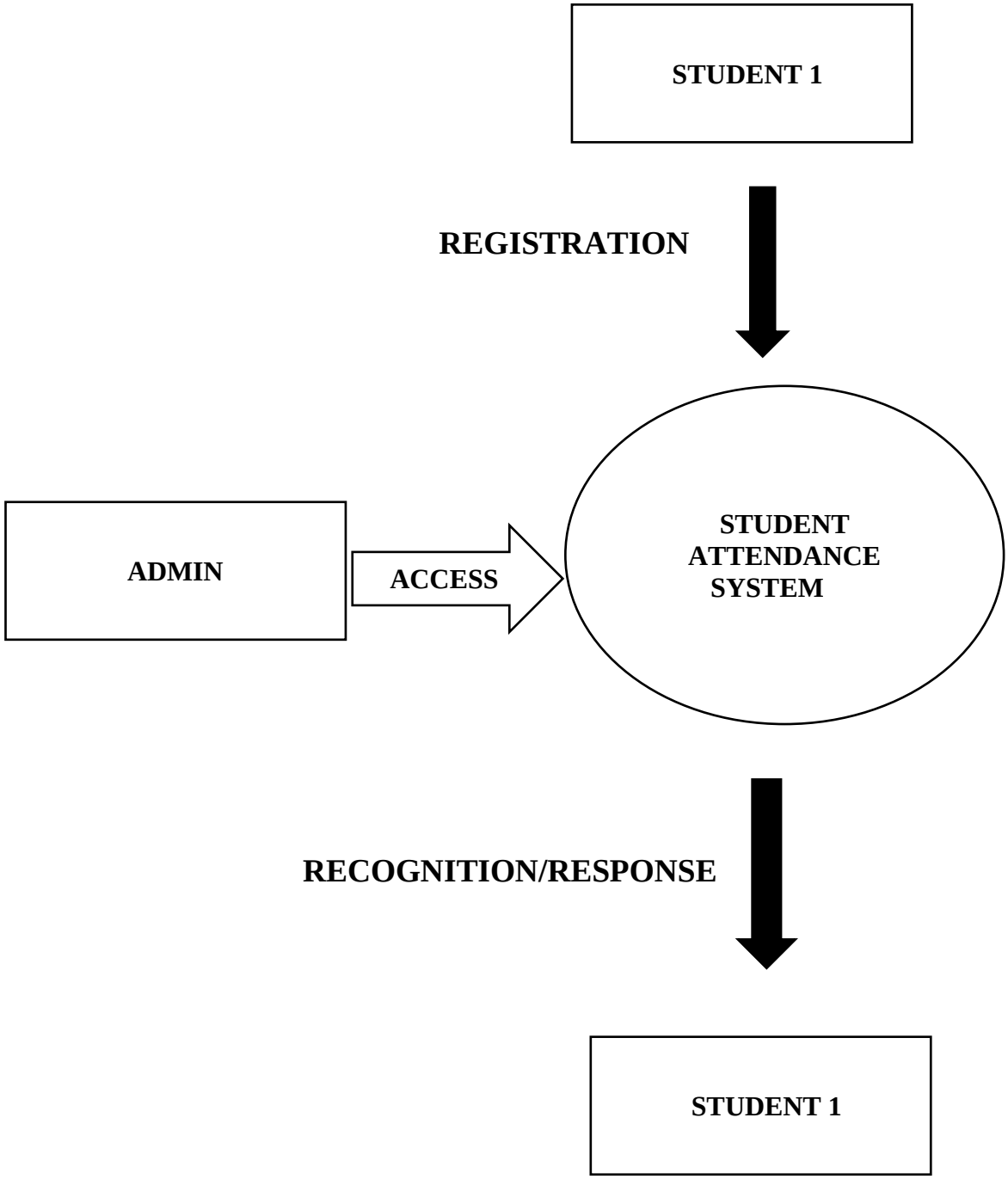
The operational feasibility of the Student Attendance System focuses on its ability to function efficiently within educational settings. The system requires a team of experts, including those skilled in technology, education, and potentially legal compliance, to manage its operations effectively. Establishing protocols for data privacy, ensuring ease of use for educators and students, and complying with relevant regulations are crucial aspects of operational feasibility.

Conclusion:

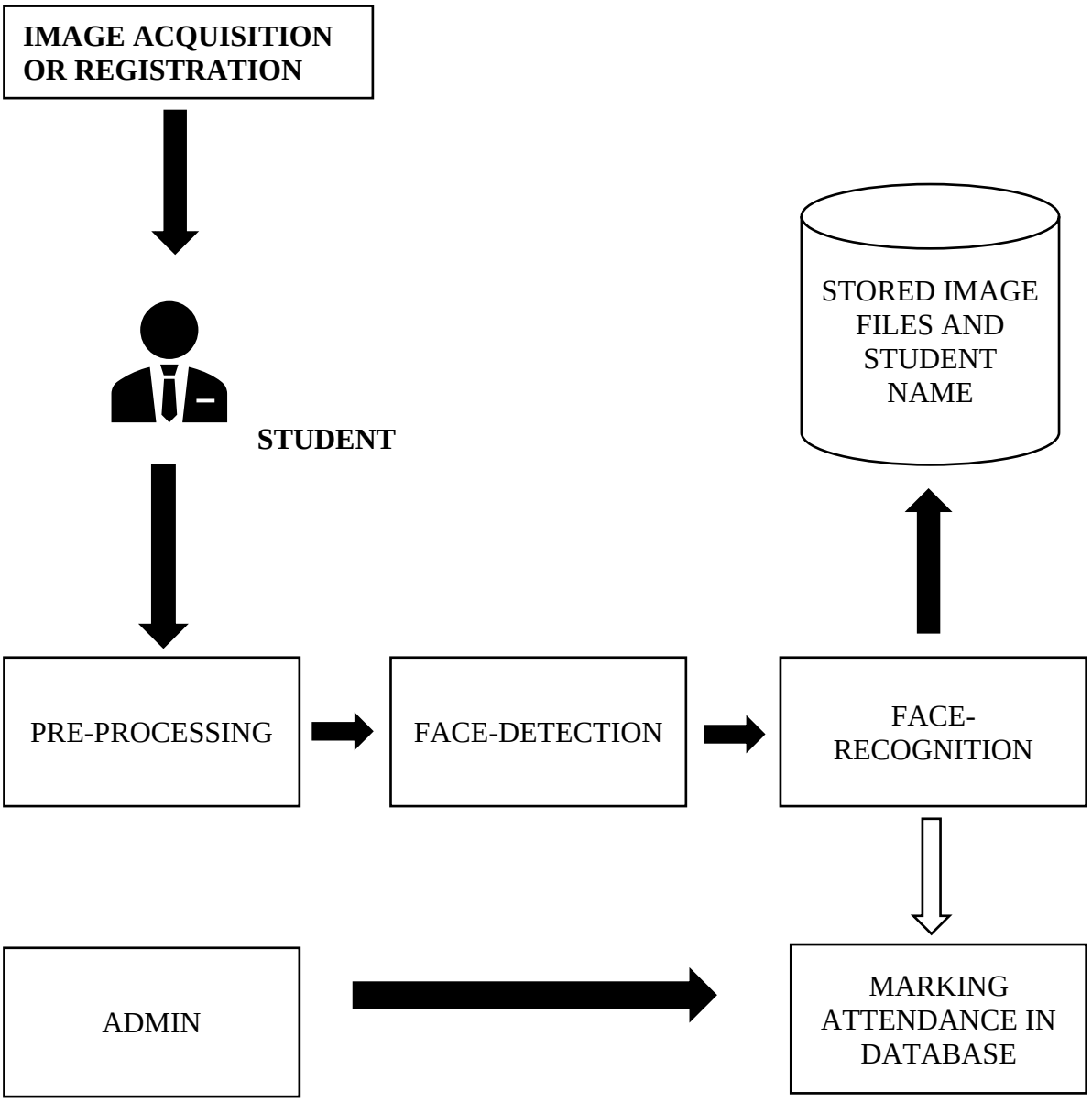
In conclusion, the feasibility study for the Student Attendance System suggests that the project is technically sound, economically viable, and operationally feasible.

3. DATA FLOW DIAGRAM

4.1DFD LEVEL – 0



4.2DFD LEVEL -FINAL



5. CODES

FACE REGISTRATION WINDOW

```
import dlib
import numpy as np
import cv2
import os
import shutil
import time
import logging
import tkinter as tk
from tkinter import font as tkFont
from PIL import Image, ImageTk

# Use frontal face detector of Dlib
detector = dlib.get_frontal_face_detector()


class Face_Register:
    def __init__(self):

        self.current_frame_faces_cnt = 0  #  cnt for counting faces in current frame
        self.existing_faces_cnt = 0  # cnt for counting saved faces
        self.ss_cnt = 0  #  cnt for screen shots

        # Tkinter GUI
        self.win = tk.Tk()
        self.win.title("Face Register")

        # PLease modify window size here if needed
        self.win.geometry("1000x500")

        # GUI left part
        self.frame_left_camera = tk.Frame(self.win)
        self.label = tk.Label(self.win)
        self.label.pack(side=tk.LEFT)
```

```
        self.frame_left_camera.pack()

        # GUI right part
        self.frame_right_info = tk.Frame(self.win)
        self.label_cnt_face_in_database = tk.Label(self.frame_right_info,
text=str(self.existing_faces_cnt))
        self.label_fps_info = tk.Label(self.frame_right_info, text="")
        self.input_name = tk.Entry(self.frame_right_info)
        self.input_name_char = ""
        self.label_warning = tk.Label(self.frame_right_info)
        self.label_face_cnt = tk.Label(self.frame_right_info, text="Faces in current
frame: ")
        self.log_all = tk.Label(self.frame_right_info)

        self.font_title = tkFont.Font(family='Helvetica', size=20, weight='bold')
        self.font_step_title = tkFont.Font(family='Helvetica', size=15, weight='bold')
        self.font_warning = tkFont.Font(family='Helvetica', size=15, weight='bold')

        self.path_photos_from_camera = "data/data_faces_from_camera/"
        self.current_face_dir = ""
        self.font = cv2.FONT_ITALIC

        # Current frame and face ROI position
        self.current_frame = np.ndarray
        self.face_ROI_image = np.ndarray
        self.face_ROI_width_start = 0
        self.face_ROI_height_start = 0
        self.face_ROI_width = 0
        self.face_ROI_height = 0
        self.ww = 0
        self.hh = 0

        self.out_of_range_flag = False
        self.face_folder_created_flag = False

        # FPS
        self.frame_time = 0
```

```
        self.frame_start_time = 0
        self.fps = 0
        self.fps_show = 0
        self.start_time = time.time()

        self.cap = cv2.VideoCapture(0)  # Get video stream from camera

        # self.cap = cv2.VideoCapture("test.mp4")   # Input local video

    #  Delete old face folders
    def GUI_clear_data(self):
        #  "/data_faces_from_camera/person_x/"...
        folders_rd = os.listdir(self.path_photos_from_camera)
        for i in range(len(folders_rd)):
            shutil.rmtree(self.path_photos_from_camera + folders_rd[i])
        if os.path.isfile("data/features_all.csv"):
            os.remove("data/features_all.csv")
        self.label_cnt_face_in_database['text'] = "0"
        self.existing_faces_cnt = 0
        self.log_all["text"] = "Face images and `features_all.csv` removed!"

    def GUI_get_input_name(self):
        self.input_name_char = self.input_name.get()
        self.create_face_folder()
        self.label_cnt_face_in_database['text'] = str(self.existing_faces_cnt)

    def GUI_info(self):
        tk.Label(self.frame_right_info,
                 text="Face register",
                 font=self.font_title).grid(row=0, column=0, columnspan=3,
sticky=tk.W, padx=2, pady=20)

        tk.Label(self.frame_right_info, text="FPS: ").grid(row=1, column=0,
sticky=tk.W, padx=5, pady=2)
        self.label_fps_info.grid(row=1, column=1, sticky=tk.W, padx=5, pady=2)

        tk.Label(self.frame_right_info, text="Faces in database: ").grid(row=2,
column=0, sticky=tk.W, padx=5, pady=2)
```

```
        self.label_cnt_face_in_database.grid(row=2, column=1, sticky=tk.W,
padx=5, pady=2)

        tk.Label(self.frame_right_info,
                 text="Faces in current frame: ").grid(row=3, column=0, columnspan=2,
sticky=tk.W, padx=5, pady=2)
        self.label_face_cnt.grid(row=3, column=2, columnspan=3, sticky=tk.W,
padx=5, pady=2)

        self.label_warning.grid(row=4, column=0, columnspan=3, sticky=tk.W,
padx=5, pady=2)

        # Step 1: Clear old data
        tk.Label(self.frame_right_info,
                 font=self.font_step_title,
                 text="Step 1: Clear face photos").grid(row=5, column=0,
columnspan=2, sticky=tk.W, padx=5, pady=20)



        # Step 2: Input name and create folders for face
        tk.Label(self.frame_right_info,
                 font=self.font_step_title,
                 text="Step 2: Input name").grid(row=7, column=0, columnspan=2,
sticky=tk.W, padx=5, pady=20)

        tk.Label(self.frame_right_info, text="Name: ").grid(row=8, column=0,
sticky=tk.W, padx=5, pady=0)
        self.input_name.grid(row=8, column=1, sticky=tk.W, padx=0, pady=2)

        tk.Button(self.frame_right_info,
                  text='Input',
                  command=self.GUI_get_input_name).grid(row=8, column=2, padx=5)

        # Step 3: Save current face in frame
        tk.Label(self.frame_right_info,
                 font=self.font_step_title,
                 text="Step 3: Save face image").grid(row=9, column=0,
columnspan=2, sticky=tk.W, padx=5, pady=20)
```

```
        tk.Button(self.frame_right_info,
                  text='Save current face',
                  command=self.save_current_face).grid(row=10, column=0,
columnspan=3, sticky=tk.W)

        # Show log in GUI
        self.log_all.grid(row=11, column=0, columnspan=20, sticky=tk.W, padx=5,
pady=20)

        self.frame_right_info.pack()

    # Mkdir for saving photos and csv
    def pre_work_mkdir(self):
        # Create folders to save face images and csv
        if os.path.isdir(self.path_photos_from_camera):
            pass
        else:
            os.mkdir(self.path_photos_from_camera)

    # Start from person_x+1
    def check_existing_faces_cnt(self):
        if os.listdir("data/data_faces_from_camera/"):
            # Get the order of latest person
            person_list = os.listdir("data/data_faces_from_camera/")
            person_num_list = []
            for person in person_list:
                person_order = person.split('_')[1].split('_')[0]
                person_num_list.append(int(person_order))
            self.existing_faces_cnt = max(person_num_list)

        # Start from person_1
        else:
            self.existing_faces_cnt = 0

    # Update FPS of Video stream
    def update_fps(self):
        now = time.time()
```

```
        #  Refresh fps per second
        if str(self.start_time).split(".")[0] != str(now).split(".")[0]:
            self.fps_show = self.fps
        self.start_time = now
        self.frame_time = now - self.frame_start_time
        self.fps = 1.0 / self.frame_time
        self.frame_start_time = now

        self.label_fps_info["text"] = str(self.fps.__round__(2))

    def create_face_folder(self):
        #  Create the folders for saving faces
        self.existing_faces_cnt += 1
        if self.input_name_char:
            self.current_face_dir = self.path_photos_from_camera + \
                                    "person_" + str(self.existing_faces_cnt) + "_" + \
                                    self.input_name_char
        else:
            self.current_face_dir = self.path_photos_from_camera + \
                                    "person_" + str(self.existing_faces_cnt)
        os.makedirs(self.current_face_dir)
        self.log_all["text"] = "\"" + self.current_face_dir + "/\" created!"
        logging.info("\n%-40s %s", "Create folders:", self.current_face_dir)

        self.ss_cnt = 0  #  Clear the cnt of screen shots
        self.face_folder_created_flag = True  # Face folder already created

    def save_current_face(self):
        if self.face_folder_created_flag:
            if self.current_frame_faces_cnt == 1:
                if not self.out_of_range_flag:
                    self.ss_cnt += 1
                    #  Create blank image according to the size of face detected
                    self.face_ROI_image = np.zeros((int(self.face_ROI_height * 2),
self.face_ROI_width * 2, 3),
                                                   np.uint8)
                    for ii in range(self.face_ROI_height * 2):
```

```
                    for jj in range(self.face_ROI_width * 2):
                        self.face_ROI_image[ii][jj] =
self.current_frame[self.face_ROI_height_start - self.hh + ii][
                            self.face_ROI_width_start - self.ww + jj]
                self.log_all["text"] = "\"" + self.current_face_dir + "/img_face_" +
str(
                    self.ss_cnt) + ".jpg\"" + " saved!"
                self.face_ROI_image = cv2.cvtColor(self.face_ROI_image,
cv2.COLOR_BGR2RGB)

                cv2.imwrite(self.current_face_dir + "/img_face_" + str(self.ss_cnt) +
".jpg", self.face_ROI_image)
                logging.info("%-40s %s/img_face_%s.jpg", "Save into：",
                             str(self.current_face_dir), str(self.ss_cnt) + ".jpg")
            else:
                self.log_all["text"] = "Please do not out of range!"
        else:
            self.log_all["text"] = "No face in current frame!"
    else:
        self.log_all["text"] = "Please run step 2!"

def get_frame(self):
    try:
        if self.cap.isOpened():
            ret, frame = self.cap.read()
            frame = cv2.resize(frame, (640,480))
            return ret, cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    except:
        print("Error: No video input!!!")

#  Main process of face detection and saving
def process(self):
    ret, self.current_frame = self.get_frame()
    faces = detector(self.current_frame, 0)
    # Get frame
    if ret:
        self.update_fps()
        self.label_face_cnt["text"] = str(len(faces))
```

```
            #  Face detected
            if len(faces) != 0:
                #   Show the ROI of faces
                for k, d in enumerate(faces):
                    self.face_ROI_width_start = d.left()
                    self.face_ROI_height_start = d.top()
                    #  Compute the size of rectangle box
                    self.face_ROI_height = (d.bottom() - d.top())
                    self.face_ROI_width = (d.right() - d.left())
                    self.hh = int(self.face_ROI_height / 2)
                    self.ww = int(self.face_ROI_width / 2)

                    # If the size of ROI > 480x640
                    if (d.right() + self.ww) > 640 or (d.bottom() + self.hh > 480) or
(d.left() - self.ww < 0) or (
                            d.top() - self.hh < 0):
                        self.label_warning["text"] = "OUT OF RANGE"
                        self.label_warning['fg'] = 'red'
                        self.out_of_range_flag = True
                        color_rectangle = (255, 0, 0)
                    else:
                        self.out_of_range_flag = False
                        self.label_warning["text"] = ""
                        color_rectangle = (255, 255, 255)
                    self.current_frame = cv2.rectangle(self.current_frame,
                                                       tuple([d.left() - self.ww, d.top() - self.hh]),
                                                       tuple([d.right() + self.ww, d.bottom() +
self.hh]),
                                                       color_rectangle, 2)
            self.current_frame_faces_cnt = len(faces)

            # Convert PIL.Image.Image to PIL.Image.PhotoImage
            img_Image = Image.fromarray(self.current_frame)
            img_PhotoImage = ImageTk.PhotoImage(image=img_Image)
            self.label.img_tk = img_PhotoImage
            self.label.configure(image=img_PhotoImage)
```

```
        # Refresh frame
        self.win.after(20, self.process)

    def run(self):
        self.pre_work_mkdir()
        self.check_existing_faces_cnt()
        self.GUI_info()
        self.process()
        self.win.mainloop()


def main():
    logging.basicConfig(level=logging.INFO)
    Face_Register_con = Face_Register()
    Face_Register_con.run()


if __name__ == '__main__':
    main()
```

FEATURE EXTRACTION CODE

```
# Extract features from images and save into "features_all.csv"

import os
import dlib
import csv
import numpy as np
import logging
import cv2

#  Path of cropped faces
path_images_from_camera = "data/data_faces_from_camera/"

#  Use frontal face detector of Dlib
detector = dlib.get_frontal_face_detector()

#  Get face landmarks
predictor =
dlib.shape_predictor('data/data_dlib/shape_predictor_68_face_landmarks.dat')

#  Use Dlib resnet50 model to get 128D face descriptor
face_reco_model =
dlib.face_recognition_model_v1("data/data_dlib/dlib_face_recognition_resnet_m
odel_v1.dat")


#  Return 128D features for single image

def return_128d_features(path_img):
    img_rd = cv2.imread(path_img)
    faces = detector(img_rd, 1)

    logging.info("%-40s %-20s", " Image with faces detected:", path_img)

    # For photos of faces saved, we need to make sure that we can detect faces from
the cropped images
    if len(faces) != 0:
        shape = predictor(img_rd, faces[0])
        face_descriptor = face_reco_model.compute_face_descriptor(img_rd, shape)
    else:
        face_descriptor = 0
        logging.warning("no face")
    return face_descriptor


#   Return the mean value of 128D face descriptor for person X
```

```
def return_features_mean_personX(path_face_personX):
    features_list_personX = []
    photos_list = os.listdir(path_face_personX)
    if photos_list:
        for i in range(len(photos_list)):
            #  return_128d_features()  128D  / Get 128D features for single image of
personX
            logging.info("%-40s %-20s", " / Reading image:", path_face_personX +
"/" + photos_list[i])
            features_128d = return_128d_features(path_face_personX + "/" +
photos_list[i])
            #  Jump if no face detected from image
            if features_128d == 0:
                i += 1
            else:
                features_list_personX.append(features_128d)
    else:
        logging.warning(" Warning: No images in%s/", path_face_personX)


    if features_list_personX:
        features_mean_personX = np.array(features_list_personX,
dtype=object).mean(axis=0)
    else:
        features_mean_personX = np.zeros(128, dtype=object, order='C')
    return features_mean_personX


def main():
    logging.basicConfig(level=logging.INFO)
    #  Get the order of latest person
    person_list = os.listdir("data/data_faces_from_camera/")
    person_list.sort()

    with open("data/features_all.csv", "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        for person in person_list:
            # Get the mean/average features of face/personX, it will be a list with a
length of 128D
            logging.info("%sperson_%s", path_images_from_camera, person)
            features_mean_personX =
return_features_mean_personX(path_images_from_camera + person)

            if len(person.split('_', 2)) == 2:
                # "person_x"
                person_name = person
```

```
            else:
                # "person_x_tom"
                person_name = person.split('_', 2)[-1]
            features_mean_personX = np.insert(features_mean_personX, 0,
person_name, axis=0)
            # features_mean_personX will be 129D, person name + 128 features
            writer.writerow(features_mean_personX)
            logging.info('\n')
        logging.info("Save all the features of faces registered into:
data/features_all.csv")


if __name__ == '__main__':
    main()
import dlib
import numpy as np
import cv2
import os
import shutil
import time
import logging
import tkinter as tk
from tkinter import font as tkFont
from PIL import Image, ImageTk

# Use frontal face detector of Dlib
detector = dlib.get_frontal_face_detector()

class Face_Register:
    def __init__(self):

        self.current_frame_faces_cnt = 0  #  cnt for counting faces in current frame
        self.existing_faces_cnt = 0  # cnt for counting saved faces
        self.ss_cnt = 0  #  cnt for screen shots

        # Tkinter GUI
        self.win = tk.Tk()
        self.win.title("Face Register")

        # PLease modify window size here if needed
        self.win.geometry("1000x500")

        # GUI left part
        self.frame_left_camera = tk.Frame(self.win)
        self.label = tk.Label(self.win)
        self.label.pack(side=tk.LEFT)
        self.frame_left_camera.pack()
```

```
        # GUI right part
        self.frame_right_info = tk.Frame(self.win)
        self.label_cnt_face_in_database = tk.Label(self.frame_right_info,
text=str(self.existing_faces_cnt))
        self.label_fps_info = tk.Label(self.frame_right_info, text="")
        self.input_name = tk.Entry(self.frame_right_info)
        self.input_name_char = ""
        self.label_warning = tk.Label(self.frame_right_info)
        self.label_face_cnt = tk.Label(self.frame_right_info, text="Faces in current
frame: ")
        self.log_all = tk.Label(self.frame_right_info)

        self.font_title = tkFont.Font(family='Helvetica', size=20, weight='bold')
        self.font_step_title = tkFont.Font(family='Helvetica', size=15, weight='bold')
        self.font_warning = tkFont.Font(family='Helvetica', size=15, weight='bold')

        self.path_photos_from_camera = "data/data_faces_from_camera/"
        self.current_face_dir = ""
        self.font = cv2.FONT_ITALIC

        # Current frame and face ROI position
        self.current_frame = np.ndarray
        self.face_ROI_image = np.ndarray
        self.face_ROI_width_start = 0
        self.face_ROI_height_start = 0
        self.face_ROI_width = 0
        self.face_ROI_height = 0
        self.ww = 0
        self.hh = 0

        self.out_of_range_flag = False
        self.face_folder_created_flag = False

        # FPS
        self.frame_time = 0
        self.frame_start_time = 0
        self.fps = 0
        self.fps_show = 0
        self.start_time = time.time()

        self.cap = cv2.VideoCapture(0)  # Get video stream from camera

        # self.cap = cv2.VideoCapture("test.mp4")   # Input local video

    #  Delete old face folders
    def GUI_clear_data(self):
```

```
        #  "/data_faces_from_camera/person_x/"...
        folders_rd = os.listdir(self.path_photos_from_camera)
        for i in range(len(folders_rd)):
            shutil.rmtree(self.path_photos_from_camera + folders_rd[i])
        if os.path.isfile("data/features_all.csv"):
            os.remove("data/features_all.csv")
        self.label_cnt_face_in_database['text'] = "0"
        self.existing_faces_cnt = 0
        self.log_all["text"] = "Face images and `features_all.csv` removed!"

    def GUI_get_input_name(self):
        self.input_name_char = self.input_name.get()
        self.create_face_folder()
        self.label_cnt_face_in_database['text'] = str(self.existing_faces_cnt)

    def GUI_info(self):
        tk.Label(self.frame_right_info,
                 text="Face register",
                 font=self.font_title).grid(row=0, column=0, columnspan=3,
sticky=tk.W, padx=2, pady=20)

        tk.Label(self.frame_right_info, text="FPS: ").grid(row=1, column=0,
sticky=tk.W, padx=5, pady=2)
        self.label_fps_info.grid(row=1, column=1, sticky=tk.W, padx=5, pady=2)

        tk.Label(self.frame_right_info, text="Faces in database: ").grid(row=2,
column=0, sticky=tk.W, padx=5, pady=2)
        self.label_cnt_face_in_database.grid(row=2, column=1, sticky=tk.W,
padx=5, pady=2)

        tk.Label(self.frame_right_info,
                 text="Faces in current frame: ").grid(row=3, column=0, columnspan=2,
sticky=tk.W, padx=5, pady=2)
        self.label_face_cnt.grid(row=3, column=2, columnspan=3, sticky=tk.W,
padx=5, pady=2)

        self.label_warning.grid(row=4, column=0, columnspan=3, sticky=tk.W,
padx=5, pady=2)

        # Step 1: Clear old data
        tk.Label(self.frame_right_info,
                 font=self.font_step_title,
                 text="Step 1: Clear face photos").grid(row=5, column=0,
columnspan=2, sticky=tk.W, padx=5, pady=20)


        # Step 2: Input name and create folders for face
```

```
        tk.Label(self.frame_right_info,
                 font=self.font_step_title,
                 text="Step 2: Input name").grid(row=7, column=0, columnspan=2,
sticky=tk.W, padx=5, pady=20)

        tk.Label(self.frame_right_info, text="Name: ").grid(row=8, column=0,
sticky=tk.W, padx=5, pady=0)
        self.input_name.grid(row=8, column=1, sticky=tk.W, padx=0, pady=2)

        tk.Button(self.frame_right_info,
                  text='Input',
                  command=self.GUI_get_input_name).grid(row=8, column=2, padx=5)

        # Step 3: Save current face in frame
        tk.Label(self.frame_right_info,
                 font=self.font_step_title,
                 text="Step 3: Save face image").grid(row=9, column=0,
columnspan=2, sticky=tk.W, padx=5, pady=20)

        tk.Button(self.frame_right_info,
                  text='Save current face',
                  command=self.save_current_face).grid(row=10, column=0,
columnspan=3, sticky=tk.W)

        # Show log in GUI
        self.log_all.grid(row=11, column=0, columnspan=20, sticky=tk.W, padx=5,
pady=20)

        self.frame_right_info.pack()

    # Mkdir for saving photos and csv
    def pre_work_mkdir(self):
        # Create folders to save face images and csv
        if os.path.isdir(self.path_photos_from_camera):
            pass
        else:
            os.mkdir(self.path_photos_from_camera)

    # Start from person_x+1
    def check_existing_faces_cnt(self):
        if os.listdir("data/data_faces_from_camera/"):
            # Get the order of latest person
            person_list = os.listdir("data/data_faces_from_camera/")
            person_num_list = []
            for person in person_list:
                person_order = person.split('_')[1].split('_')[0]
                person_num_list.append(int(person_order))
```

```
            self.existing_faces_cnt = max(person_num_list)

        # Start from person_1
        else:
            self.existing_faces_cnt = 0

    # Update FPS of Video stream
    def update_fps(self):
        now = time.time()
        #  Refresh fps per second
        if str(self.start_time).split(".")[0] != str(now).split(".")[0]:
            self.fps_show = self.fps
        self.start_time = now
        self.frame_time = now - self.frame_start_time
        self.fps = 1.0 / self.frame_time
        self.frame_start_time = now

        self.label_fps_info["text"] = str(self.fps.__round__(2))

    def create_face_folder(self):
        #  Create the folders for saving faces
        self.existing_faces_cnt += 1
        if self.input_name_char:
            self.current_face_dir = self.path_photos_from_camera + \
                                    "person_" + str(self.existing_faces_cnt) + "_" + \
                                    self.input_name_char
        else:
            self.current_face_dir = self.path_photos_from_camera + \
                                    "person_" + str(self.existing_faces_cnt)
        os.makedirs(self.current_face_dir)
        self.log_all["text"] = "\"" + self.current_face_dir + "/\" created!"
        logging.info("\n%-40s %s", "Create folders:", self.current_face_dir)

        self.ss_cnt = 0  #  Clear the cnt of screen shots
        self.face_folder_created_flag = True  # Face folder already created

    def save_current_face(self):
        if self.face_folder_created_flag:
            if self.current_frame_faces_cnt == 1:
                if not self.out_of_range_flag:
                    self.ss_cnt += 1
                    #  Create blank image according to the size of face detected
                    self.face_ROI_image = np.zeros((int(self.face_ROI_height * 2),
self.face_ROI_width * 2, 3),
                                                   np.uint8)
                    for ii in range(self.face_ROI_height * 2):
                        for jj in range(self.face_ROI_width * 2):
```

```
                    self.face_ROI_image[ii][jj] =
self.current_frame[self.face_ROI_height_start - self.hh + ii][
                        self.face_ROI_width_start - self.ww + jj]
                self.log_all["text"] = "\"" + self.current_face_dir + "/img_face_" +
str(
                    self.ss_cnt) + ".jpg\"" + " saved!"
                self.face_ROI_image = cv2.cvtColor(self.face_ROI_image,
cv2.COLOR_BGR2RGB)

                cv2.imwrite(self.current_face_dir + "/img_face_" + str(self.ss_cnt) +
".jpg", self.face_ROI_image)
                logging.info("%-40s %s/img_face_%s.jpg", "Save into：",
                             str(self.current_face_dir), str(self.ss_cnt) + ".jpg")
            else:
                self.log_all["text"] = "Please do not out of range!"
        else:
            self.log_all["text"] = "No face in current frame!"
    else:
        self.log_all["text"] = "Please run step 2!"

def get_frame(self):
    try:
        if self.cap.isOpened():
            ret, frame = self.cap.read()
            frame = cv2.resize(frame, (640,480))
            return ret, cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    except:
        print("Error: No video input!!!")

#  Main process of face detection and saving
def process(self):
    ret, self.current_frame = self.get_frame()
    faces = detector(self.current_frame, 0)
    # Get frame
    if ret:
        self.update_fps()
        self.label_face_cnt["text"] = str(len(faces))
        #  Face detected
        if len(faces) != 0:
            #   Show the ROI of faces
            for k, d in enumerate(faces):
                self.face_ROI_width_start = d.left()
                self.face_ROI_height_start = d.top()
                #  Compute the size of rectangle box
                self.face_ROI_height = (d.bottom() - d.top())
                self.face_ROI_width = (d.right() - d.left())
                self.hh = int(self.face_ROI_height / 2)
```

```
                self.ww = int(self.face_ROI_width / 2)

                # If the size of ROI > 480x640
                if (d.right() + self.ww) > 640 or (d.bottom() + self.hh > 480) or
(d.left() - self.ww < 0) or (
                        d.top() - self.hh < 0):
                    self.label_warning["text"] = "OUT OF RANGE"
                    self.label_warning['fg'] = 'red'
                    self.out_of_range_flag = True
                    color_rectangle = (255, 0, 0)
                else:
                    self.out_of_range_flag = False
                    self.label_warning["text"] = ""
                    color_rectangle = (255, 255, 255)
                self.current_frame = cv2.rectangle(self.current_frame,
                                                   tuple([d.left() - self.ww, d.top() - self.hh]),
                                                   tuple([d.right() + self.ww, d.bottom() +
self.hh]),
                                                   color_rectangle, 2)
            self.current_frame_faces_cnt = len(faces)

            # Convert PIL.Image.Image to PIL.Image.PhotoImage
            img_Image = Image.fromarray(self.current_frame)
            img_PhotoImage = ImageTk.PhotoImage(image=img_Image)
            self.label.img_tk = img_PhotoImage
            self.label.configure(image=img_PhotoImage)

        # Refresh frame
        self.win.after(20, self.process)

    def run(self):
        self.pre_work_mkdir()
        self.check_existing_faces_cnt()
        self.GUI_info()
        self.process()
        self.win.mainloop()

def main():
    logging.basicConfig(level=logging.INFO)
    Face_Register_con = Face_Register()
    Face_Register_con.run()

if __name__ == '__main__':
    main()
```

FACE DETECTION CODE

```
import dlib
import numpy as np
import cv2
import os
import pandas as pd
import time
import logging
import sqlite3
import datetime
# Dlib  / Use frontal face detector of Dlib
detector = dlib.get_frontal_face_detector()

# Dlib landmark / Get face landmarks
predictor =
dlib.shape_predictor('data/data_dlib/shape_predictor_68_face_landmarks.dat')

# Dlib Resnet Use Dlib resnet50 model to get 128D face descriptor
face_reco_model =
dlib.face_recognition_model_v1("data/data_dlib/dlib_face_recognition_resnet_m
odel_v1.dat")

# Create a connection to the database
conn = sqlite3.connect("attendance.db")
cursor = conn.cursor()

# Create a table for the current date
current_date = datetime.datetime.now().strftime("%Y_%m_%d")  # Replace
hyphens with underscores
table_name = "attendance"
create_table_sql = f"CREATE TABLE IF NOT EXISTS {table_name} (name
TEXT, time TEXT, date DATE, UNIQUE(name, date))"
cursor.execute(create_table_sql)

# Commit changes and close the connection
conn.commit()
conn.close()

class Face_Recognizer:
    def __init__(self):
        self.font = cv2.FONT_ITALIC

        # FPS
        self.frame_time = 0
        self.frame_start_time = 0
        self.fps = 0
```

```
        self.fps_show = 0
        self.start_time = time.time()

        # cnt for frame
        self.frame_cnt = 0

        #  Save the features of faces in the database
        self.face_features_known_list = []
        # / Save the name of faces in the database
        self.face_name_known_list = []

        #  List to save centroid positions of ROI in frame N-1 and N
        self.last_frame_face_centroid_list = []
        self.current_frame_face_centroid_list = []

        # List to save names of objects in frame N-1 and N
        self.last_frame_face_name_list = []
        self.current_frame_face_name_list = []

        #  cnt for faces in frame N-1 and N
        self.last_frame_face_cnt = 0
        self.current_frame_face_cnt = 0

        # Save the e-distance for faceX when recognizing
        self.current_frame_face_X_e_distance_list = []

        # Save the positions and names of current faces captured
        self.current_frame_face_position_list = []
        #  Save the features of people in current frame
        self.current_frame_face_feature_list = []

        # e distance between centroid of ROI in last and current frame
        self.last_current_frame_centroid_e_distance = 0

        #  Reclassify after 'reclassify_interval' frames
        self.reclassify_interval_cnt = 0
        self.reclassify_interval = 10

    #  "features_all.csv"  / Get known faces from "features_all.csv"
    def get_face_database(self):
        if os.path.exists("data/features_all.csv"):
            path_features_known_csv = "data/features_all.csv"
            csv_rd = pd.read_csv(path_features_known_csv, header=None)
            for i in range(csv_rd.shape[0]):
                features_someone_arr = []
                self.face_name_known_list.append(csv_rd.iloc[i][0])
                for j in range(1, 129):
```

```
                if csv_rd.iloc[i][j] == '':
                    features_someone_arr.append('0')
                else:
                    features_someone_arr.append(csv_rd.iloc[i][j])
            self.face_features_known_list.append(features_someone_arr)
        logging.info("Faces in Database：  %d",
len(self.face_features_known_list))
        return 1
    else:
        logging.warning("'features_all.csv' not found!")
        logging.warning("Please run 'get_faces_from_camera.py' "
                        "and 'features_extraction_to_csv.py' before
'face_reco_from_camera.py'")
        return 0

  def update_fps(self):
    now = time.time()
    # Refresh fps per second
    if str(self.start_time).split(".")[0] != str(now).split(".")[0]:
        self.fps_show = self.fps
    self.start_time = now
    self.frame_time = now - self.frame_start_time
    self.fps = 1.0 / self.frame_time
    self.frame_start_time = now

  @staticmethod
  # / Compute the e-distance between two 128D features
  def return_euclidean_distance(feature_1, feature_2):
    feature_1 = np.array(feature_1)
    feature_2 = np.array(feature_2)
    dist = np.sqrt(np.sum(np.square(feature_1 - feature_2)))
    return dist

  # / Use centroid tracker to link face_x in current frame with person_x in last
frame
  def centroid_tracker(self):
    for i in range(len(self.current_frame_face_centroid_list)):
        e_distance_current_frame_person_x_list = []
        #  For object 1 in current_frame, compute e-distance with object 1/2/3/4/...
in last frame
        for j in range(len(self.last_frame_face_centroid_list)):
            self.last_current_frame_centroid_e_distance =
self.return_euclidean_distance(
                self.current_frame_face_centroid_list[i],
self.last_frame_face_centroid_list[j])

            e_distance_current_frame_person_x_list.append(
```

```
                self.last_current_frame_centroid_e_distance)

            last_frame_num = e_distance_current_frame_person_x_list.index(
                min(e_distance_current_frame_person_x_list))
            self.current_frame_face_name_list[i] =
self.last_frame_face_name_list[last_frame_num]

    #  cv2 window / putText on cv2 window
    def draw_note(self, img_rd):
        #  / Add some info on windows
        cv2.putText(img_rd, "Face Recognizer with Deep Learning", (20, 40),
self.font, 1, (255, 255, 255), 1, cv2.LINE_AA)
        cv2.putText(img_rd, "Frame:  " + str(self.frame_cnt), (20, 100), self.font,
0.8, (0, 255, 0), 1,
                    cv2.LINE_AA)
        cv2.putText(img_rd, "FPS:    " + str(self.fps.__round__(2)), (20, 130),
self.font, 0.8, (0, 255, 0), 1,
                    cv2.LINE_AA)
        cv2.putText(img_rd, "Faces:  " + str(self.current_frame_face_cnt), (20, 160),
self.font, 0.8, (0, 255, 0), 1,
                    cv2.LINE_AA)
        cv2.putText(img_rd, "Q: Quit", (20, 450), self.font, 0.8, (255, 255, 255), 1,
cv2.LINE_AA)

        for i in range(len(self.current_frame_face_name_list)):
            img_rd = cv2.putText(img_rd, "Face_" + str(i + 1), tuple(
                [int(self.current_frame_face_centroid_list[i][0]),
int(self.current_frame_face_centroid_list[i][1])]),
                            self.font,
                            0.8, (255, 190, 0),
                            1,
                            cv2.LINE_AA)
    # insert data in database

    def attendance(self, name):
        current_date = datetime.datetime.now().strftime('%Y-%m-%d')
        conn = sqlite3.connect("attendance.db")
        cursor = conn.cursor()
        # Check if the name already has an entry for the current date
        cursor.execute("SELECT * FROM attendance WHERE name = ? AND date
= ?", (name, current_date))
        existing_entry = cursor.fetchone()

        if existing_entry:
            print(f"{name} is already marked as present for {current_date}")
        else:
            current_time = datetime.datetime.now().strftime('%H:%M:%S')
```

```
            cursor.execute("INSERT INTO attendance (name, time, date) VALUES
(?, ?, ?)", (name, current_time, current_date))
            conn.commit()
            print(f"{name} marked as present for {current_date} at {current_time}")

        conn.close()

    #  Face detection and recognition wit OT from input video stream
    def process(self, stream):
        # 1.  Get faces known from "features.all.csv"
        if self.get_face_database():
            while stream.isOpened():
                self.frame_cnt += 1
                logging.debug("Frame " + str(self.frame_cnt) + " starts")
                flag, img_rd = stream.read()
                kk = cv2.waitKey(1)

                # 2.  Detect faces for frame X
                faces = detector(img_rd, 0)

                # 3.  Update cnt for faces in frames
                self.last_frame_face_cnt = self.current_frame_face_cnt
                self.current_frame_face_cnt = len(faces)

                # 4.  Update the face name list in last frame
                self.last_frame_face_name_list = self.current_frame_face_name_list[:]

                # 5.  update frame centroid list
                self.last_frame_face_centroid_list =
self.current_frame_face_centroid_list
                self.current_frame_face_centroid_list = []

                # 6.1  if cnt not changes
                if (self.current_frame_face_cnt == self.last_frame_face_cnt) and (
                        self.reclassify_interval_cnt != self.reclassify_interval):
                    logging.debug("scene 1:   No face cnt changes in this frame!!!")

                    self.current_frame_face_position_list = []

                    if "unknown" in self.current_frame_face_name_list:
                        self.reclassify_interval_cnt += 1

                    if self.current_frame_face_cnt != 0:
                        for k, d in enumerate(faces):
                            self.current_frame_face_position_list.append(tuple(
                                [faces[k].left(), int(faces[k].bottom() + (faces[k].bottom() -
faces[k].top()) / 4)]))
```

```
                    self.current_frame_face_centroid_list.append(
                        [int(faces[k].left() + faces[k].right()) / 2,
                        int(faces[k].top() + faces[k].bottom()) / 2])

                    img_rd = cv2.rectangle(img_rd,
                                           tuple([d.left(), d.top()]),
                                           tuple([d.right(), d.bottom()]),
                                           (255, 255, 255), 2)

                #  Multi-faces in current frame, use centroid-tracker to track
                if self.current_frame_face_cnt != 1:
                    self.centroid_tracker()

                for i in range(self.current_frame_face_cnt):
                    # 6.2 Write names under ROI
                    img_rd = cv2.putText(img_rd,
self.current_frame_face_name_list[i],
                                         self.current_frame_face_position_list[i], self.font, 0.8,
(0, 255, 255), 1,
                                         cv2.LINE_AA)
                self.draw_note(img_rd)

            # 6.2  If cnt of faces changes, 0->1 or 1->0 or ...
            else:
                logging.debug("scene 2: / Faces cnt changes in this frame")
                self.current_frame_face_position_list = []
                self.current_frame_face_X_e_distance_list = []
                self.current_frame_face_feature_list = []
                self.reclassify_interval_cnt = 0

                # 6.2.1  Face cnt decreases: 1->0, 2->1, ...
                if self.current_frame_face_cnt == 0:
                    logging.debug("  / No faces in this frame!!!")
                    # clear list of names and features
                    self.current_frame_face_name_list = []
                # 6.2.2 / Face cnt increase: 0->1, 0->2, ..., 1->2, ...
                else:
                    logging.debug("  scene 2.2  Get faces in this frame and do face
recognition")
                    self.current_frame_face_name_list = []
                    for i in range(len(faces)):
                        shape = predictor(img_rd, faces[i])
                        self.current_frame_face_feature_list.append(
                            face_reco_model.compute_face_descriptor(img_rd, shape))
                        self.current_frame_face_name_list.append("unknown")

                    # 6.2.2.1 Traversal all the faces in the database
```

```
                for k in range(len(faces)):
                    logging.debug("  For face %d in current frame:", k + 1)
                    self.current_frame_face_centroid_list.append(
                        [int(faces[k].left() + faces[k].right()) / 2,
                        int(faces[k].top() + faces[k].bottom()) / 2])

                    self.current_frame_face_X_e_distance_list = []

                    # 6.2.2.2  Positions of faces captured
                    self.current_frame_face_position_list.append(tuple(
                        [faces[k].left(), int(faces[k].bottom() + (faces[k].bottom() -
faces[k].top()) / 4)]))

                    # 6.2.2.3
                    # For every faces detected, compare the faces in the database
                    for i in range(len(self.face_features_known_list)):
                        #
                        if str(self.face_features_known_list[i][0]) != '0.0':
                            e_distance_tmp = self.return_euclidean_distance(
                                self.current_frame_face_feature_list[k],
                                self.face_features_known_list[i])
                            logging.debug("      with person %d, the e-distance: %f", i +
1, e_distance_tmp)
                            self.current_frame_face_X_e_distance_list.append(e_distan
ce_tmp)
                        else:
                            #  person_X
                            self.current_frame_face_X_e_distance_list.append(999999
999)

                    # 6.2.2.4 / Find the one with minimum e distance
                    similar_person_num =
self.current_frame_face_X_e_distance_list.index(
                        min(self.current_frame_face_X_e_distance_list))

                    if min(self.current_frame_face_X_e_distance_list) < 0.4:
                        self.current_frame_face_name_list[k] =
self.face_name_known_list[similar_person_num]
                        logging.debug("  Face recognition result: %s",
                                  self.face_name_known_list[similar_person_num])

                        # Insert attendance record
                        nam =self.face_name_known_list[similar_person_num]

                        print(type(self.face_name_known_list[similar_person_num]))
                        print(nam)
                        self.attendance(nam)
```

```
                    else:
                        logging.debug("  Face recognition result: Unknown person")

                    # 7.  / Add note on cv2 window
                    self.draw_note(img_rd)

                # 8.  'q'  / Press 'q' to exit
                if kk == ord('q'):
                    break

                self.update_fps()
                cv2.namedWindow("camera", 1)
                cv2.imshow("camera", img_rd)

                logging.debug("Frame ends\n\n")


    def run(self):
        # cap = cv2.VideoCapture("video.mp4")  # Get video stream from video file
        cap = cv2.VideoCapture(0)              # Get video stream from camera
        self.process(cap)

        cap.release()
        cv2.destroyAllWindows()


def main():

    logging.basicConfig(level=logging.INFO)
    Face_Recognizer_con = Face_Recognizer()
    Face_Recognizer_con.run()

if __name__ == '__main__':
    main()
```

Flask code

```
from flask import Flask, render_template, request
import sqlite3
from datetime import datetime

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html', selected_date='', no_data=False)

@app.route('/attendance', methods=['POST'])
def attendance():
    selected_date = request.form.get('selected_date')
    selected_date_obj = datetime.strptime(selected_date, '%Y-%m-%d')
    formatted_date = selected_date_obj.strftime('%Y-%m-%d')

    conn = sqlite3.connect('attendance.db')
    cursor = conn.cursor()

    cursor.execute("SELECT name, time FROM attendance WHERE date = ?",
(formatted_date,))
    attendance_data = cursor.fetchall()

    conn.close()

    if not attendance_data:
        return render_template('index.html', selected_date=selected_date,
no_data=True)

    return render_template('index.html', selected_date=selected_date,
attendance_data=attendance_data)


if __name__ == '__main__':
    app.run(debug=True)
from flask import Flask, render_template, request
import sqlite3
from datetime import datetime

app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html', selected_date='', no_data=False)
```

```
@app.route('/attendance', methods=['POST'])
def attendance():
    selected_date = request.form.get('selected_date')
    selected_date_obj = datetime.strptime(selected_date, '%Y-%m-%d')
    formatted_date = selected_date_obj.strftime('%Y-%m-%d')

    conn = sqlite3.connect('attendance.db')
    cursor = conn.cursor()

    cursor.execute("SELECT name, time FROM attendance WHERE date = ?",
(formatted_date,))
    attendance_data = cursor.fetchall()

    conn.close()

    if not attendance_data:
        return render_template('index.html', selected_date=selected_date,
no_data=True)

    return render_template('index.html', selected_date=selected_date,
attendance_data=attendance_data)


if __name__ == '__main__':
    app.run(debug=True)
```

5.1 INTERFACE:

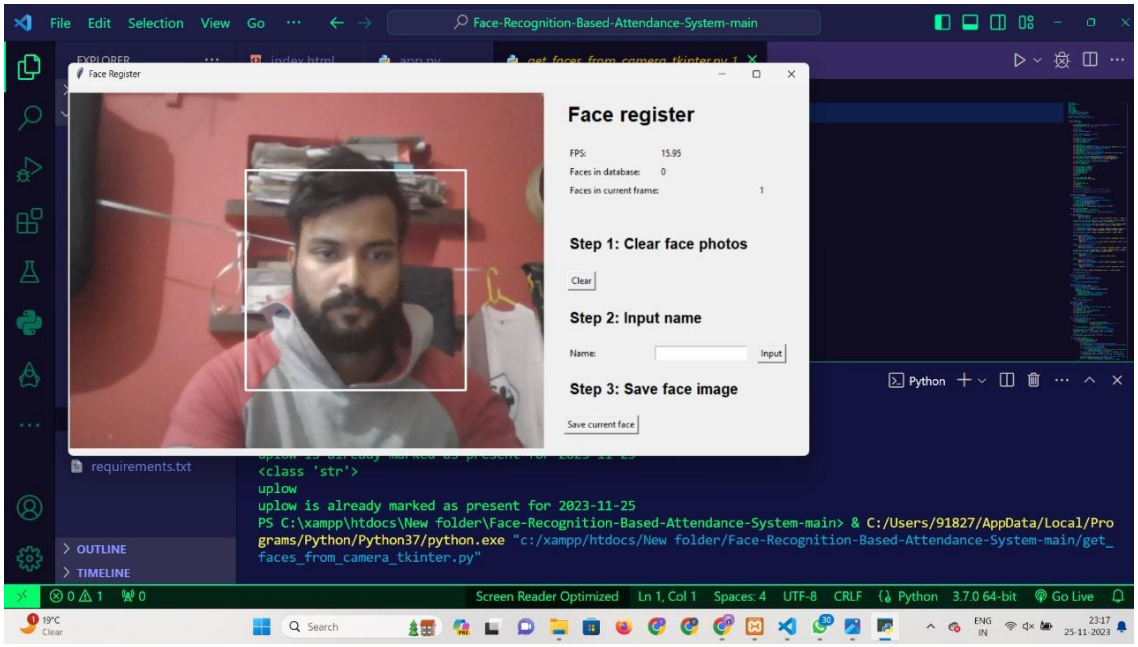


Fig (5.1.1)- face recognition window

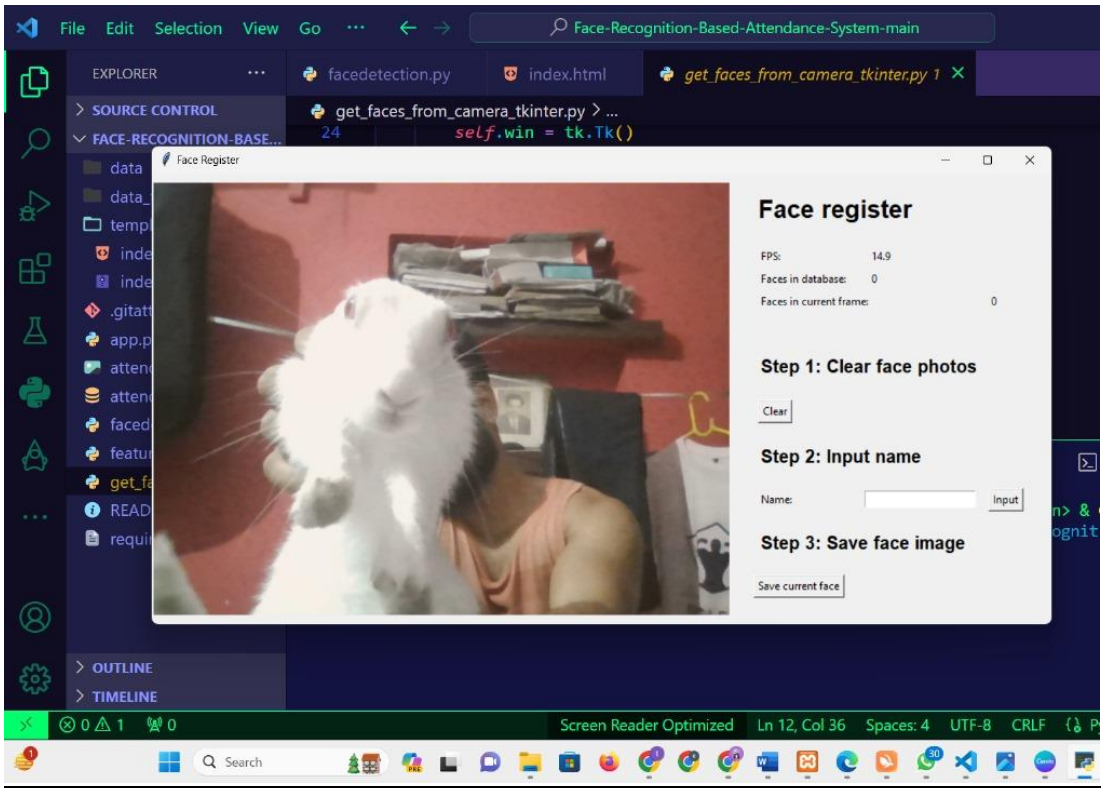


Fig (5.1.2)- The model does not recognize any other objects (animals, things, etc) except for the human face.

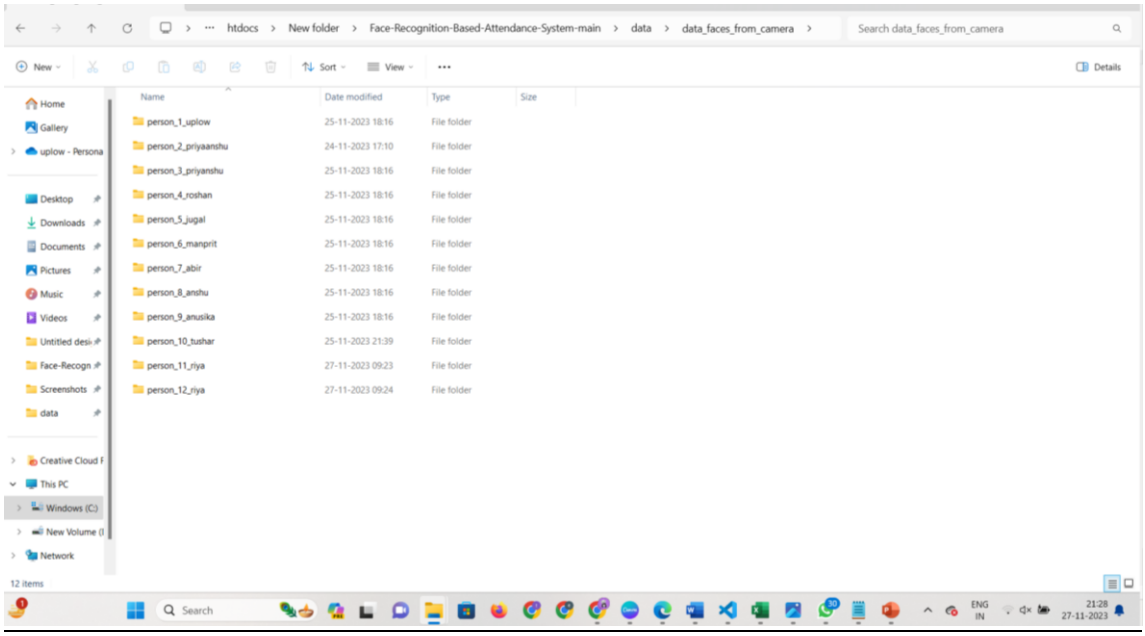


Fig (5.1.3)- files getting saved with the respective students' names as entered in the registration window.

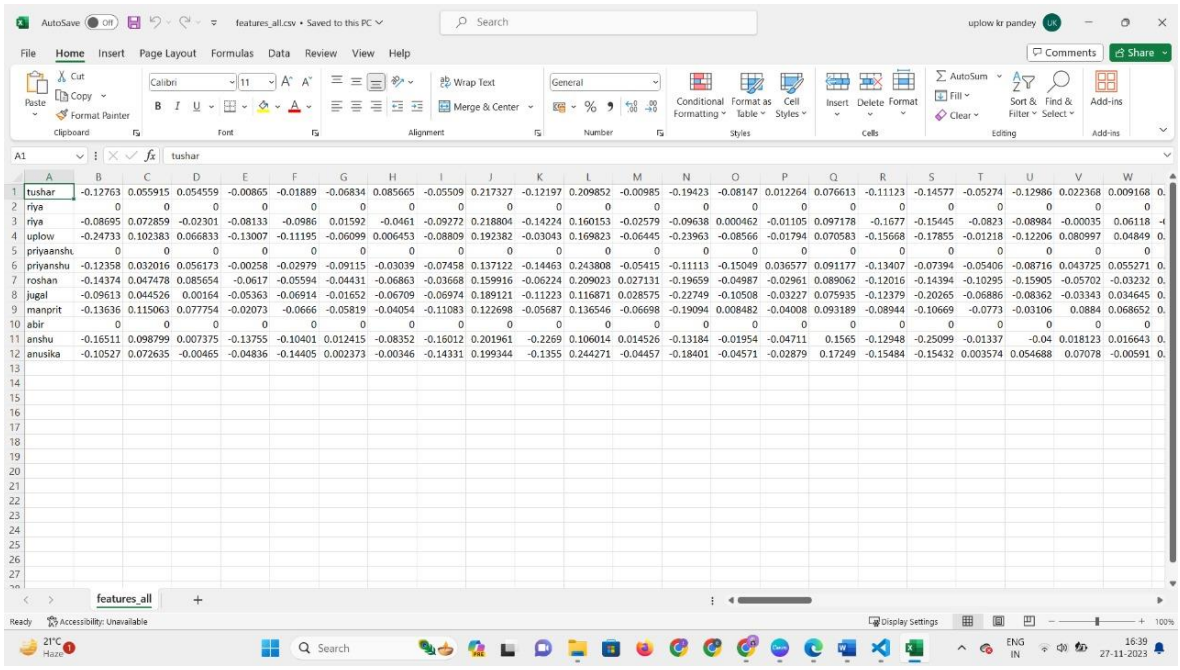
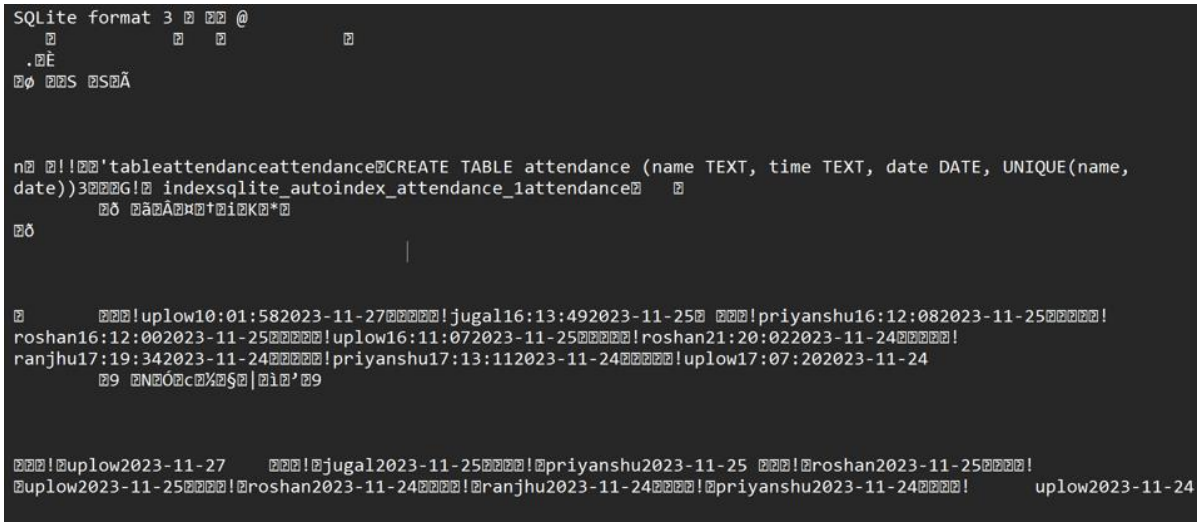


Fig (5.1.4)- feature extraction data of the registered image on training the model.



Fig(5.1.5)- SQLite code for setting up data in the database.

5.2 OUTPUT:

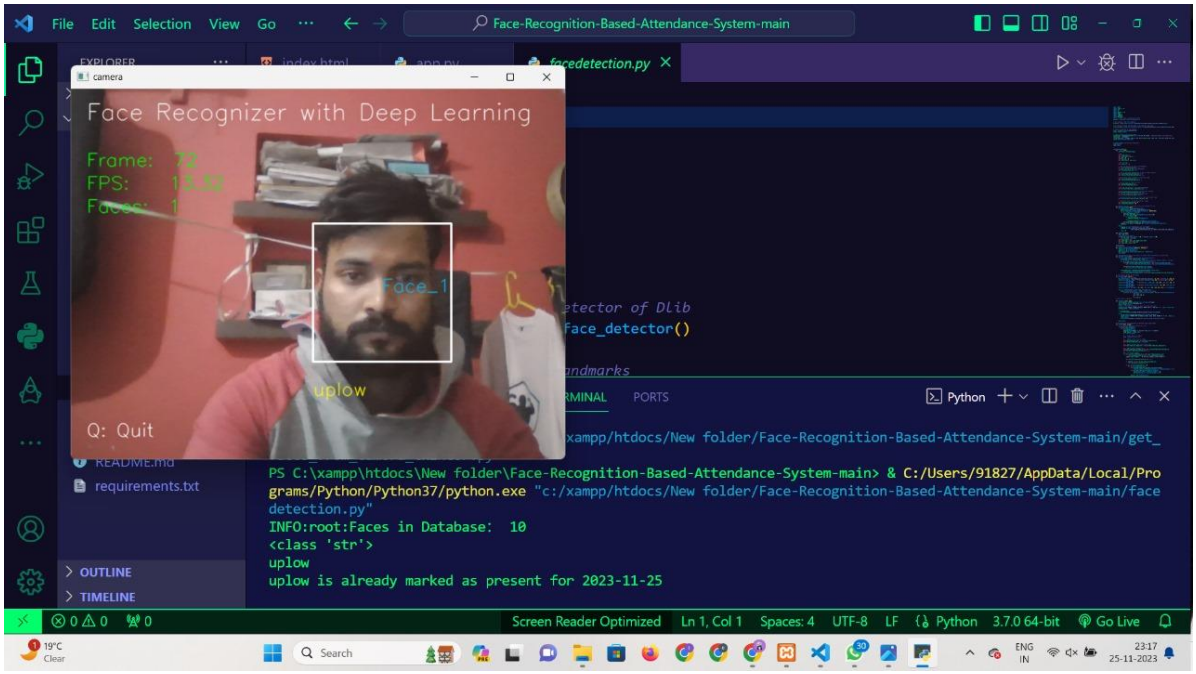


Fig (5.2.1)- system recognizing and capturing the names of students who are already registered.

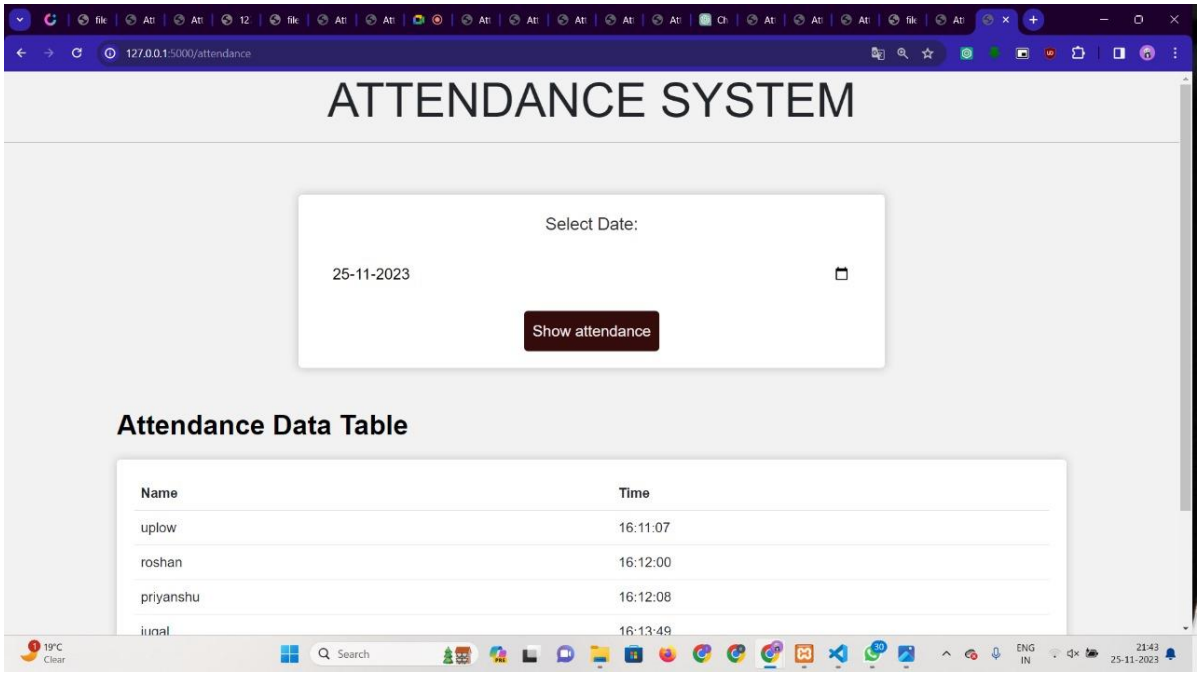


Fig (5.2.2)- web page that displays list of students attendance date wise.

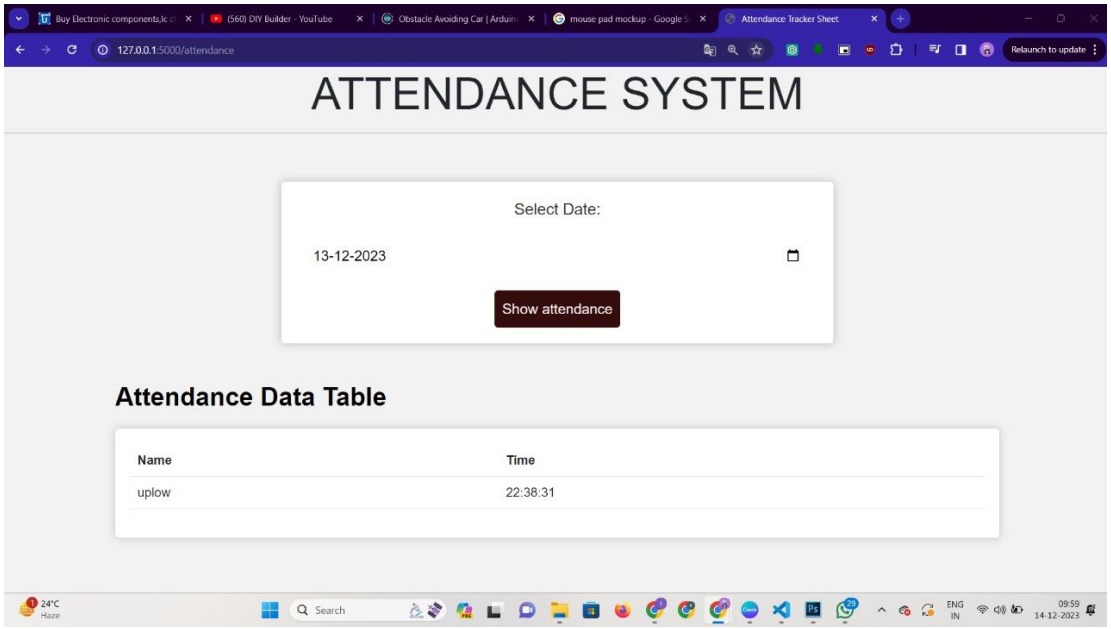


Fig (5.2.3)- web page that displays list of students attendance date wise.

6. CONCLUSION

Using the face characteristics as biometric, the face recognition system can be implemented as a touchless technology.

The training database is created by training the system with the faces of the authorized students. The cropped images are then stored as a database with respective labels.

The small institutions as well as organizations will be able to use this system as we have reduced costs as to other software comparatively.

With this automation, the time initially used for the attendance will instead be used on work, increasing efficiency.

If the work becomes efficient then it will bring more of an economic and monetary value to the organizations.

For future evaluations, we will be able to recognize the attendance pattern and implement a better working environment both for the clients and consumers.

Getting an updated list of attendees, and managing notifications may also become easier, as that of a manual process of maintaining contact lists.

Furthermore, by increasing system specification, features, and training data, our model does have an influential future scope, as our motive is to reduce the software development cost for future accessibility.

7. REFERENCES

https://www.softwaresuggest.com/face-recognition-attendance-system

https://truein.com/face-recognition-attendance-system/

https://www.analyticsvidhya.com/blog/2021/11/build-face-recognition-attendance-system-using-python/

https://chat.openai.com/

https://www.youtube.com/