Python Project :

Face Recognition Based Attendance System

TEAM

Arnab Bera
21051979

Roshan Aryan
2105570

Sagnik Ghosh
21052699

Tushar Goyal
21052462

CONTENTS

01 Problem Statement

02 Introduction

03 Objectives

04 Library Requirements

05 Working

06 Step 1

07 Step 2

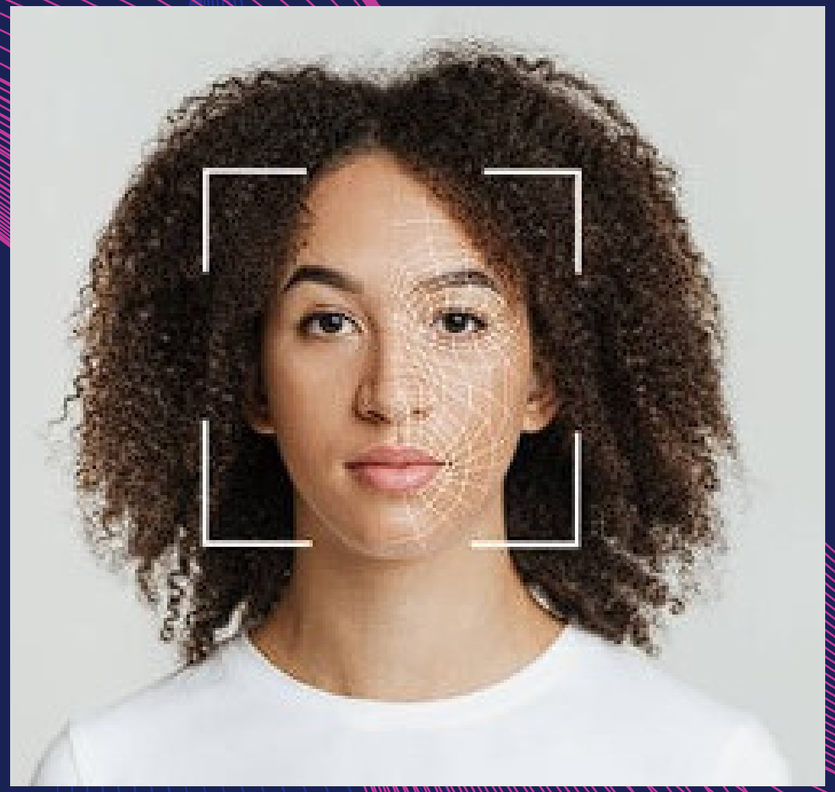
08 Step 3

09 Step 4

10

PROBLEM STATEMENT

In traditional attendance tracking systems, manual methods like paper-based registers or card swiping systems are prone to errors, time-consuming, and lack efficiency. Moreover, these methods can be easily manipulated leading to inaccurate attendance records. With the advancement of technology, there is a need for a more reliable and automated solution for attendance tracking.



INTRODUCTION

This project is a face recognition-based attendance system that uses OpenCV and Python. The system uses a camera to capture images of individuals and then compares them with the images in the database to mark attendance.

Objectives:

Accurate Attendance Tracking:
Implement a robust face recognition algorithm to accurately identify individuals and record their attendance.

Real-time Monitoring:
Provide administrators with real-time attendance data, enabling them to monitor attendance patterns and address issues promptly.

Security Enhancement:
Improve security measures by utilizing biometric data (facial features) for authentication, reducing the risk of proxy attendance or unauthorized access.

Efficiency Improvement:
Automate the attendance tracking process to save time and resources, enhancing overall operational efficiency.

Scalability:
Design the system to be scalable, capable of handling a large number of individuals without compromising performance.

Libraries :

Requirements.txt

dlib

Version : 19.17.0

Dlib is a popular toolkit for machine learning that is used primarily for computer vision and image processing tasks, such as face recognition, facial landmark detection, object detection, and more. It is written in C++ but has Python bindings, making it easily accessible from Python code.

numpy

Version : 1.22.0

NumPy is a Python library used for working with arrays. It also has functions for working in domain of linear algebra, fourier transform, and matrices.

scikit-image

Version : 0.18.3

scikit-image is a Python package dedicated to image processing, and using natively NumPy arrays as image objects.

pandas

Version : 1.3.4

Pandas is a Python library used for working with data sets. It has functions for analyzing, cleaning, exploring, and manipulating data. .

opencv-python

Version : 4.5.4.58

OpenCV is a tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more.

flask

Flask is a small and lightweight Python web framework that provides useful tools and features that make creating web applications in Python easier.

WORKING

Step 1:

Collect the Faces Dataset by running python get_faces_from_camera_tkinter.py

Step 2:

Convert the dataset into python features_extraction_to_csv.py.

Step 3:

To take the attendance run python attendance_taker.py .

Step 4:

Check the Database by python app.py.

Step 1: get_faces_from_camera_tkinter.py

Initialization:

Imports libraries like OpenCV and dlib for computer vision tasks.
Creates a class Face_Register to handle the registration process.
Initializes variables to store frame counts, folder paths, and GUI elements.
Creates a window using Tkinter for displaying the camera feed and controls.

Pre-processing:

Clears any existing folders storing previously registered faces.
Defines functions to handle GUI interactions like getting user input for names.
Sets up logging to record important events.
Creates folders for saving captured faces if they don't exist.
Determines the number of existing registered faces based on folder structure.

Main loop:

Continuously captures frames from the camera.
Uses dlib's face detector to identify faces in the current frame.
Updates the FPS (Frames Per Second) to show how smoothly the video is processing.
Displays the number of detected faces on the GUI.

Face detection and saving:

If a face is detected:
Calculates the bounding box coordinates around the face.
Checks if the face is within the frame boundaries to avoid partial captures.
If within boundaries, creates a new folder for the registered person (if not done already).
Captures a screenshot of the detected face and saves it within the person's folder.
Updates the GUI with logs and warnings.

GUI update and loop continuation:

Converts the captured frame (OpenCV format) to a format suitable for Tkinter display.
Updates the image on the GUI window to show the latest camera feed with detected faces.
Schedules the process function to be called again after a short delay, creating a continuous loop for capturing and processing frames.

Main function:

Sets up logging configuration.
Creates an instance of Face_Register.

-

Step 2: features_extraction_to_csv.py.

Initialization:
Imports libraries for computer vision, file operations, and logging.
Defines paths to folders containing cropped face images and pre-trained face recognition models.
Creates functions for:
Extracting 128-dimensional facial features from a single image.
Calculating the average facial features (mean) for a person based on their registered images.

Main loop:
Sets up logging configuration.
Gets a list of folders representing registered people based on the folder structure ("person_X").

Processing each person's folder:
Opens a CSV file for writing ("data/features_all.csv").
Iterates through each person's folder ("person_X").
Calculates the average facial features (mean) for the current person using the return_features_mean_personX function.

Extracting and storing features:
Extracts the person's name from the folder name format ("person_X" or "person_X_name").
Combines the person's name with their average facial features (128 dimensions) into a single list.
Writes the combined list (name + features) as a row in the CSV file.
Logs informative messages about the process.

Saving results:
Closes the CSV file.

- Logs a message indicating successful storage of all registered faces' features in "data/features_all.csv".

Step 3: attendance_taker.py

Initialization: An empty list (current_frame_face_position_list) is created to store the positions of detected faces in the current frame.

Unknown Face Check: The code checks if there are any faces identified as "unknown" in the current frame (current_frame_face_name_list).

Reclassification Counter: If unknown faces are found, a counter (reclassify_interval_cnt) is incremented, possibly tracking frames since the last attempt to reclassify them.

Processing Faces (if any): If faces are detected (current_frame_face_cnt is not zero), the missing code likely iterates through them:

Extracting their positions (bounding boxes).
Appending positions to current_frame_face_position_list.

Step 4: app.py.

Manages an attendance system.

Main Page (index route):

- Displays attendance information.
- Handles scenarios when data is available or missing.
- Attendance Data Handling (attendance route):
- Processes form submissions (possibly for recording or filtering attendance data).
- Interacts with a database to retrieve attendance information.

THANK YOU