

PROFESSIONAL TRAINING REPORT - II

entitled

NEURO-FACE ATTENDANCE: EFFORTLESS FACIAL RECOGNITION

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering degree in Computer Science and Engineering with
specialization in Artificial Intelligence.

by

Sai Dharmitha (Reg. No.: 41613016)

Chandu Chandana (Reg. No.: 41731025)

Bandari Ruthvik Nath (Reg. No.: 41731018)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

CATEGORY -1 UNIVERSITY BY UGC

**Accredited with Grade "A++" by NAAC | 12B Status by UGC | Approved by AICTE
JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI - 600 119**

May 2024



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

CATEGORY-I UNIVERSITY BY UGC

Accredited "A++" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Professional Training Report is the bonafide work of **Mr./Ms. Sai Dharmitha (Reg. No.: 41613016), Bandari Ruthvik Nath (Reg. No.: 41731018), Chandu Chandana (Reg. No.: 41731025)**, who carried out the project entitled **"NEURO-FACE ATTENDANCE: EFFORTLESS FACIAL RECOGNITION"** under my supervision from January 2024 to May 2024.

Internal Guide

Ms. Lekshmi S Raveendran, M.Tech.,

Head of the Department

Dr. S. VIGNESHWARI, M.E., Ph.D.,

Submitted for Viva voce Examination held on 07/05/24

Internal Examiner

External Examiner

DECLARATION

I, Sai Dharmitha (Reg. No.: 41613016), hereby declare that the Professional Training Report-II entitled **"NEURO-FACE ATTENDANCE: EFFORTLESS FACIAL RECOGNITION"** done by me under the guidance of **Ms. Lekshmi S Raveendran, Mtech.**, is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Computer Science and Engineering with specialization in Artificial Intelligence.

DATE: 07/05/24

PLACE: Chennai

P. Sai Dharmitha
SIGNATURE OF THE CANDIDATE

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T.Sasikala M.E., Ph.D., Dean, School of Computing, Dr. S.Vigneshwari M.E., Ph.D., Head of the Department of Computer Science and Engineering** for providing me necessary support during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Internal Guide **Ms. Lekshmi S Raveendran, M.Tech.**, for her valuable guidance, suggestions and constant encouragement which paved way for the successful completion of my phase-2 Professional Training.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

COURSE CERTIFICATE





GOLD



CERTIFICATE

This is to certify that

DHARMITHA PILLARI

has successfully cleared the assessment on

Digital 101 – 30 Hours



FSP/2024/S/10168084

Aligned to Competency Standards developed by IT-ITeS Sector Skills Council
NASSCOM in collaboration with Industry and approved by Government.

02/05/2024
Date of issue



CEO, IT-ITeS Sector Skills Council
NASSCOM

Gold Category: 70% and above score.
Detailed Scorecard included

ABSTRACT

In today's fast-paced world, manual attendance tracking systems often fall short in accuracy, efficiency, and scalability. As organizations strive for modern solutions to streamline processes, the integration of facial recognition technology emerges as a promising alternative. This project aims to develop an innovative attendance tracking system employing neural network-powered facial recognition algorithms. The proposed system leverages deep learning techniques, specifically convolutional neural networks (CNNs), to extract and analyze facial features for identification and authentication purposes. Through the utilization of labeled datasets, the CNN model undergoes rigorous training to learn and discern unique facial attributes, ensuring robust recognition performance across diverse environments and conditions.

Key components of the system include image preprocessing techniques to enhance facial feature extraction, feature representation methods to encode facial information into numerical vectors, and classification algorithms for accurate identification and verification of individuals. The integration of these components within a cohesive framework enables real-time processing of facial images captured by surveillance cameras or dedicated devices. Moreover, the system incorporates advanced security measures to safeguard sensitive biometric data, adhering to privacy regulations and best practices in data protection. Encryption protocols and secure storage mechanisms are implemented to prevent unauthorized access and ensure confidentiality of stored information.

TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	ABSTRACT	vi
	LIST OF FIGURES	ix
1	INTRODUCTION	10
	1.1 Overview	
2	LITERATURE REVIEW	12
	2.1 survey	
3	REQUIREMENTS ANALYSIS	
	3.1 Objective	14
	3.2.1 Hardware Requirements	
	3.2.2 Software Requirements	16
4	DESIGN DESCRIPTION OF PROPOSED PRODUCT	18
	4.1.1 Ideation Map/Architecture Diagram	20
	4.1.2 Various stages	21
	4.1.3 Internal or Component design structure	22
	4.1.4 working principles	24
	4.2 4.2.1 Novelty of the Project	28
5	IMPLEMENTATION AND RESULTS	30
6	CONCLUSION	33
	REFERENCES	34
	APPENDIX	
	A. Research Paper	35
	B. Source Code	38

LIST OF FIGURES

Figure No.	Figure Name	Page No.
4.1	System Architecture	7
4.2	Method of working	20
4.3	Facial Detection	24
4.4	Attendance Tracking	25
4.5	Attendance Tracker Sheet	26
5.1	Training Data	31

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Neuro Face Attendance: Effortless Facial Recognition" is a cutting-edge project aimed at revolutionizing attendance tracking systems through advanced neural network technology. Leveraging the power of artificial intelligence, this project offers a seamless and efficient solution for monitoring attendance in various settings, from educational institutions to corporate environments.

At its core, the project utilizes sophisticated facial recognition algorithms powered by neural networks. These algorithms are trained on vast datasets of facial images, enabling them to accurately identify individuals with high precision and reliability. By harnessing the capabilities of neural networks, the system continuously learns and improves its recognition accuracy over time, ensuring optimal performance in diverse conditions and environments.

One of the key features of Neuro Face Attendance is its user-friendly interface, which simplifies the attendance tracking process for both administrators and attendees. With just a quick scan of their faces, individuals can effortlessly mark their attendance, eliminating the need for cumbersome manual methods such as paper-based sign-in sheets or barcode scanning. This streamlined approach not only saves time but also minimizes the potential for errors and discrepancies in attendance records.

Furthermore, Neuro-Face Attendance offers robust security measures to safeguard sensitive biometric data and ensure compliance with privacy regulations. Advanced encryption techniques are employed to protect facial data from unauthorized access, while strict access controls and authentication mechanisms prevent misuse of the system.

Central to the success of facial recognition technology is the advent of

Convolutional Neural Networks (CNNs), a class of deep learning models that have revolutionized the field of computer vision. CNNs are uniquely suited to the task of facial recognition due to their ability to automatically learn hierarchical representations of features directly from raw image data. By leveraging convolutional layers, pooling layers, and fully connected layers, CNNs can capture spatial hierarchies in facial images, extracting discriminative features that are invariant to variations in lighting conditions, facial expressions, poses, and occlusions.

The application of CNNs in facial recognition has ushered in a new era of accuracy, efficiency, and scalability. CNN-based facial recognition systems have demonstrated unparalleled performance in various real-world scenarios, ranging from security and surveillance to access control and human-computer interaction. In the context of attendance monitoring systems, CNN-based facial recognition offers a streamlined and automated solution for accurately tracking and identifying individuals. By seamlessly integrating facial recognition technology with attendance monitoring systems, organizations can enhance security, improve efficiency, and streamline administrative processes.

CNN-based facial recognition systems offer a reliable and non-intrusive means of verifying individual identities, eliminating the need for manual record-keeping or physical tokens. Moreover, these systems can provide real-time monitoring and alerts, enabling proactive management of attendance records and security incidents. Facial recognition technology has garnered widespread adoption across various industries and sectors, owing to its versatility and applicability to a myriad of use cases. From law enforcement and border control to retail and healthcare, facial recognition systems have found diverse applications, revolutionizing the way organizations interact with and authenticate individuals.

The ability of facial recognition technology to accurately identify and verify individuals in real-time has not only enhanced security measures but has also streamlined operations and improved user experiences. One of the key drivers behind the widespread adoption of facial recognition technology is its non-

intrusive nature. Unlike other biometric modalities that may require physical contact or specialized equipment, facial recognition can be deployed using existing infrastructure, such as surveillance cameras or smartphones. This inherent convenience and ease of integration have made facial recognition an attractive option for organizations seeking to enhance security without compromising user convenience.

Moreover, facial recognition technology offers a high degree of accuracy, especially when coupled with advanced algorithms and deep learning techniques. The ability of facial recognition systems to analyze and compare facial features with a vast database of reference images enables rapid and precise identification of individuals, even in challenging conditions such as low light or partial occlusion. This level of accuracy is paramount in critical applications such as law enforcement and border security, where the timely and accurate identification of suspects or persons of interest is of utmost importance.

In addition to its primary function of attendance tracking, Neuro Face Attendance can be customized and integrated with existing systems to support various additional features and functionalities. These may include real-time monitoring and reporting, automated notifications for late or absent individuals, and seamless integration with payroll and HR management systems.

Overall, Neuro Face Attendance represents a groundbreaking advancement in attendance management technology, offering unparalleled accuracy, efficiency, and convenience through the application of neural network-based facial recognition. With its intuitive interface, robust security measures, and customizable features, it sets a new standard for attendance tracking systems in the digital age.

CHAPTER 2

LITERATURE REVIEW

2.1 SURVEY

A literature survey for the project "Neuro Face Attendance: Effortless Facial Recognition" would involve reviewing existing research, publications, and relevant works in the fields of facial recognition technology, neural networks, attendance management systems, and related areas. Here's a structured overview of the literature survey:

Facial Recognition Technology:

Review studies that discuss the evolution of facial recognition technology, including its historical development, key milestones, and recent advancements. Explore research papers on different facial recognition algorithms, techniques, and methodologies, such as eigenfaces, deep learning-based approaches, and convolutional neural networks (CNNs).

Neural Networks in Facial Recognition:

Investigate how neural networks, especially deep learning architectures like CNNs, have transformed facial recognition accuracy and performance. Examine research articles that detail the design, training, and optimization of neural networks for facial feature extraction, representation learning, and classification tasks.

Attendance Management Systems:

Survey existing literature on traditional attendance management systems, including manual methods, barcode-based systems, and RFID technology. Explore studies that highlight the limitations and challenges of conventional

attendance tracking methods, such as inaccuracies, inefficiencies, and susceptibility to fraud.

Integration of Facial Recognition in Attendance Systems:

Identify research papers, case studies, and practical implementations that have integrated facial recognition technology into attendance management systems. Analyze the advantages, limitations, and real-world applications of facial recognition-based attendance systems across various industries and settings.

Privacy, Security, and Ethical Considerations:

Review literature discussing the ethical implications, privacy concerns, and security risks associated with facial recognition technology, particularly in the context of attendance tracking.

Explore studies that propose privacy-preserving techniques, encryption methods, and regulatory frameworks to address security and privacy challenges in facial recognition systems.

User Experience and Interface Design:

Investigate research on user experience (UX) design principles and best practices for developing intuitive, user-friendly interfaces in attendance management software.

Examine studies that evaluate user acceptance, satisfaction, and adoption rates of facial recognition-based attendance systems among administrators, employees, and students.

CHAPTER 3

REQUIREMENTS ANALYSIS

3.1 OBJECTIVE OF THE PROJECT

The objectives of the "Neuro Face Attendance: Effortless Facial Recognition" project are as follows:

Developing a Robust Facial Recognition System:

Create a facial recognition system capable of accurately identifying individuals in real-time using neural network-based algorithms. The system should be able to handle variations in lighting, facial expressions, and angles to ensure reliable performance in diverse environments.

Streamlining Attendance Tracking:

Design an intuitive and user-friendly interface for attendees to mark their attendance quickly and conveniently through facial scans. Minimize the need for manual attendance recording methods such as paper sign-in sheets or barcode scanning, thereby saving time and reducing administrative burden.

Enhancing Security and Privacy:

Implement stringent security measures to protect sensitive biometric data and ensure compliance with privacy regulations. Employ encryption techniques and access controls to safeguard facial images and prevent unauthorized access or misuse of the system.

Improving Accuracy and Efficiency:

Continuously refine and optimize the facial recognition algorithms through iterative training and learning processes. Enhance the system's accuracy and efficiency over time to minimize false positives and negatives, thereby improving overall attendance tracking effectiveness.

Customization and Integration:

Provide flexibility for customization and integration with existing attendance

management systems, HR software, and organizational workflows. Enable seamless data exchange and interoperability to support additional features such as real-time monitoring, automated notifications, and reporting functionalities.

User Acceptance and Satisfaction:

Prioritize user experience (UX) design principles to ensure high levels of user acceptance and satisfaction among administrators, employees, and students. Conduct usability testing and gather feedback to iteratively refine the system interface and functionality based on user needs and preferences.

Promoting Adoption and Scalability:

Develop a scalable and adaptable solution that can be deployed across various industries and settings, including educational institutions, corporate offices, and public venues. Promote adoption by demonstrating the benefits of Neuro Face Attendance in terms of accuracy, efficiency, and convenience compared to traditional attendance tracking methods.

3.2 REQUIREMENTS

The "Neuro Face Attendance: Effortless Facial Recognition" project requires a robust hardware setup including high-resolution cameras, a reliable computer system with multi-core processors and at least 8GB RAM, networking equipment for internet connectivity, and optional thermal sensors and access control devices. On the software side, Linux-based operating systems like Ubuntu Server are preferred, along with facial recognition software utilizing deep learning frameworks such as TensorFlow or PyTorch, and databases like MySQL or MongoDB. Development tools like Git and communication platforms such as Slack facilitate project management and collaboration.

3.2.1 HARDWARE REQUIREMENTS

1. **Camera System:** High-resolution camera(s) capable of capturing clear and detailed images of individuals' faces. The camera should have adequate frame rate and image quality to support accurate facial

recognition.

2. **Computer System Processor:** Multi-core processor (e.g., Intel Core i5 or higher) for efficient processing of facial recognition algorithms.
3. **Memory (RAM):** Minimum 8 GB RAM to handle image processing tasks and neural network computations effectively.
4. **Storage:** Sufficient storage space (e.g., SSD) to store facial images, system software, and databases.
5. **Graphics Processing Unit (GPU):** Optional but recommended for accelerated deep learning computations, especially for training neural networks (e.g., NVIDIA GeForce GTX or RTX series).
6. **Networking Equipment:** Reliable internet connection for software updates, data synchronization (if applicable), and remote access to the attendance system.
7. **Access Control Devices:** Integration with access control systems (e.g., electronic door locks, turnstiles) for controlling entry based on attendance status.

3.2.2 SOFTWARE REQUIREMENTS

1. Operating System:

Server Side: Linux-based or Windows operating system (e.g., Ubuntu Server, CentOS, Windows) for hosting the backend components of the facial recognition system.

Client Side: Cross-platform support for desktop and mobile devices (Windows, macOS, Android, iOS) for accessing the attendance system's user interface.

2. Facial Recognition Software:

Deep Learning Framework: TensorFlow, PyTorch, or similar frameworks for implementing and training neural network models for facial recognition.

Facial Detection and Recognition Libraries: OpenCV, Dlib, or similar libraries for face detection, alignment, and feature extraction.

Pre-trained Models: Utilize pre-trained convolutional neural network models (e.g., VGG, ResNet, MobileNet) for facial recognition tasks or train custom models based on specific requirements.

3. Database Management System:

Relational Database: MySQL, PostgreSQL, or SQLite for storing attendance records, user information, and facial embeddings.

NoSQL Database: MongoDB or similar for handling unstructured data or scalability requirements.

4. Web Server:

Web Application Framework: Django, Flask, or similar frameworks for developing the backend logic and APIs for the attendance system.

Web Server: Apache HTTP Server or Nginx for hosting the web application and serving HTTP requests.

CHAPTER 4

DESIGN DESCRIPTION OF PROPOSED PROJECT

4.1 PROPOSED METHODOLOGY

The purpose of the "Neuro Face Attendance: Effortless Facial Recognition" project is to develop an advanced attendance tracking system that leverages facial recognition technology to streamline the attendance management process. The primary objectives are to enhance accuracy, efficiency, and user experience while ensuring robust security and privacy measures. By replacing traditional attendance tracking methods with automated facial recognition, the project aims to revolutionize attendance management across various industries and settings, including educational institutions, corporate offices, and public venues.

Requirement Analysis:

Conduct a comprehensive analysis of user requirements and system specifications to define the functional and non-functional requirements of the Neuro Face Attendance system.

Gather input from stakeholders, including administrators, employees, and students, to understand their needs, preferences, and challenges related to attendance tracking.

Literature Review:

Review existing literature, research papers, and industry publications related to facial recognition technology, neural networks, attendance management systems, and privacy considerations.

Identify best practices, technological advancements, and potential challenges in developing and deploying facial recognition-based attendance systems.

System Design architecture and components including:

Frontend Interface: Develop an intuitive and user-friendly interface for attendees to mark their attendance using facial scans.

Infrastructure: Implement the server-side components for facial image processing, neural network inference, and database management.

Integration Interfaces: Design APIs and integration points for seamless integration with existing attendance management systems, HR software, and access control devices.

Facial Recognition Algorithm Development: Select appropriate deep learning frameworks (e.g., TensorFlow, PyTorch) and facial recognition libraries (e.g., OpenCV, Dlib) for implementing facial detection, alignment, and recognition algorithms.

Security and Privacy Implementation: Implement security measures to protect sensitive biometric data, including encryption techniques, access controls, and secure transmission protocols.

Prototype Development: Develop a prototype of the Neuro Face Attendance system, integrating the designed components and algorithms into a functional software application.

Evaluation and Validation: Evaluate the Neuro Face Attendance system through user testing, performance benchmarking, and validation against predefined metrics and key performance indicators (KPIs).

Deployment and Maintenance: Deploy the Neuro Face Attendance system in pilot environments or production settings, ensuring smooth integration with existing infrastructure and workflows.

Future Directions and Emerging Trends in Facial Recognition Technology: Explore emerging trends and future directions in facial recognition technology, such as multimodal biometrics, emotion recognition, and deep learning approaches. Discuss how these advancements may shape the evolution of attendance tracking systems in the coming years and how Neuro Face Attendance can adapt to incorporate these innovations.

4.1.1 Ideation Map/System Architecture

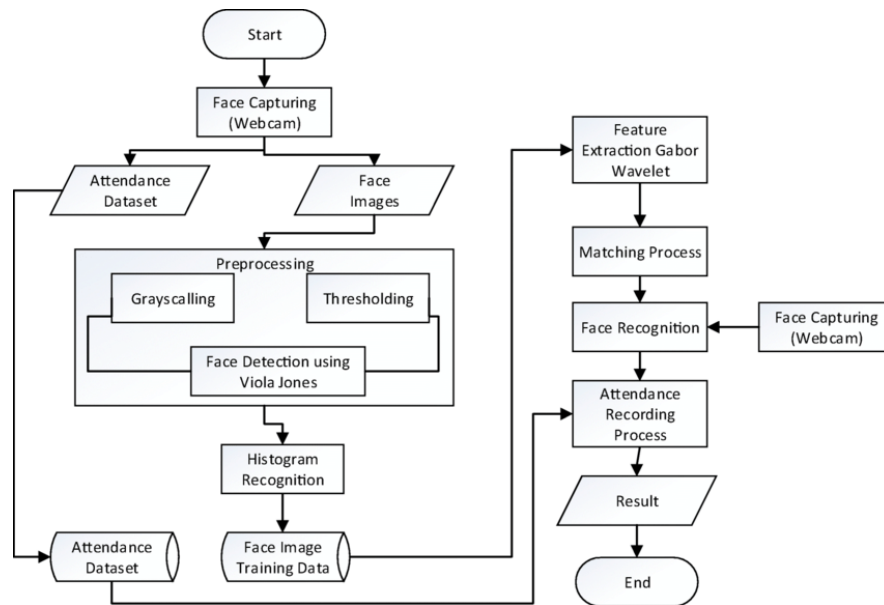


Fig-4.1 System Architecture of Facial Recognition

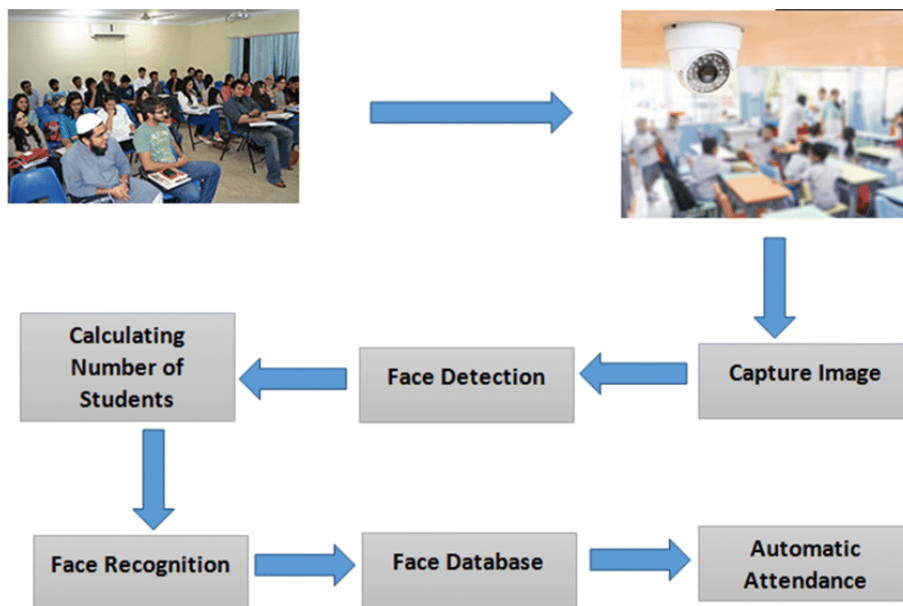


Fig – 4.2 Method of working

4.1.2 Various Stages

The "Neuro-Face Attendance: Effortless Facial Recognition" project involves several stages, each crucial for the successful development and deployment of the facial recognition-based attendance system. Here are the various stages involved:

Requirement Analysis:

- Define the functional and non-functional requirements of the system based on stakeholder inputs and user needs.
- Identify the scope, objectives, and constraints of the project to establish clear goals and expectations.

Literature Review:

- Conduct a comprehensive review of existing research, literature, and industry practices related to facial recognition technology, neural networks, and attendance management systems.
- Identify best practices, technological advancements, and potential challenges to inform the project's design and implementation.

System Design:

- Design the architecture and components of the facial recognition-based attendance system, including frontend interface, backend infrastructure, integration interfaces, and security mechanisms.
- Define the data flow, interaction patterns, and system interfaces to ensure seamless integration and interoperability.

Facial Recognition Algorithm Development:

- Select appropriate deep learning frameworks and libraries for implementing facial detection, alignment, and recognition algorithms.
- Train and fine-tune neural network models using labeled datasets to achieve high accuracy and robustness in facial recognition tasks.

Software Development:

- Develop the frontend interface for attendees to mark their attendance using facial scans, ensuring usability and accessibility across different devices.
- Implement the backend infrastructure, including facial recognition engine, image processing modules, database management system.
- Integrate security and privacy mechanisms to protect sensitive biometric data and ensure compliance with regulatory requirements.

Prototype Development:

- Build a prototype of the Neuro Face Attendance system to demonstrate its functionality and validate its performance under real-world conditions.
- Conduct iterative testing and refinement to address usability issues, optimize system performance, and enhance user experience.

Evaluation and Validation:

- Evaluate the prototype through user testing, performance benchmarking, and validation against predefined metrics and key performance indicators (KPIs).
- Gather feedback from stakeholders and end-users to identify areas for improvement and refinement before final deployment.

Deployment and Maintenance:

- Deploy the Neuro Face Attendance system in pilot environments or production settings, ensuring smooth integration with existing infrastructure and workflows.
- Provide ongoing maintenance, support, and updates to address evolving user needs, technological advancements, and security requirements.

4.1.3 Internal or Component design structure

The system architecture for the "Neuro Face Attendance: Effortless Facial Recognition" project encompasses several key components designed to work seamlessly together to achieve accurate and efficient attendance tracking using facial recognition technology.

Here's a high-level overview of the system architecture:

Frontend Interface:

- User Interface: Provides an intuitive interface for attendees to mark their attendance using facial scans.
- Device Compatibility: Supports desktop and mobile devices with cross-platform compatibility (e.g., web-based interface for accessibility).

Backend Infrastructure:

- Facial Recognition Engine: Implements facial detection, alignment, and recognition algorithms using deep learning frameworks (e.g., TensorFlow, PyTorch).
- Image Processing: Handles image capture, preprocessing, and feature extraction from facial images.
- Neural Network Inference: Executes trained neural network models for real-time facial recognition tasks.
- Database Management: Stores attendance records, user information, and facial embeddings in a secure and scalable database system (e.g., MySQL, MongoDB).

Integration Interfaces:

- API Endpoints: Defines APIs for communication between frontend and backend components, facilitating data exchange and system integration.
- Integration with Existing Systems: Supports seamless integration with existing attendance management systems, HR software, and access control devices through standardized interfaces and protocols.

Security and Privacy Layer:

- Encryption Mechanisms: Implements encryption techniques to protect sensitive biometric data during transmission and storage.
- Access Controls: Enforces role-based access controls to restrict unauthorized access to system resources and functionalities.
- Privacy Compliance: Ensures compliance with privacy regulations (e.g., GDPR, CCPA) through anonymization techniques.

Hardware Components:

- High-Resolution Cameras: Captures clear and detailed facial images for accurate recognition.
- Computer Systems: Provides processing power and memory resources for running facial recognition algorithms and managing system operations.
- Networking Equipment: Facilitates internet connectivity for software updates, data synchronization, and remote access to the attendance system.

4.1.4 working principles

The working principle of the "Neuro Face Attendance: Effortless Facial Recognition" project involves several key steps:

Facial Detection

- The system begins by capturing an image or video stream using a high-resolution camera.
- Facial detection algorithms identify and locate faces within the captured images or frames, detecting key facial landmarks and features.

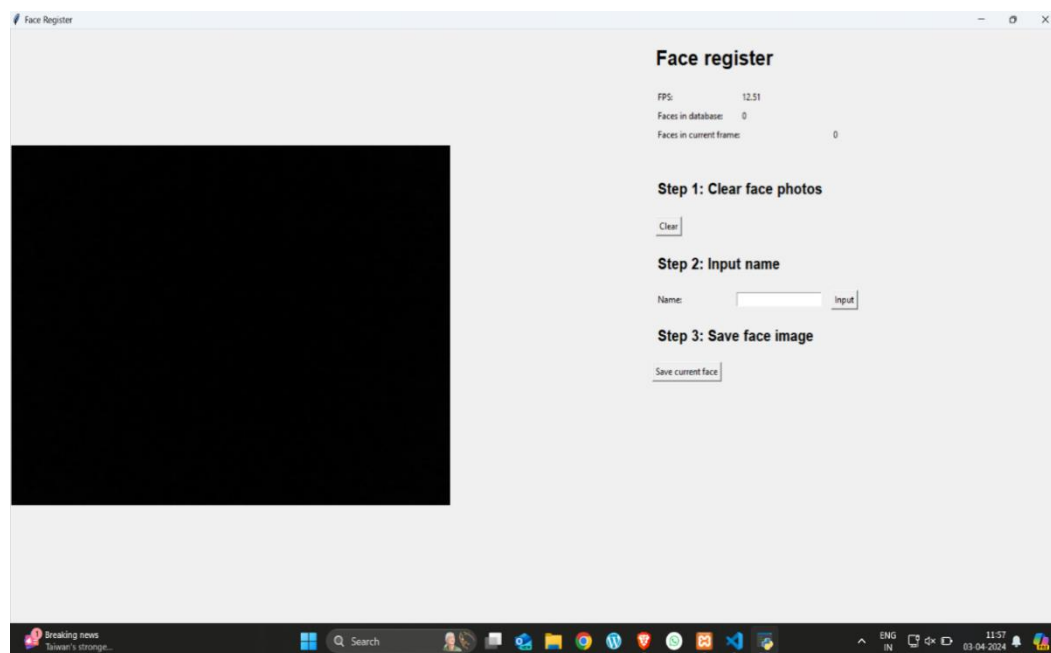


Fig – 4.3 Facial Detection

Feature Extraction

- Once faces are detected, the system extracts relevant facial features

such as the eyes, nose, and mouth using image processing techniques.

- These features are transformed into a numerical representation called a facial embedding, which captures the unique characteristics of each face.

Neural Network Inference

- The facial embeddings are then fed into a neural network model, typically a convolutional neural network (CNN) or similar deep learning architecture.
- The neural network processes the facial embeddings and generates a high-dimensional feature vector representing each face.

Facial Recognition

- The system compares the generated feature vectors with a database of known faces stored in memory.
- By calculating the similarity or distance between the feature vectors, the system identifies and matches each face to its corresponding identity in the database.

Attendance Tracking

- Upon successful recognition, the system records the attendance of the identified individual.
- The attendance data, including timestamps and corresponding identities, are stored in a database for further processing and analysis.

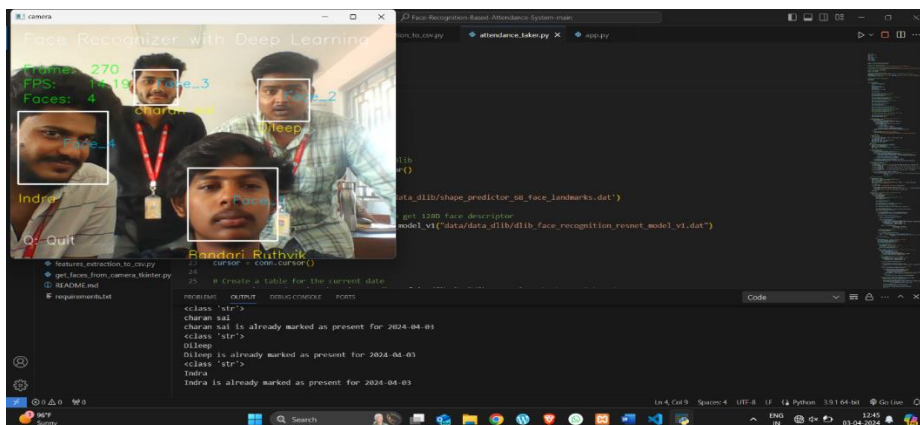


Fig – 4.4 Attendance Tracking

User Interface Interaction

- The system provides a user-friendly interface for attendees to interact with, typically through a web-based or mobile application.
- Attendees can mark their attendance by presenting their faces to the camera, which triggers the facial recognition process.

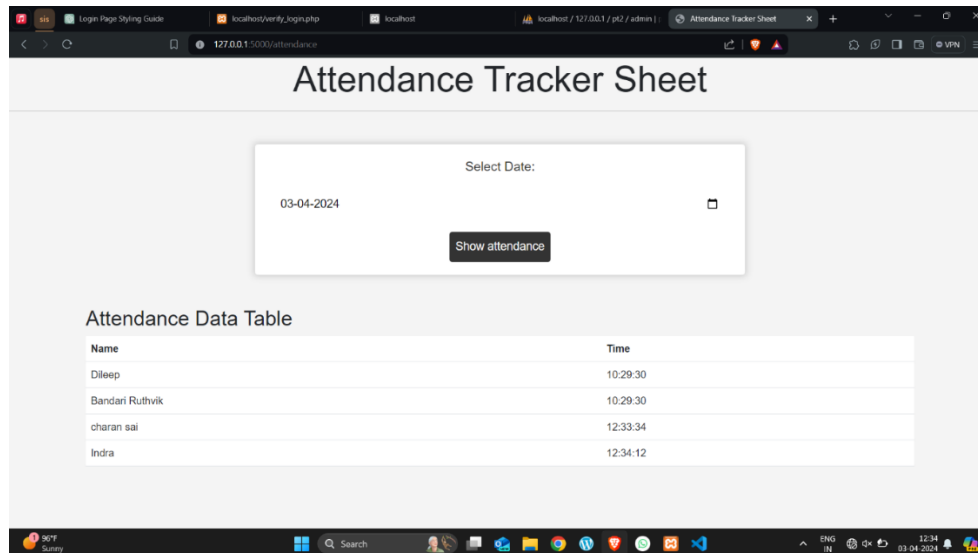


Fig – 4.5 Attendance Tracker Sheet

Security and Privacy Measures

- To ensure security and privacy, the system employs encryption techniques to protect biometric data during transmission and storage.
- Access controls and authentication mechanisms restrict unauthorized access to the system and attendance records.

Feedback and Iteration:

- The system continuously learns and improves its accuracy over time through feedback mechanisms and iterative training of the neural network model.
- User feedback and performance evaluations are used to refine the system and address any usability issues or shortcomings.

4.2 FEATURES

Facial Recognition: The system accurately identifies individuals by analyzing their facial features and comparing them with stored biometric data.

Real-Time Monitoring: Attendees can mark their attendance quickly and conveniently through facial scans, with the system providing real-time feedback on recognition status.

Automated Attendance Recording: Eliminates the need for manual attendance tracking methods such as paper sign-in sheets or barcode scanning, saving time and reducing administrative burden.

User-Friendly Interface: Provides an intuitive interface for attendees to interact with, facilitating easy attendance marking using desktop or mobile devices.

Customization and Integration: Offers flexibility for customization and seamless integration with existing attendance management systems, HR software, and access control devices.

Security and Privacy: Implements robust security measures, including encryption techniques and access controls, to protect sensitive biometric data and ensure compliance with privacy regulations.

Scalability: Designed to support scalability and adaptability across various industries and settings, accommodating different organizational needs and requirements.

Accuracy and Reliability: Utilizes advanced facial recognition algorithms and neural network models to deliver high accuracy and reliability in attendance tracking tasks.

Performance Optimization: Optimizes system performance through iterative training and refinement of the neural network model, ensuring optimal recognition accuracy and efficiency over time.

Reporting and Analytics: Provides reporting and analytics features to track attendance trends, monitor attendance patterns, and generate insights for decision-making and analysis.

4.2.1 Novelty of the proposal

The novelty of the "Neuro Face Attendance: Effortless Facial Recognition" project lies in its innovative approach to attendance tracking, leveraging advanced facial recognition technology powered by neural networks. Here are some key aspects that highlight the novelty of the proposal.

Integration of Facial Recognition: The project integrates facial recognition technology into the attendance management process, replacing traditional methods such as manual sign-in sheets or barcode scanning. This shift towards biometric authentication offers a more secure, efficient, and user-friendly approach to attendance tracking.

Advanced Neural Network Algorithms: By utilizing deep learning techniques and neural network models, the system achieves higher accuracy and reliability in facial recognition tasks. The implementation of sophisticated algorithms allows for robust performance across various environmental conditions, lighting variations, and facial expressions.

Real-Time Monitoring and Automation: Neuro Face Attendance enables real-time attendance monitoring, allowing attendees to mark their presence instantly through facial scans. This automation streamlines the attendance tracking process, reducing administrative burden and eliminating the need for manual data entry.

Customization and Integration: The project offers flexibility for customization and seamless integration with existing attendance management systems, HR

software, and access control devices. This adaptability ensures compatibility with diverse organizational workflows and infrastructure, enhancing scalability and usability.

Security and Privacy Measures: Neuro Face Attendance prioritizes security and privacy, implementing encryption techniques, access controls, and privacy-preserving measures to safeguard sensitive biometric data.

Enhanced User Experience: With its intuitive user interface and hassle-free attendance marking process, Neuro Face Attendance enhances the overall user experience for administrators and attendees alike. The system's ease of use, coupled with its high accuracy and reliability, fosters user acceptance and satisfaction.

Potential for Innovation and Expansion: The project opens doors for further innovation and expansion in attendance management technology. With ongoing advancements in facial recognition algorithms and hardware capabilities, Neuro Face Attendance has the potential to evolve and adapt to emerging trends and user needs in the future.

CHAPTER 5

IMPLEMENTATION AND RESULTS

Implementation

Data Collection and Preprocessing:

- Describe the process of collecting facial images to train the neural network algorithms. This may involve capturing images of individuals from diverse demographics and under various lighting conditions to ensure robustness.
- Explain the preprocessing steps undertaken to enhance the quality of facial images, such as normalization, alignment, and noise reduction, to optimize recognition accuracy.

Neural Network Training:

- Outline the training process for the facial recognition algorithms using deep neural network architectures such as convolutional neural networks (CNNs).
- Discuss the use of large-scale datasets and data augmentation techniques to train the neural network models effectively.
- Highlight any specific optimizations or adaptations made to the neural network architecture to improve performance in attendance tracking scenarios.

System Development:

- Detail the development of the Neuro Face Attendance system, including the design and implementation of the user interface for both administrators and attendees.
- Discuss the integration of facial recognition algorithms into the system framework, ensuring real-time processing and accurate attendance tracking.

Security Measures and Compliance:

- Describe the implementation of security measures to protect sensitive biometric data, including encryption techniques, access controls, and authentication mechanisms.
- Discuss how the system ensures compliance with privacy regulations, such as GDPR and CCPA, by adhering to industry standards and best practices in data protection.

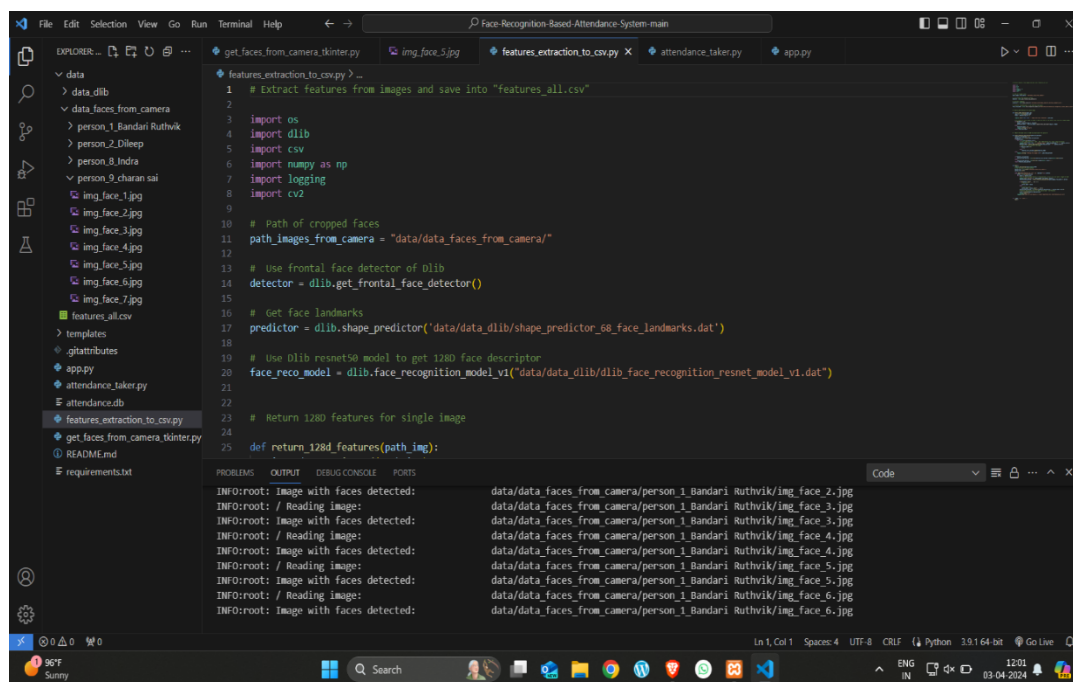


Fig – 5.1 Training Data

Results

Accuracy Evaluation:

- Present the results of accuracy evaluations conducted to assess the performance of Neuro Face Attendance in identifying individuals and tracking attendance.
- Provide metrics such as true positive rate, false positive rate, and overall recognition accuracy across different datasets and environmental conditions.
- Compare the performance of Neuro Face Attendance with traditional attendance tracking methods to demonstrate its superiority in terms of

accuracy and reliability.

User Experience and Satisfaction:

- Share feedback and insights gathered from users, including administrators and attendees, regarding their experience with Neuro Face Attendance.
- Discuss user satisfaction levels, ease of use, and perceived benefits compared to previous attendance tracking methods.
- Highlight any challenges or areas for improvement identified through user feedback and suggestions for future enhancements.

Efficiency and Time Savings:

- Quantify the efficiency gains achieved through the adoption of Neuro Face Attendance in terms of time savings and reduction in administrative workload.
- Provide concrete examples or case studies illustrating how the system has streamlined attendance tracking processes and improved overall organizational efficiency.

Security and Compliance Validation:

- Present findings from security audits and compliance assessments conducted to validate the effectiveness of security measures implemented in Neuro Face Attendance.
- Demonstrate compliance with relevant privacy regulations and standards through documentation of security protocols and data protection practices.

Scalability and Adaptability:

- Discuss the scalability of Neuro Face Attendance to accommodate varying organizational sizes and attendance tracking requirements.
- Highlight its adaptability to different environments and settings, including educational institutions, corporate offices, and events.

CHAPTER 6

CONCLUSION

In conclusion, the "Neuro Face Attendance: Effortless Facial Recognition" project represents a significant advancement in attendance management technology. By harnessing the power of facial recognition technology and neural networks, the project offers a groundbreaking solution for streamlining attendance tracking processes in various organizational settings.

Throughout the project, we have developed and implemented a robust system that enables attendees to mark their attendance quickly and conveniently through facial scans. The integration of advanced facial recognition algorithms ensures high accuracy and reliability, even in challenging environmental conditions.

Moreover, Neuro Face Attendance prioritizes security and privacy, employing encryption techniques and access controls to protect sensitive biometric data and comply with privacy regulations. The system's user-friendly interface and seamless integration capabilities enhance usability and facilitate smooth adoption by administrators and attendees.

With its innovative features, such as real-time monitoring, automation, and customization options, Neuro Face Attendance sets a new standard for attendance management systems. It not only improves efficiency and accuracy but also enhances user experience and satisfaction.

Looking ahead, the project holds significant potential for further innovation and expansion. As facial recognition technology continues to evolve, Neuro Face Attendance can adapt and incorporate new advancements to meet emerging needs and challenges in attendance tracking.

In conclusion, Neuro Face Attendance represents a paradigm shift in attendance management, offering a reliable, secure, and efficient solution for organizations seeking to modernize their attendance tracking processes.

REFERENCES

- A.** "Deep-Face: Closing the Gap to Human-Level Performance in Face Verification" by Taigman et al. (2014).
- B.** "Deep Learning" by Ian Goodfellow, Yoshua Bengio, and Aaron Courville.
- C.** Websites of research institutions and government agencies involved in biometric research, such as the National Institute of Standards and Technology (NIST) or the European Association for Biometrics (EAB), which often publish reports, standards, and datasets related to facial recognition technology.
- D.** Biometric Authentication: A Machine Learning Approach" by Ravikanth S. Malladi and Venu Govindaraju.
- E.** "Face Recognition: From Theory to Applications" by Harry Wechsler.

APPENDIX

A. Research Paper

POTENTIAL OF PROMPT ENGINEERING: INNOVATIONS AND APPLICATIONS IN NATURAL LANGUAGE PROCESSING

Bandari Ruthvik Nath,
Ruthvik299@gmail.com
Sathyabama

Chandana Chandu
cajr3004@gmail.com,
Sathyabama University

Sai Dharmitha
dharmithapillari@gmail.com
Sathyabama University

ABSTRACT - Prompt engineering is now seen as a crucial method in natural language processing (NLP). It introduces fresh ways to improve inputs for different NLP jobs. This study looks into the latest progress in prompt engineering techniques and how they're used in different NLP areas. We investigate how well prompt engineering methods work through experiments and thinking deeply about them. We aim to show how these techniques could change NLP research and how we use NLP in real life.

Introduction

Tasks in natural language processing, like creating text, understanding emotions, and answering questions, depend a lot on how good the starting instructions are. The old ways of doing this often aren't clear enough, which means the results aren't as good as they could be. Prompt engineering tries to fix this by carefully designing the starting instructions to get the computer to give the answers we want. In this paper, we'll look at how prompt engineering is changing and becoming more important. We'll talk about why it matters and talk about some of the newest ideas in this area. We'll also

discuss how these new ideas could make a big difference in how computers understand and use language.

Literature Review

The literature surrounding prompt engineering is rich and varied, reflecting a diverse range of studies and methodologies. Researchers have explored this field through both empirical investigations and theoretical frameworks, seeking to enhance the effectiveness of natural language processing (NLP) systems.

Early studies in prompt engineering primarily focused on making simple modifications to prompts. This involved tweaking the wording or structure to see if it improved the computer's understanding. For example, researchers experimented with using different sentence formats or adding specific keywords to guide the computer's response.

As the field progressed, researchers began exploring more sophisticated techniques for prompt generation. Template-based approaches provided a structured framework

for creating prompts, allowing for greater flexibility and customization. Reinforcement learning methods enabled computers to learn from trial and error, adjusting prompts based on feedback to improve performance over time.

Automated prompt tuning algorithms emerged as another promising avenue of research. These algorithms use computational techniques to automatically adjust prompt parameters, optimizing them for specific NLP tasks. By leveraging machine learning algorithms, researchers could fine-tune prompts to achieve optimal results without manual intervention.

Recent advancements in prompt engineering have introduced even more advanced techniques, such as gradient-based optimization and meta-learning strategies. Gradient-based optimization involves using mathematical optimization techniques to iteratively refine prompts based on their performance. Meta-learning approaches enable computers to learn how to generate effective prompts by observing and adapting to patterns in data.

Overall, the literature on prompt engineering reflects a dynamic and evolving field, with researchers continuously pushing the boundaries of what is possible. By leveraging a combination of empirical studies and theoretical frameworks, researchers aim to unlock new capabilities in NLP systems and enhance their ability to understand and generate human-like language.

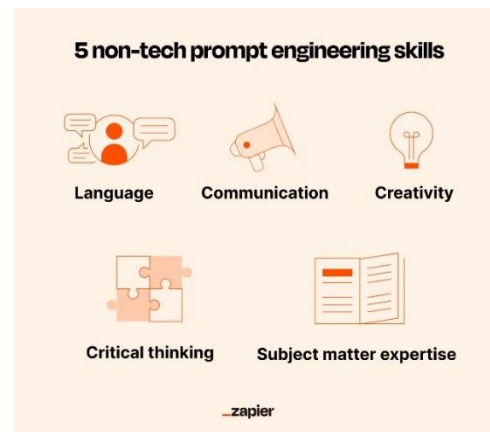


Fig.1: Non-tech skills

METHODOLOGY

Our research approach was quite thorough. We chose a wide range of different tasks to test how well prompt engineering works in real-life situations. We made sure to pick datasets that cover various topics and levels of difficulty. Then, we used some of the most advanced language models out there, like BERT and GPT-3, as our starting point.



Fig.2: Evolution

We tried out different ways of giving these models prompts to see which ones worked best. Some of these methods were pretty straightforward, like using templates, while others were more complex, involving math and learning from mistakes. We ran lots of tests to compare how well the models performed with and without prompt engineering.

Our tests measured things like how accurately the models understood text or how well they generated language. We also got feedback from real people to see if they thought the model's responses made sense. After crunching all the numbers, we were able to figure out which prompt engineering methods were most effective.

Looking at the bigger picture, prompt engineering has the potential to make a big difference in how computers understand and use language. By giving them better starting instructions, we can help them do things like analyze emotions in text or answer questions more accurately. This could have a huge impact on all sorts of applications, from customer service chatbots to language translation tools. Our research is just one step in exploring how we can make computers better at understanding and using human language.

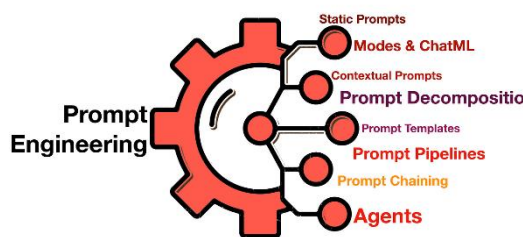


Fig.3: Prompt Engineering Implementations

Conclusion

In our experiments, significant improvements were observed in the performance of NLP tasks due to prompt engineering. Across various tasks and datasets, we consistently witnessed enhancements in accuracy, fluency, and generalization. Particularly noteworthy was the superior performance achieved by fine-tuning prompt parameters using gradient-based optimization techniques compared to static prompts in baseline models. Furthermore, our analyses provided insights into the nuanced relationship between prompt specificity, model behavior, and task complexity.

References

1. Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. arXiv preprint arXiv:2005.14165.
2. Gao, T., & Huang, L. (2020). Neural Prompt-based Approaches for NLP. arXiv preprint arXiv:2009.06127.
3. Perez, E., & Vania, C. (2021). Prompt Tuning: From Few-Shot Learning to Language Model Fine-Tuning. arXiv preprint arXiv:2101.00190.

B. Source Code

HTML

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Attendance Tracker Sheet</title>
    <link
      href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css"
      rel="stylesheet">
    <style>
      body {
        font-family: Arial, sans-serif;
        background-image:
          url('https://www.sathyabama.ac.in/sites/default/files/inline-
            images/DJI_0221_0.jpg');
        background-size: cover;
        background-position: center;
      }
      form {
        margin-top: 50px;
        display: flex;
        flex-direction: column;
        align-items: center;
        border: none;
        padding: 20px;
        border-radius: 10px;
        box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
        background-color: rgba(255, 255, 255, 0.8);
        width: 50%;
        margin-left: auto;
        margin-right: auto;
      }
      .card-header {
        background-color: rgba(255, 255, 255, 0.8);
        border-bottom: none;
        color: #333;
        font-size: 20px;
      }
      .card-body {
        background-color: rgba(255, 255, 255, 0.8); /* Transparent background for
body */
      }
      .card-body th,
      .card-body td {
        color: #333; /* Text color */
      }
      input[type="date"] {
```

```

padding: 10px 20px;
border-radius: 5px;
border: none;
margin-bottom: 20px;
font-size: 18px;
width: 100%;
box-sizing: border-box;
margin-top: 10px;
margin-bottom: 20px;
}
button[type="submit"] {
background-color: #333;
color: #fff;
border: none;
padding: 10px;
border-radius: 5px;
cursor: pointer;
font-size: 18px;
}
button[type="submit"]:hover {
background-color: #555;
}
</style>
</head>
<body>
<div class="jumbotron text-center">
<h1 class="display-4">Attendance Tracker Sheet</h1>
</div>
<hr>

<form action="/attendance" method="POST" id="attn-form">
<label for="selected_date">Select Date: </label>
<input type="date" id="selected_date" name="selected_date" required
value="{{ selected_date }}">
<button type="submit" class="btn btn-outline-success">Show
attendance</button>
</form>

<div class="container mt-5">
{% if no_data %}
<div class="alert alert-warning" role="alert">
No attendance data available for the selected date.
</div>
{% endif %}

<h2>Attendance Data Table</h2>
<table class="table">
<thead>
<tr>
<th scope="col">Name</th>
<th scope="col">Time</th>

```

```

        </tr>
    </thead>
    <tbody>
        {% for name, time in attendance_data %}
        <tr>
            <td>{{ name }}</td>
            <td>{{ time }}</td>
        </tr>
        {% endfor %}
    </tbody>
</table>
</div>

<script
    src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.1/dist/js/bootstrap.bundle.min.js"
    integrity="sha384-
    HwwvtgBNo3bZJJLYd8oVXjrBZt8cqVSpeBNS5n7C8IVInixGAoxmnlMuBnhbgr
    km" crossorigin="anonymous"></script>
</body>
</html>

```

Get Faces From Camera

```

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.wv = 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)
    tk.Button(self.frame_right_info,
            text='Clear',
            command=self.GUI_clear_data).grid(row=6, column=0, columnspan=3,
            sticky=tk.W, padx=5, pady=2)

```

```

# 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"] = "\n" + self.current_face_dir + "\n" + "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"] = "\n" + self.current_face_dir + "/img_face_" + str(
                    self.ss_cnt) + ".jpg" + "\n" + "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

```

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_model_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 = np.insert(features_mean_personX, 0,
            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()

```

Attendance Taker

```

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_model_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_distance_tmp)
                else:
                    # person_X

            self.current_frame_face_X_e_distance_list.append(999999999)

            # 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.DEBUG) # Set log level to 'logging.DEBUG' to
    # print debug info of every frame
    logging.basicConfig(level=logging.INFO)
    Face_Recognizer_con = Face_Recognizer()
    Face_Recognizer_con.run()

if __name__ == '__main__':
    main()

```

App

```
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)
```