



VIT[®]
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

Course Title : PJT – Product Development Project

Course Code : BCSE394J

Project Title : School Security System using Facial Recognition

Team Members :

- Krish Goel [21BCE6097]**
- Meghna Das [21BCE1422]**
- Aastha Kumar [21BCE5067]**
- Godavari Tanmayi [21BCE5976]**
- T Suraj Kumar [21BCE5922]**

Faculty Supervisor : Dr. Suganya R.

Faculty Co-Supervisor : Dr. Subbulakshmi T.

Date : June 2024

DECLARATION FORM

I hereby declare that the project entitled “**School Security System using Facial Recognition**” submitted by me to the School of Computer Science and Engineering, Vellore Institute of Technology, Chennai Campus, Chennai, 600127 in partial fulfilment of the requirements for the award of the degree of **Bachelor of Technology – Computer Science and Engineering** is a record of the bonafide work carried out by me. I further declare that the work reported in this report has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma of this institute or of any other institute or university.

Signature :

Krish Goel [21BCE6097]

Meghna Das [21BCE1422]

Aastha Kumar [21BCE5067]

Godavari Tanmayi [21BCE5976]

T Suraj Kumar [21BCE5922]

TEAM LIST

1. **Krish Goel [21BCE6097]**
2. **Meghna Das [21BCE1422]**
3. **Aastha Kumar [21BCE5067]**
4. **Godavari Tanmayi [21BCE5976]**
5. **T Suraj Kumar [21BCE5922]**

ACKNOWLEDGEMENT

It is with gratitude that I would like to extend my thanks to the visionary leader Dr. G. Viswanathan our Honorable Chancellor, Mr. Sankar Viswanathan, Dr. Sekar Viswanathan, Dr. G V Selvam Vice Presidents, Dr. Sandhya Pentareddy, Executive Director, Ms. Kadhambari S. Viswanathan, Assistant Vice-President, Dr. V. S. Kanchana Bhaaskaran Vice-Chancellor, Dr. T. Thyagarajan Pro-Vice Chancellor, VIT Chennai and Dr. P. K. Manoharan, Additional Registrar for providing an exceptional working environment and inspiring all of us during the tenure of the course.

In jubilant state, I express ingeniously my whole-hearted thanks to Dr. Nithyanandam P, Head of the Department, B.Tech. CSE and the Project Coordinators for their valuable support and encouragement to take up and complete the thesis.

We are pleased to acknowledge our sincere gratitude to Dr. Suganya R. and Dr. Subbulakshmi T. of Vellore Institute of Technology, Chennai for their kind encouragement in doing this project and completing it successfully. We convey our gratitude to VIT Management for providing us with the necessary support and details timely during the progressive reviews.

My sincere thanks to all the faculty and staff at Vellore Institute of Technology, Chennai who helped me acquire the requisite knowledge. I would like to thank my parents for their support. It is indeed a pleasure to thank my friends who encouraged me to take up and complete this task.

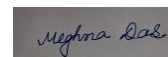
Place: Chennai

Signature:

Krish Goel



Meghna Das



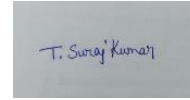
Aastha Kumar



Godavari Tanmayi



T Suraj Kumar



Date: 15.06.2024

INTRODUCTION

In recent years, ensuring the security of educational institutions has become a paramount concern, with schools seeking innovative solutions to protect students and staff. Traditional security measures, such as ID cards and manual attendance, often fall short in providing real-time and efficient security. To address these challenges, our project proposes the development of a School Security System using Facial Recognition technology. This system aims to enhance the security infrastructure of schools by leveraging advanced Facial Recognition algorithms to monitor and control access to the premises.

Facial Recognition technology has gained widespread recognition for its applications in various sectors, including law enforcement, banking, and personal devices. Its accuracy and efficiency in identifying individuals make it an ideal choice for School Security Systems. By integrating this technology, we aim to create a seamless, automated system that can recognize and verify individuals in real-time, thereby preventing unauthorized access and ensuring the safety of the school environment.

The primary objective of this project is to develop a robust Security System that automates the identification process, reducing the reliance on manual checks and mitigating the risks associated with human error. The system will consist of strategically placed cameras that capture images of individuals entering the school premises. These images will be processed and compared against a pre-registered database of authorized personnel, including students, teachers, and staff members. Upon successful identification, access will be granted; otherwise, alerts will be triggered, notifying security personnel of potential intrusions.

In addition to enhancing security, this system aims to streamline administrative processes, such as attendance tracking. By automatically logging the entry and exit times of the students and the staff, the system can generate accurate attendance reports, reducing administrative burdens and improving record-keeping efficiency.

Our project will involve the design and implementation of software components. We will utilize Facial Recognition libraries like OpenCV and Machine Learning frameworks to develop the recognition algorithms. A secure database management system will store facial data and access logs, while a user-friendly interface will allow administrators to manage the system effectively.

The potential impact of this system extends beyond enhanced security. By fostering a safe and secure environment, schools can focus on their primary mission of education. Additionally, the integration of modern technology in school operations sets a precedent for future advancements, paving the way for smarter, more efficient educational institutions.

In conclusion, the School Security System using Facial Recognition presents a forward-thinking solution to contemporary security challenges. By combining advanced technology with practical applications, this project aims to create a safer, more secure environment for educational institutions, ensuring peace of mind for students, parents, and staff alike.

TABLE OF CONTENTS

S.No.	Title	Pg. No.
1.	Title Page	1
2.	Declaration	2
3.	Team List	3
4.	Acknowledgement	4 – 5
5.	Introduction	6
6.	Table of Contents	7
7.	Problem Statement	8
8.	Functionalities of the Model	9 – 10
9.	Product Life Cycle	11 – 12
10.	Embedded Component	13 – 43
	10.1 Flow Diagram	13 – 14
	10.2 Algorithm	15 – 17
	10.3 Code	18 – 43
11.	Backend Server	44
12.	Webapp Preview	45 – 47
13.	Research Paper Details	48
14.	Summary	49 – 50
15.	Conclusion	51 – 52
16.	Future Scope	53 – 54
17.	References	55 – 57

PROBLEM STATEMENT

The conventional method of recording student attendance is frequently fraught with difficulties. The Face Recognition Student Attendance System emphasizes its simplicity by eliminating classical student attendance marking techniques such as calling student names or checking respective identification cards. Not only do they impede the teaching process, but they also divert students' attention during exam periods. During the lecture hours, attendance sheets are distributed throughout the classroom in addition to calling names. It could be challenging to circulate the attendance sheet around the lecture hall, particularly in classes with a lot of pupils. As a result, the manual method of manually signing kids' attendance—which is tedious and distracts students—is to be replaced with a Face Recognition Attendance System. Moreover, the automated Student Attendance System based on Facial Recognition can effectively tackle the issue of fraudulent approaches, hence eliminating the need for lecturers to repeatedly count the number of students to verify their presence.

The work that Zhao, W. et al. (2003) proposed, enumerated the challenges associated with Facial Recognition. Making the distinction between recognized and unidentified photos is one of the challenges in Facial Recognition. Furthermore, it was discovered in a report by Pooja G.R et al. (2010), that the Facial Recognition Student Attendance System's training procedure is laborious and slow. Furthermore, the study put out by Priyanka Wagh et al. (2015) noted that variations in head positions and lighting are frequent issues that could impair the effectiveness of a Face Recognition-based Student Attendance System.

Therefore, in order to prevent omission, the identification procedure must be completed within predetermined time limitations. This calls for the development of a real-time functioning Student Attendance System. The traits that have been retrieved from the students' faces to indicate their identities must remain consistent as the background, lighting, stance, and expression vary. The performance will be assessed using high accuracy and quick computation times.

Creating a Facial Recognition Attendance System is the project's goal. The following results are anticipated in order to meet the goals:

- To identify the facial segment from the video frame.
- To take the identified face and extract its useful traits.
- To categorize the features so that the detected face may be identified.
- To document the designated student's attendance.

FUNCTIONALITIES OF THE MODEL

The School Security System using Facial Recognition encompasses several key functionalities designed to enhance security and streamline administrative processes within educational institutions. These functionalities include Facial Recognition, Access Control, Alert System, Database Management, User Interface, Attendance Tracking, Visitor Management, and Data Analytics.

1. Facial Recognition:

The core functionality of the system is Facial Recognition, which involves capturing images of individuals, entering the school premises using strategically placed cameras. These images are processed using advanced algorithms to identify and verify the individuals. The system compares the captured images with a pre-registered database of authorized personnel, ensuring that only recognized individuals are allowed access. This process is efficient and non-intrusive, providing a seamless security check without causing delays.

2. Access Control:

Based on the results of the Facial Recognition process, the system manages Access Control. Authorized individuals, such as students, teachers, and staff, are granted entry automatically. In the case of unrecognized or unauthorized individuals, access is denied, and appropriate actions are triggered. This automated access control minimizes the need for manual checks, reduces human error, and enhances overall security by ensuring that only permitted individuals enter the premises.

3. Database Management:

The system includes a robust Database Management component to store and manage facial data securely. This database contains records of all authorized individuals, including their facial images and relevant identification details. The system ensures data privacy and security through encryption and access control measures. Additionally, the database logs entry and exit times, which can be used for generating attendance reports and monitoring movement within the school.

4. User Interface:

A User-friendly Interface is provided for administrators to manage and monitor the system effectively. The interface allows for easy registration of new individuals, updating of existing records, and view of access logs and alerts. The dashboard displays real-time data and status updates, enabling administrators to oversee the security operations smoothly. This interface ensures that the system is accessible and manageable, even for users with limited technical expertise.

5. Attendance Tracking:

By automatically logging the entry and exit times of the students and the staff, the system can generate accurate attendance reports. This feature reduces administrative burdens, eliminates the need for manual roll calls, and ensures accurate record-keeping. Attendance data can be integrated with the school's existing management systems to provide a seamless workflow.

6. Data Analytics:

The system can analyze the collected data to provide insights and reports on various security and administrative aspects. For instance, it can generate reports on peak entry times, frequent visitors, and security incidents. These analytics can help school administrators make informed decisions about

resource allocation and identify areas for improvement in the security infrastructure.

7. Integration with Existing Systems:

The Facial Recognition System can be integrated with the school's existing security infrastructure, such as CCTV systems, alarm systems, and access control mechanisms. This integration ensures a cohesive and comprehensive security solution, allowing for centralized management and monitoring of all security-related activities.

8. Scalability and Customization:

The system is designed to be scalable, allowing it to accommodate the needs of schools of various sizes. Customization options enable schools to tailor the system to their specific requirements, such as adding more cameras, adjusting recognition sensitivity, or integrating additional functionalities as needed.

Each of these functionalities work in concert to create a comprehensive security solution for schools. The integration of Facial Recognition with access control and real-time alerts significantly enhances the safety and security of the school environment. Database Management ensures accurate and secure record-keeping, while the User Interface facilitates easy administration. Attendance Tracking and Visitor Management further streamline administrative processes, and Data Analytics provide valuable insights for continuous improvement. Together, these functionalities make the School Security System using Facial Recognition, a powerful tool for modern educational institutions.

PRODUCT LIFE CYCLE

The product life cycle for the School Security System using Facial Recognition can be outlined in several key phases: Concept Development, Design and Planning, Software Development, Testing and Validation, Deployment and Implementation, Maintenance and Support, and Upgrades and Enhancement.

1. Concept Development:

- **Idea Generation:** Identifying the need for enhanced School Security through software solutions, specifically using Facial Recognition technology.
- **Feasibility Study:** Assessing the technical feasibility and potential benefits of developing a software-based Facial Recognition System for School Security.
- **Initial Research:** Conducting market research and consulting with stakeholders, including school administrators and security experts, to gather insights and requirements.

2. Design and Planning:

- **Requirement Analysis:** Gathering detailed requirements from stakeholders and defining the scope of the software project, including user needs, security protocols, and data management requirements.
- **System Design:** Creating detailed design documents, including software architecture, flowcharts, and interface designs. Specifying the algorithms and technologies to be used, such as Facial Recognition libraries (e.g., OpenCV, TensorFlow) and Database Systems (e.g., MySQL Workbench).
- **Project Planning:** Developing a comprehensive project plan, timeline, and budget, identifying key milestones, deliverables, and resource allocation.

3. Software Development:

- **Algorithm Development:** Writing and testing the Facial Recognition algorithms to ensure accuracy and efficiency in identifying individuals.
- **Database Development:** Setting up the database to securely store facial data, access logs, and user information. Implementing data encryption and access control measures to protect sensitive information.
- **Interface Development:** Designing and developing a User-friendly Interface for administrators to manage the system, register new users, and monitor access logs and alerts. Creating a responsive web or mobile application for real-time access and control.
- **Integration:** Ensuring seamless integration between the Facial Recognition algorithms, the database, and the user interface components.

4. Testing and Validation:

- **Unit Testing:** Testing individual software modules and components to ensure they function correctly and meet design specifications.
- **Integration Testing:** Verifying that all software components work together seamlessly, ensuring that data flows correctly between the Facial Recognition algorithms, the database, and the user interface.

- **User Testing:** Conducting user acceptance testing with actual users (e.g., school administrators, teachers) to validate the system's usability, functionality, and effectiveness in real-world scenarios.
- **Validation:** Ensuring that the software meets all specified requirements and making any necessary adjustments based on user feedback and test results.

5. Deployment and Implementation:

- **Installation:** Deploying the software system on the school's existing infrastructure, including setting up the server and configuring the database.
- **Training:** Providing training sessions and documentation for administrators and users on how to use the system effectively, including managing user registrations, monitoring access logs, and responding to alerts.
- **Initial Rollout:** Gradually rolling out the software system, starting with a pilot phase to ensure smooth integration into daily operations and address any initial issues.

6. Maintenance and Support:

- **Ongoing Support:** Offering technical support to address any software-related issues or malfunctions that arise. Providing regular updates and patches to improve system performance and security.
- **Maintenance:** Performing regular maintenance tasks, such as software updates, database optimization, and backup procedures to ensure data integrity and system reliability.
- **User Feedback:** Continuously gathering feedback from users to identify areas for improvement and enhance the user experience.

7. Upgrades and Enhancement:

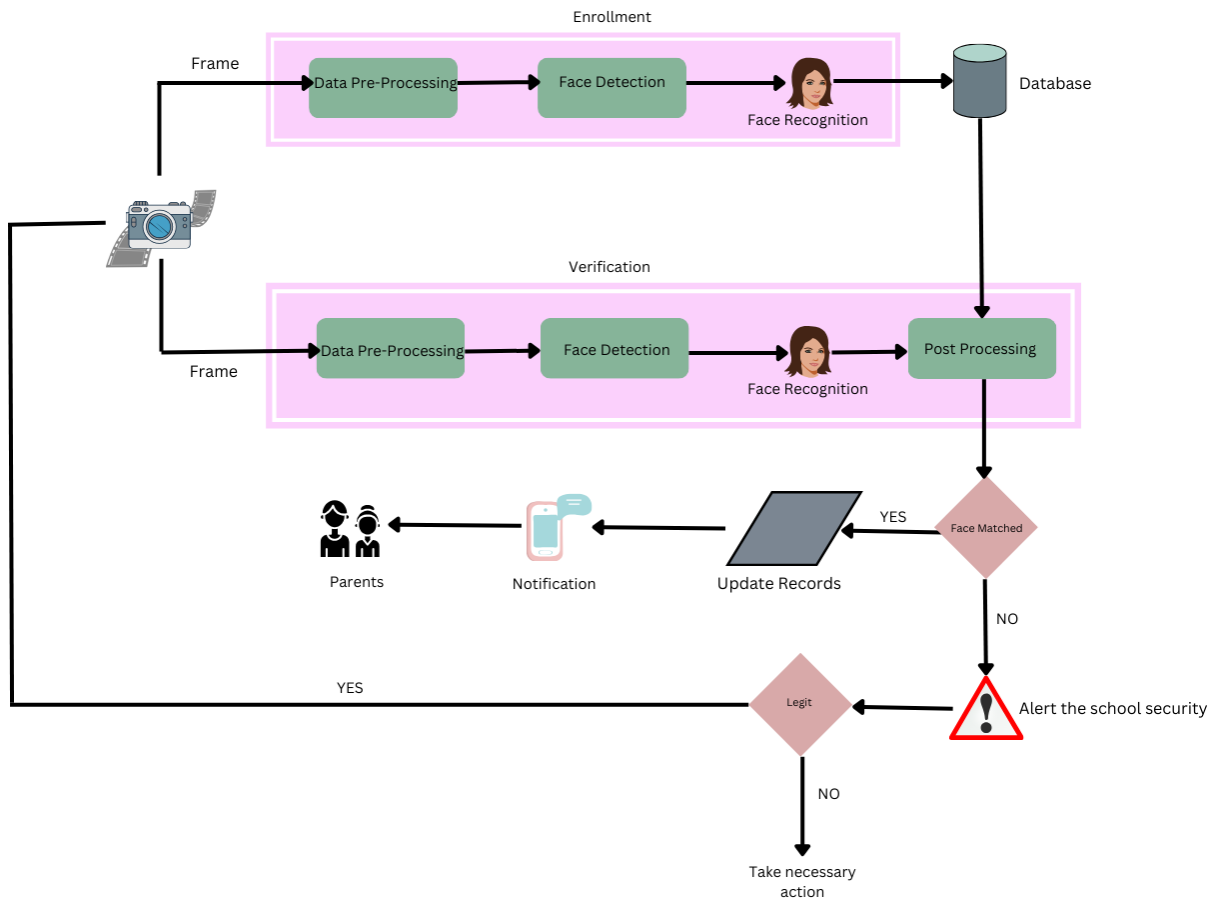
- **Feature Enhancements:** Adding new features or improving existing ones based on user feedback, technological advancements, and evolving security needs. This could include additional analytics, enhanced reporting capabilities, or improved user interfaces.
- **Scalability:** Ensuring the software can scale to accommodate growing needs, such as handling more users, processing higher volumes of data, or integrating with additional systems.
- **Technology Updates:** Keeping the software up-to-date with the latest advancements in Facial Recognition technology and security measures, ensuring that the system remains effective and secure.

8. End-of-Life Considerations:

- **Evaluation:** Periodically evaluating the software system's performance and relevance to determine when updates or replacements are necessary.
- **Decommissioning:** Planning for the eventual decommissioning of the software, including data migration, user notifications, and system shutdown procedures.
- **Replacement:** Considering new technologies or software solutions that can replace or enhance the current system to continue providing robust security and operational efficiency.

EMBEDDED COMPONENT

Flow Diagram:



Enrollment Process:

1. **Take a picture of a person's face:** The process starts with capturing a picture of the person to be enrolled in the system.
2. **Frame:** The captured image is formatted for further processing.
3. **Data Pre-Processing:** The image data is prepared for facial recognition. This may involve tasks like noise reduction, contrast adjustment, and normalization.
4. **Face Detection:** The system identifies the presence and location of a face in the image frame.
5. **Face Recognition:** Facial features are extracted from the detected face and converted into a data set called a faceprint.
6. **Database:** The extracted faceprint is stored in a database along with a unique identifier for the person.

Verification Process:

1. **Take a picture:** A picture is taken of a person seeking verification.
2. **Frame:** The captured image is formatted for further processing.
3. **Data Pre-Processing:** Similar to enrollment, the image data is prepared for facial recognition.
4. **Face Detection:** The system again identifies the presence and location of a face in the image frame.
5. **Face Recognition:** Facial features are extracted from the detected face and converted into a faceprint, just like in enrollment.
6. **Post Processing:** The system may perform additional processing on the faceprint from the verification image.
7. **Database:** The faceprint from the verification image is compared to the faceprints stored in the database.
8. **Face Matched:** The system determines if the faceprint from the verification image matches a faceprint in the database.
 - **Yes:** If there is a match, the system proceeds to process based on whether it is enrollment or verification mode.
 - **Enrollment:**
 - **Parents Notification:** Parents or guardians are informed about the enrollment.
 - **Update Records:** The person's record in the database is updated.
 - **Verification:**
 - **Legit:** If the system is in verification mode and there is a match, the person is granted access or otherwise confirmed as legitimate.
 - **No:** If there is no match, the system proceeds differently depending on the mode.
 - **Enrollment:** No action is taken, as the person is not yet enrolled in the system.
 - **Verification:**
 - **Alert the school security:** An alert is sent to security personnel to investigate why an unrecognized person is trying to gain access.
 - **Take necessary action:** School security will take appropriate action based on their procedures.

ALGORITHM

Haar Cascades

Haar Cascades use Haar-like features, which are digital image features used to detect objects in an image or video. These features are computed by taking the sum of pixel intensities in adjacent rectangular regions and calculating their difference. To compute Haar-like features efficiently, Haar Cascades use integral images, which are intermediate representations that allow rapid computation of the sum of pixel intensities in a rectangular region. Haar Cascades employ a cascade of classifiers, where each successive classifier is trained to focus on the remaining objects which passed through the previous stage. This allows the algorithm to quickly discard negative regions (non-objects) and focus on promising object-like regions. Haar Cascades are particularly effective for Face Detection because they can rapidly scan an image for potential face-like regions, and then verify or reject each region using the cascade of classifiers. While Haar Cascades were initially designed for Face Detection, they can be trained to detect other types of objects as well, such as pedestrians, cars, or any object with distinct features.

Haar Cascades were widely used in computer vision applications, especially before the advent of Deep Learning-based Object Detection methods like Convolutional Neural Networks (CNNs). They are computationally efficient and can achieve real-time performance, making them suitable for applications like video surveillance or human-computer interaction.

LBPHs (Local Binary Pattern Histograms)

Local Binary Patterns (LBPs) is a simple yet very efficient texture operator which labels the pixels of an image by thresholding the neighborhood of each pixel and considers the result as a binary number. Since its initial description in 1994 (LBPs), it has been discovered to be an effective feature for texture categorization. Furthermore, for some datasets, it has been found that combining LBPs with the Histogram of Oriented Gradients (HOGs) descriptor significantly increases the detection performance. We may use a straightforward data vector to represent the facial photos by combining the LBPs with the Histograms.

The LBPH algorithm works in 5 steps :

1. Parameters: The LBPH algorithm uses 4 parameters:

- Radius: It denotes the area surrounding the central pixel and is utilized to construct the circular Local Binary Pattern. It is usually set to 1.
- Neighbors: The quantity of sample points needed to create a Local Binary Pattern that is circular. Remember that the computational cost increases with the number of sample points you provide. Normally, it is set to 8.
- Grid X: The quantity of cells arranged horizontally. Higher the dimensionality the generated feature vector has, the finer the grid and the more cells it contains. Normally, it is set to 8.
- Grid Y: The quantity of cells arranged vertically. Higher the dimensionality the generated feature vector has, the finer the grid and the more cells it contains. Normally, it is set to 8.

2. Algorithm Training: The algorithm must first be trained. In order to accomplish this, we

must utilize a dataset containing the faces of the individuals we wish to identify. To enable the algorithm to identify an input image and provide you with an output, we also need to specify an ID (which may be a number or the person's name) for each image. The same ID must appear in all images of the same person. Now that the training set has been created, let's examine the LBPH computational procedures.

3. Using the LBP Operation: The initial computational stage of the LBPH is to produce an intermediate image which, more accurately describes the original image, by emphasizing the face features. The algorithm does this by using the idea of a sliding window, which is dependent on the neighbors and radius parameters. This process is depicted in the picture below:

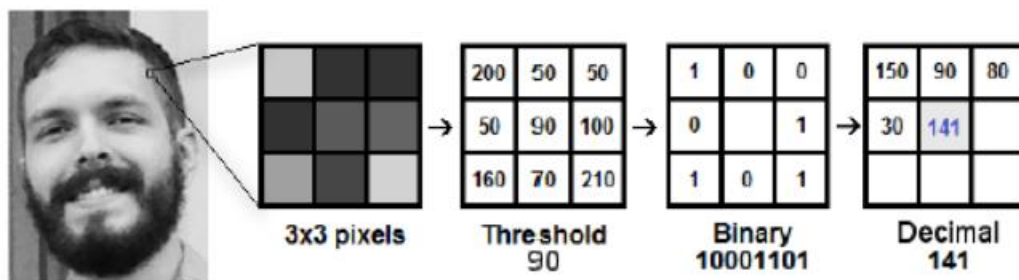


Fig. : LBP Operation

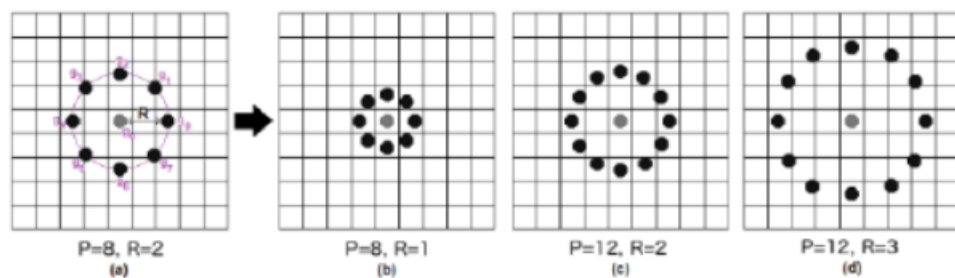


Fig. : The LBP Operation – Radius Change

4. Extracting the Histograms: The image created in the previous stage can now be divided into various grids using the Grid X and Grid Y parameters, as seen in the image below:

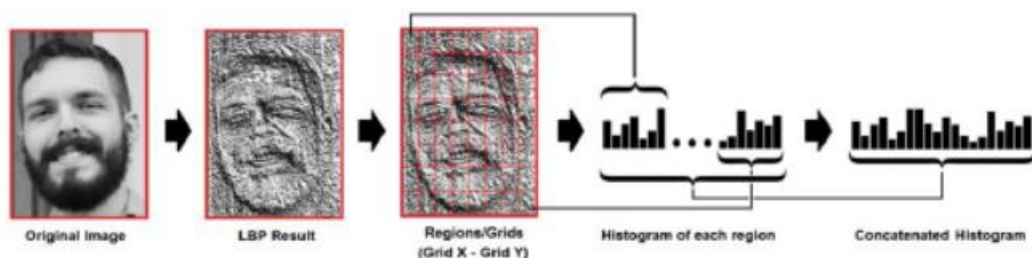


Fig. : Extracting the Histogram

The following is how we may extract each region's histogram from the above image:

- Because the image is grayscale, there will only be 256 places (0~255) in each histogram (from each grid) that correspond to the occurrences of each pixel intensity.
- To make a new, larger histogram, we must then concatenate each individual histogram. Assuming 8x8 grids, the final histogram will have $8 \times 8 \times 256 = 16,384$ places. The

resulting histogram displays the original image's features.

5. Identifying Faces: The algorithm has already been trained for this stage. Every image from the training dataset is represented by one of the produced histograms. In order to generate a histogram that accurately depicts an input image, we repeat the previous stages for the new image.

- We only need to compare two histograms in order to identify the image that matches the input image, and then return the image with the closest histogram.
- The distance between two histograms can be calculated using a variety of methods, such as the Euclidean distance, chi-square, absolute value, etc. Using the following formula, we may use the widely recognized Euclidean distance in this instance:

$$D = \sqrt{\sum_{i=1}^n (hist1_i - hist2_i)^2}$$

The ID from the picture with the closest histogram is therefore the algorithm's output. The computed distance, which may be utilized as a "confidence" metric, should also be returned by the algorithm.

- Then, we can automatically determine whether the algorithm has correctly identified the image by using a threshold and the "confidence" value. If the confidence is less than the established threshold, we can presume that the algorithm has identified the data correctly.

CODE

- main.py

```
### Imports and Setup

1. **Importing Required Libraries**:
```python
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
from student import Student
import os
from train import Train
from face_recognition import Face_Recognition
from attendance import Attendance
```

- **tkinter**: Provides tools for creating the GUI.
- **PIL**: For handling and displaying images.
- **student**: Custom module for student details.
- **os**: For interacting with the operating system.
- **train**: Custom module for training data.
- **face_recognition**: Custom module for face recognition.
- **attendance**: Custom module for attendance management.

### Class Definition: Face_Recognition_System

2. **Class Initialization**:
```python
class Face_Recognition_System:
 def __init__(self, root):
 self.root = root
 self.root.geometry("1530x790+0+0")
 self.root.title("Face Recognition System")
```

- Initializes the main window with a specified size and title.

3. **Image Setup**:
```python
first img
img=Image.open(r"college_images\img2.jpg")
img=img.resize((500,130),Image.ANTIALIAS)
self.photoimg=ImageTk.PhotoImage(img)

f_lbl = Label(self.root, image=self.photoimg)
f_lbl.place(x=0, y=0, width=500, height=130)

second img
img1=Image.open(r"college_images\img1.jpg")

```

```

img1=img1.resize((500,130),Image.ANTIALIAS)
self.photoimg1=ImageTk.PhotoImage(img1)

f_lbl1 = Label(self.root, image=self.photoimg1)
f_lbl1.place(x=500, y=0, width=500, height=130)

third img
img2=Image.open(r"college_images\img3.jpg")
img2=img2.resize((500,130),Image.ANTIALIAS)
self.photoimg2=ImageTk.PhotoImage(img2)

f_lbl1 = Label(self.root, image=self.photoimg2)
f_lbl1.place(x=1000, y=0, width=550, height=130)

background img
img3=Image.open(r"college_images\img4.jpg")
img3=img3.resize((1530,710),Image.ANTIALIAS)
self.photoimg3=ImageTk.PhotoImage(img3)

bg_img = Label(self.root, image=self.photoimg3)
bg_img.place(x=0, y=130, width=1530, height=710)
```


- Loads, resizes, and displays images at specific locations in the window.



4. **Title Label and Main Frame**:



```

```python
title_lbl=Label(bg_img, text="School Security System Using Facial
Recognition", font=("times new roman",35,"bold"),bg="white", fg="red")
title_lbl.place(x=0, y=0,width=1530, height=56)
```

```



- Creates a title label for the application.



### Buttons and Functionality



5. **Student Details Button**:



```

```python
b1 = Button(bg_img, image=self.photoimg4, command=self.student_details,
cursor="hand2")
b1.place(x=400, y=100, width=220, height=220)

b1_1 = Button(bg_img, text="Student Details",
command=self.student_details, cursor="hand2", font=("times new
roman",15,"bold"), bg="darkblue", fg="white")
b1_1.place(x=400, y=300, width=220, height=40)
```

```



- Creates a button and associated text label for displaying student details.


```

```

6. **Face Detector Button**:
    ```python
 b1 = Button(bg_img, image=self.photoimg5, cursor="hand2",
command=self.face_data)
 b1.place(x=700, y=100, width=220, height=220)

 b1_1 = Button(bg_img, text="Face Detector", cursor="hand2",
command=self.face_data, font=("times new roman",15,"bold"), bg="darkblue",
fg="white")
 b1_1.place(x=700, y=300, width=220, height=40)
    ```

    - Creates a button and associated text label for launching the face
    detector.

7. **Attendance Button**:
    ```python
 b1 = Button(bg_img, image=self.photoimg6, cursor="hand2",
command=self.attendance_data)
 b1.place(x=1000, y=100, width=220, height=220)

 b1_1 = Button(bg_img, text="Attendance", cursor="hand2",
command=self.attendance_data, font=("times new roman",15,"bold"),
bg="darkblue", fg="white")
 b1_1.place(x=1000, y=300, width=220, height=40)
    ```

    - Creates a button and associated text label for managing attendance.

8. **Train Data Button**:
    ```python
 b1 = Button(bg_img, image=self.photoimg8, cursor="hand2",
command=self.train_data)
 b1.place(x=550, y=380, width=220, height=220)

 b1_1 = Button(bg_img, text="Train Data", cursor="hand2",
command=self.train_data, font=("times new roman",15,"bold"),
bg="darkblue", fg="white")
 b1_1.place(x=550, y=580, width=220, height=40)
    ```

    - Creates a button and associated text label for training face
    recognition data.

9. **Photos Button**:
    ```python
 b1 = Button(bg_img, image=self.photoimg9, cursor="hand2",
command=self.open_img)
 b1.place(x=850, y=380, width=220, height=220)

```

```

b1_1 = Button(bg_img, text="Photos", cursor="hand2",
command=self.open_img, font=("times new roman",15,"bold"), bg="darkblue",
fg="white")
b1_1.place(x=850, y=580, width=220, height=40)
'''

```

- Creates a button and associated text label for opening photos.

### ### Function Definitions

#### 10. **\*\*Function Definitions\*\*:**

```

'''python
def open_img(self):
 os.startfile("data")

def student_details(self):
 self.new_window=Toplevel(self.root)
 self.app=Student(self.new_window)

def train_data(self):
 self.new_window=Toplevel(self.root)
 self.app=Train(self.new_window)

def face_data(self):
 self.new_window=Toplevel(self.root)
 self.app=Face_Recognition(self.new_window)

def attendance_data(self):
 self.new_window=Toplevel(self.root)
 self.app=Attendance(self.new_window)
'''

```

- Functions for opening images, displaying student details, training data, face recognition, and managing attendance.

### ### Main Function

#### 11. **\*\*Main Function\*\*:**

```

'''python
if __name__== "__main__":
 root = Tk()
 obj = Face_Recognition_System(root)
 root.mainloop()
'''

```

- Initializes the Tkinter window and starts the main loop.

This code sets up a GUI for a face recognition system with buttons to manage student details, attendance, face detection, training data, and viewing photos. Each button triggers a corresponding function that opens a new window or performs an action related to that feature.

- attendance.py

### ### Imports and Setup

#### 1. **\*\*Importing Required Libraries\*\***:

```
```python
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
from tkinter import messagebox
import mysql.connector
import cv2
import os
import csv
from tkinter import filedialog
```

- **tkinter**: Provides tools for creating the GUI.
- **PIL**: For handling and displaying images.
- **messagebox**: For displaying message boxes.
- **mysql.connector**: For connecting to MySQL databases.
- **cv2**: OpenCV library for computer vision tasks (not used in the provided code).
- **os**: For interacting with the operating system.
- **csv**: For handling CSV files.
- **filedialog**: For file dialogs (open/save).
```

#### 2. **\*\*Global Data Variable\*\***:

```
```python
mydata=[]
```

- `mydata`: A list to store data read from CSV files.
```

### ### Class Definition: Attendance

#### 3. **\*\*Class Initialization\*\***:

```
```python
class Attendance:
    def __init__(self, root):
        self.root = root
        self.root.geometry("1530x790+0+0")
        self.root.title("Face Recognition System")
```

- Initializes the main window with a specified size and title.
```

#### 4. **\*\*Variable Initialization\*\***:

```
```python
self.var_atten_id=StringVar()
```

```

self.var_aten_roll=StringVar()
self.var_aten_name=StringVar()
self.var_aten_dep=StringVar()
self.var_aten_time=StringVar()
self.var_aten_date=StringVar()
self.var_aten_attendance=StringVar()
```


- Creates variables for storing form data.


```

#### 5. **\*\*Image Setup\*\*:**

```

```python
# First image
img=Image.open(r"college_images\img1.jpg")
img=img.resize((800,200),Image.ANTIALIAS)
self.photoimg=ImageTk.PhotoImage(img)

f_lbl=Label(self.root,image=self.photoimg)
f_lbl.place(x=0,y=0,width=800,height=200)

# Second image
img1=Image.open(r"college_images\img2.jpg")
img1=img1.resize((800,200),Image.ANTIALIAS)
self.photoimg1=ImageTk.PhotoImage(img1)

f_lbl=Label(self.root,image=self.photoimg1)
f_lbl.place(x=800,y=0,width=800,height=200)

# Background image
img3=Image.open(r"college_images\img1.jpg")
img3=img3.resize((1530,710),Image.ANTIALIAS)
self.photoimg3=ImageTk.PhotoImage(img3)

bg_img = Label(self.root, image=self.photoimg3)
bg_img.place(x=0, y=200, width=1530, height=710)
```

```

- Loads, resizes, and displays images at specific locations in the window.

#### 6. **\*\*Title Label and Main Frame\*\*:**

```

```python
title_lbl=Label(bg_img, text="ATTENDANCE MANAGEMENT SYSTEM",
font=("times new roman",35,"bold"),bg="white", fg="red")
title_lbl.place(x=0, y=0,width=1530, height=45)

main_frame=Frame(bg_img,bd=2,bg="white")
main_frame.place(x=20,y=55,width=1480,height=600)
```

```

- Creates a title label and main frame for organizing the layout.

### ### Left Frame: Student Attendance Details

#### 7. **\*\*Left Frame Setup\*\***:

```
```python
Left_frame=LabelFrame(main_frame,bd=2,bg="white",relief=RIDGE,text="Student Attendance Details",font=("times new roman",12,"bold"))
Left_frame.place(x=10,y=10,width=730,height=580)

img_left=Image.open(r"college_images\img3.jpg")
img_left=img_left.resize((720,130),Image.ANTIALIAS)
self.photoimg_left=ImageTk.PhotoImage(img_left)

f_lbl=Label(Left_frame,image=self.photoimg_left)
f_lbl.place(x=5,y=0,width=720,height=130)
```

- Sets up a labeled frame on the left for student attendance details and adds an image.
```

#### 8. **\*\*Input Fields and Labels\*\***:

```
```python
left_inside_frame=Frame(Left_frame,bd=2,relief=RIDGE,bg="white")
left_inside_frame.place(x=0,y=180,width=720,height=370)

# Label and Entry
# Attendance ID
attendanceId_lael=Label(left_inside_frame,text="AttendanceId:",font=("times new roman",13,"bold"),bg="white")
attendanceId_lael.grid(row=0,column=0,padx=10,pady=5,sticky=W)

attendanceId_entry=ttk.Entry(left_inside_frame,width=20,textvariable=self.var_atten_id,font=("times new roman",13,"bold"))
attendanceId_entry.grid(row=0,column=1,padx=10,pady=5,sticky=W)

# Roll No
rollLabel=Label(left_inside_frame,text="Roll:",bg="white",font="comicsansns 11 bold")
rollLabel.grid(row=0,column=2,padx=4,pady=8)

atten_roll=ttk.Entry(left_inside_frame,width=22,textvariable=self.var_atten_roll,font="comicsansns 11 bold")
atten_roll.grid(row=0,column=3,pady=8)

# Name
nameLabel=Label(left_inside_frame,text="Name:",bg="white",font="comicsansns 11 bold")
nameLabel.grid(row=1,column=0)
```



```

        atten_name=ttk.Entry(left_inside_frame,width=22,textvariable=self.var_
        atten_name,font=("comicsansns 11 bold"))
        atten_name.grid(row=1,column=1,pady=8)

        # Department
        deptLabel=Label(left_inside_frame,text="Department:",bg="white",font="
comicsansns 11 bold")
        deptLabel.grid(row=1,column=2)

        atten_dept=ttk.Entry(left_inside_frame,width=22,
        textvariable=self.var_atten_dep,font=("comicsansns 11 bold"))
        atten_dept.grid(row=1,column=3,pady=8)

        # Time
        timeLabel=Label(left_inside_frame,text="Time:",bg="white",font="comics
ansns 11 bold")
        timeLabel.grid(row=2,column=0)

        atten_time=ttk.Entry(left_inside_frame,width=22,textvariable=self.var_
        atten_time,font=("comicsansns 11 bold"))
        atten_time.grid(row=2,column=1,pady=8)

        # Date
        dateLabel=Label(left_inside_frame,text="Date:",bg="white",font="comics
ansns 11 bold")
        dateLabel.grid(row=2,column=2)

        atten_date=ttk.Entry(left_inside_frame,width=22,
        textvariable=self.var_atten_date, font=("comicsansns 11 bold"))
        atten_date.grid(row=2,column=3,pady=8)

        # Attendance
        attendanceLabel=Label(left_inside_frame,text="Attendance
Status:",bg="white",font="comicsansns 11 bold")
        attendanceLabel.grid(row=3,column=0)

        self.atten_status=ttk.Combobox(left_inside_frame,width=20,textvariable
        =self.var_atten_attendance,font=("comicsansns 11 bold"),state="readonly")
        self.atten_status["values"]=("Status","Present","Absent")
        self.atten_status.grid(row=3,column=1,pady=8)
        self.atten_status.current(0)
        ``

        - Sets up labels and entry fields for capturing attendance details.

9. **Buttons for CSV Operations**:
    ``python
    btn_frame=Frame(left_inside_frame,bd=2,relief=RIDGE,bg="white")
    btn_frame.place(x=0,y=270,width=715,height=35)

```

```

        save_btn=Button(btn_frame,text="Import
csv",command=self.importCsv,width=17,font=("times new
roman",13,"bold"),bg="blue",fg="white")
        save_btn.grid(row=0,column=0)

        update_btn=Button(btn_frame,text="Export
csv",command=self.exportCsv,width=17,font=("times new
roman",13,"bold"),bg="blue",fg="white")
        update_btn.grid(row=0,column=1)

        delete_btn=Button(btn_frame,text="Update",width=17,font=("times new
roman",13,"bold"),bg="blue",fg="white")
        delete_btn.grid(row=0,column=2)

        reset_btn=Button(btn_frame,text="Reset",command=self.reset_data,width=
17,font=("times new roman",13,"bold"),bg="blue",fg="white")
        reset_btn.grid(row=0,column=3)
        ```

 - Creates buttons for importing, exporting, updating, and resetting
data.

Right Frame: Attendance Details

10. **Right Frame Setup**:
    ```python
    Right_frame=LabelFrame(main_frame,bd=2,bg="white",relief=RIDGE,text="A
ttendance Details",font=("times new roman",12,"bold"))
    Right_frame.place(x=750

,y=10,width=720,height=580)

    table_frame=LabelFrame(Right_frame,bd=2,bg="white",relief=RIDGE)
    table_frame.place(x=5,y=5,width=700,height=445)
    ```

 - Sets up a labeled frame on the right for displaying attendance
details.

11. **Attendance Table**:
    ```python
    # Scroll bars
    scroll_x=ttk.Scrollbar(table_frame,orient=HORIZONTAL)
    scroll_y=ttk.Scrollbar(table_frame,orient=VERTICAL)

    self.AttendanceReportTable=ttk.Treeview(table_frame,column=("id","roll
","name","department","time","date","attendance"),xscrollcommand=scroll_x.
set,yscrollcommand=scroll_y.set)

```

```

scroll_x.pack(side=BOTTOM,fill=X)
scroll_y.pack(side=RIGHT,fill=Y)

scroll_x.config(command=self.AttendanceReportTable.xview)
scroll_y.config(command=self.AttendanceReportTable.yview)

self.AttendanceReportTable.heading("id",text="Attendance Id")
self.AttendanceReportTable.heading("roll",text="Roll")
self.AttendanceReportTable.heading("name",text="Name")
self.AttendanceReportTable.heading("department",text="Department")
self.AttendanceReportTable.heading("time",text="Time")
self.AttendanceReportTable.heading("date",text="Date")
self.AttendanceReportTable.heading("attendance",text="Attendance")

self.AttendanceReportTable["show"]="headings"
self.AttendanceReportTable.column("id",width=100)
self.AttendanceReportTable.column("roll",width=100)
self.AttendanceReportTable.column("name",width=100)
self.AttendanceReportTable.column("department",width=100)
self.AttendanceReportTable.column("time",width=100)
self.AttendanceReportTable.column("date",width=100)
self.AttendanceReportTable.column("attendance",width=100)

self.AttendanceReportTable.pack(fill=BOTH,expand=1)

self.AttendanceReportTable.bind("<ButtonRelease>",self.get_cursor)
'''
- Sets up a table with scroll bars to display attendance data.

### Data Handling Functions

12. **Fetch Data**:
    ```python
 def fetchData(self,rows):
 self.AttendanceReportTable.delete(*self.AttendanceReportTable.get_
children())
 for i in rows:
 self.AttendanceReportTable.insert("",END,values=i)
 '''
 - Fetches and displays data in the table.

13. **Import CSV**:
    ```python
    def importCsv(self):
        global mydata
        mydata.clear()
        fln=filedialog.askopenfilename(initialdir=os.getcwd(),title="Open
CSV",filetypes=(("CSV File","*.csv"),("ALL File","*.*")),parent=self.root)
    ```

```

```

 with open(fln) as myfile:
 csvread=csv.reader(myfile,delimiter=",")
 for i in csvread:
 mydata.append(i)
 self.fetchData(mydata)
 """
 - Imports data from a CSV file and populates the table.

14. **Export CSV**:
 """python
 def exportCsv(self):
 try:
 if len(mydata)<1:
 messagebox.showerror("No Data","No Data Found to
Export",parent=self.root)
 return False
 fln=filedialog.asksaveasfilename(initialdir=os.getcwd(),title=
"Open CSV",filetypes=(("CSV File","*.csv"),("ALL
File","*.*")),parent=self.root)
 with open(fln,mode="w",newline="") as myfile:
 exp_write=csv.writer(myfile,delimiter=",")
 for i in mydata:
 exp_write.writerow(i)
 messagebox.showinfo("Data Export","Your Data Exported to
"+os.path.basename(fln)+" successfully")
 except Exception as es:
 messagebox.showerror("Error",f"Due To :
{str(es)}",parent=self.root)
 """
 - Exports table data to a CSV file.

15. **Get Cursor**:
 """python
 def get_cursor(self,event=""):
 cursor_row=self.AttendanceReportTable.focus()
 content=self.AttendanceReportTable.item(cursor_row)
 rows=content["values"]
 self.var_atten_id.set(rows[0])
 self.var_atten_roll.set(rows[1])
 self.var_atten_name.set(rows[2])
 self.var_atten_dep.set(rows[3])
 self.var_atten_time.set(rows[4])
 self.var_atten_date.set(rows[5])
 self.var_atten_attendance.set(rows[6])
 """
 - Gets data from the selected row in the table and sets it in the
corresponding entry fields.

```

```

16. **Reset Data**:
    ```python
    def reset_data(self):
        self.var_attn_id.set("")
        self.var_attn_roll.set("")
        self.var_attn_name.set("")
        self.var_attn_dep.set("")
        self.var_attn_time.set("")
        self.var_attn_date.set("")
        self.var_attn_attendance.set("")
    ```

 - Clears all entry fields.

Main Function

17. **Main Function**:
    ```python
    if __name__=="__main__":
        root=Tk()
        obj=Attendance(root)
        root.mainloop()
    ```

 - Initializes the Tkinter window and starts the main loop.

```

This code sets up a GUI for an attendance management system with features to import/export CSV files, and display and manage attendance records.

- student.py

```

Imports and Libraries

```python
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
from tkinter import messagebox
import mysql.connector
import cv2
```

- **tkinter**: Used for creating the GUI application.
- **PIL (Pillow)**: Python Imaging Library, used for image processing tasks.
- **mysql.connector**: Connector for MySQL database.
- **cv2 (OpenCV)**: Open Source Computer Vision Library, used for computer vision tasks like face detection and image processing.

Student Class Initialization

```

```

```python
class Student:
    def __init__(self, root):
        self.root = root
        self.root.geometry("1530x790+0+0")
        self.root.title("Face Recognition System")
```

- **Student Class**: A class to encapsulate the entire GUI application.
- **self.root**: Represents the main Tkinter window.
- **geometry()**: Sets the initial size and position of the main window.
- **title()**: Sets the title of the main window.

```

### ### Instance Variables Initialization

```

```python
self.var_dep = StringVar()
self.var_course = StringVar()
self.var_year = StringVar()
self.var_semester = StringVar()
self.var_std_id = StringVar()
self.var_std_name = StringVar()
self.var_div = StringVar()
self.var_roll = StringVar()
self.var_gender = StringVar()
self.var_dob = StringVar()
self.var_email = StringVar()
self.var_phone = StringVar()
self.var_address = StringVar()
self.var_teacher = StringVar()
```

- **StringVar**: Tkinter variable type to hold string data.
- These variables are used to store and manipulate data from GUI components, such as entry fields, labels, and comboboxes.

```

### ### Setting Up Images

```

```python
# First Image
img =
Image.open(r"C:\Users\Meghna\OneDrive\Desktop\Attendance\college_images\img2.jpg")
img = img.resize((500, 130), Image.ANTIALIAS)
self.photoimg = ImageTk.PhotoImage(img)

f_lbl = Label(self.root, image=self.photoimg)

```

```

f_lbl.place(x=0, y=0, width=500, height=130)

# Second Image
img1 =
Image.open(r"C:\Users\Meghna\OneDrive\Desktop\Attendance\college_images\img1.jpg")
img1 = img1.resize((500, 130), Image.ANTIALIAS)
self.photoimg1 = ImageTk.PhotoImage(img1)

f_lbl = Label(self.root, image=self.photoimg1)
f_lbl.place(x=500, y=0, width=500, height=130)

# Third Image
img2 =
Image.open(r"C:\Users\Meghna\OneDrive\Desktop\Attendance\college_images\img3.jpg")
img2 = img2.resize((500, 130), Image.ANTIALIAS)
self.photoimg2 = ImageTk.PhotoImage(img2)

f_lbl = Label(self.root, image=self.photoimg2)
f_lbl.place(x=1000, y=0, width=550, height=130)

# Background Image
img3 =
Image.open(r"C:\Users\Meghna\OneDrive\Desktop\Attendance\college_images\img4.jpg")
img3 = img3.resize((1530, 710), Image.ANTIALIAS)
self.photoimg3 = ImageTk.PhotoImage(img3)

bg_img = Label(self.root, image=self.photoimg3)
bg_img.place(x=0, y=130, width=1530, height=710)
```


- Image and ImageTk: Used from PIL to work with images and display them in Tkinter.

- resize(): Resizes the images to fit into the GUI window dimensions.

- Label: Tkinter widget used to display images on the GUI.

Title Label


```

```python
title_lbl = Label(bg_img, text="Student Management System", font=("times new roman", 35, "bold"), bg="white", fg="black")
title_lbl.place(x=0, y=0, width=1530, height=45)
```

```


- Label Widget: Displays text in the GUI.

- bg: Background color.


```

```

- **fg**: Text color.
- **place()**: Positions the label within the GUI window.

Main Frame Setup

```python
main_frame = Frame(bg_img, bd=2, bg="white")
main_frame.place(x=5, y=55, width=1500, height=600)
```

- **Frame Widget**: Used to group and organize other widgets.
- **bd**: Border width.
- **bg**: Background color.
- **place()**: Positions the frame within the GUI window.

Left Frame Setup

```python
Left_frame = LabelFrame(main_frame, bd=2, bg="white", relief=RIDGE,
text="Student Details", font=("times new roman", 12, "bold"))
Left_frame.place(x=10, y=10, width=760, height=580)
```

- **LabelFrame Widget**: A frame with a label and a border.
- **relief**: Border style.
- **text**: Label text.
- **place()**: Positions the frame within the main frame.

Current Course Frame

```python
current_course_frame = LabelFrame(Left_frame, bd=2, bg="white",
relief=RIDGE, text="Current Course Information", font=("times new roman",
12, "bold"))
current_course_frame.place(x=5, y=135, width=720, height=150)
```

- **LabelFrame Widget**: Another frame within the Left_frame to display
course information.

Comboboxes and Labels

- Numerous comboboxes and labels are created throughout the application to
input and display student details such as department, course, year,
semester, student ID, name, division, roll number, gender, date of birth,
email, phone number, address, and teacher information.

Class Student Information Frame

```



```

```python
class_student_frame = LabelFrame(Left_frame, bd=2, bg="white",
relief=RIDGE, text="Class Student Information", font=("times new roman",
12, "bold"))
class_student_frame.place(x=5, y=250, width=720, height=300)
```

- LabelFrame Widget: A frame within Left_frame to display student
information such as student ID, name, division, and roll number.

Buttons and Functionality

```python
save_btn = Button(btn_frame, text="Save", command=self.add_data, width=17,
font=("times new roman", 13, "bold"), bg="blue", fg="white")
save_btn.grid(row=0, column=0)
```

- Button Widget: Used to create buttons.
- command: Function to be called when the button is clicked.
- grid(): Organizes widgets in a table-like structure within their
parent widget.

Right Frame

```python
Right_frame = LabelFrame(main_frame, bd=2, bg="white", relief=RIDGE,
text="Student Details", font=("times new roman", 12, "bold"))
Right_frame.place(x=780, y=10, width=700, height=580)
```

- LabelFrame Widget: A frame within the main_frame to display and
search for student details.

Search System and Table Frame

- A search system and table frame with a scrollbar are created to display
fetched data from the MySQL database.

Function Definitions

- Functions such as `add_data`, `fetch_data`, `get_cursor`, `update_data`,
`delete_data`, `reset_data`, and `generate_dataset` are defined to perform
operations like adding, updating, deleting student data, resetting the
interface, and capturing face samples.

Face Cropping and Database Operations

```

- Code for capturing face samples using OpenCV and storing them in the MySQL database.

### ### Summary

This application integrates Tkinter for the GUI, MySQL for database operations, and OpenCV for image processing. It manages student details through a user-friendly interface with the ability to add, update, delete, and search for student records, as well as capture and store student face images for recognition purposes. The GUI layout is structured using frames, labels, buttons, and input fields to provide a comprehensive student management system.

- train.py

### ### Imports and Libraries

```
```python
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
from tkinter import messagebox
import mysql.connector
import cv2
import os
import numpy as np
```

- tkinter: Used for creating the GUI application.
- PIL (Pillow): Python Imaging Library, used for image processing tasks.
- mysql.connector: Connector for MySQL database.
- cv2 (OpenCV): Open Source Computer Vision Library, used for computer vision tasks like face detection and image processing.
- os: Provides functions to interact with the operating system.
- numpy: Library for numerical computations in Python.
```

### ### Class Initialization

```
```python
class Train:
    def __init__(self,root):
        self.root = root
        self.root.geometry("1530x790+0+0")
        self.root.title("face Recognition System")
```
```

```

- **Train Class**: A class to encapsulate the entire GUI application for
training the face recognition system.
- **self.root**: Represents the main Tkinter window.
- **geometry()**: Sets the initial size and position of the main window.
- **title()**: Sets the title of the main window.

Title Label

```python
title_lbl=Label(self.root,text="TRAIN DATA SET",font=("times new
roman",35,"bold"),bg="white",fg="red")
title_lbl.place(x=0,y=0,width=1530,height=45)
```

- **Label Widget**: Displays text in the GUI.
- **bg**: Background color.
- **fg**: Text color.
- **place()**: Positions the label within the GUI window.

Top Image

```python
img_top=Image.open(r"college_images\img4.jpg")
img_top=img_top.resize((1530,325),Image.ANTIALIAS)
self.photoimg_top=ImageTk.PhotoImage(img_top)

f_lbl=Label(self.root,image=self.photoimg_top)
f_lbl.place(x=0,y=55,width=1530,height=325)
```

- **Image and ImageTk**: Used from PIL to work with images and display
them in Tkinter.
- **resize()**: Resizes the images to fit into the GUI window dimensions.
- **Label**: Tkinter widget used to display images on the GUI.

Button

```python
b1_1=Button(self.root,text="TRAIN
DATA",command=self.train_classifier,cursor="hand2",font=("times new
roman",30,"bold"),bg="red",fg="white")
b1_1.place(x=0,y=380,width=1530,height=60)
```

- **Button Widget**: Used to create a button.
- **command**: Function to be called when the button is clicked.
- **cursor**: Cursor style.
- **font**: Font settings.

```

```

- bg: Background color.
- fg: Text color.
- place(): Positions the button within the GUI window.

Bottom Image

```python
img_bottom=Image.open(r"college_images\train1.jpg")
img_bottom=img_bottom.resize((1530,1325),Image.ANTIALIAS)
self.photoimg_bottom=ImageTk.PhotoImage(img_bottom)

f_lbl=Label(self.root,image=self.photoimg_bottom)
f_lbl.place(x=0,y=440,width=1530,height=325)
```

- Image and ImageTk: Used from PIL to work with images and display them in Tkinter.
- resize(): Resizes the images to fit into the GUI window dimensions.
- Label: Tkinter widget used to display images on the GUI.

Training Method

```python
def train_classifier(self):
    data_dir="data" # directory name
    path=[os.path.join(data_dir,file) for file in os.listdir(data_dir)]

    faces=[]
    ids=[]

    for image in path:
        img=Image.open(image).convert('L') # gray scale image
        imageNp=np.array(img,'uint8')
        id=int(os.path.split(image)[1].split('.')[1])

        faces.append(imageNp)
        ids.append(id)
        cv2.imshow("Training",imageNp)
        cv2.waitKey(1)==13
    ids=np.array(ids)

    # Train the classifier and save
    clf=cv2.face.LBPHFaceRecognizer_create()
    clf.train(faces,ids)
    clf.write("classifier.xml")
    cv2.destroyAllWindows()
    messagebox.showinfo("Result","Training datasets completed!!")
```

```

```

- **train_classifier(**: Method to train the face recognition classifier.
- **os.path.join(**: Joins paths of the directory and files.
- **os.listdir(**: Lists files in a directory.
- **Image.open().convert(**: Opens an image and converts it to grayscale.
- **np.array(**: Converts the image to a numpy array.
- **cv2.imshow(**: Displays an image in a window.
- **cv2.waitKey(**: Waits for a pressed key.
- **cv2.face.LBPHFaceRecognizer_create(**: Creates a LBPH (Local Binary
Patterns Histograms) face recognizer.
- **clf.train(**: Trains the face recognizer with faces and corresponding
ids.
- **clf.write(**: Saves the trained classifier to a file.
- **cv2.destroyAllWindows(**: Closes all OpenCV windows.
- **messagebox.showinfo(**: Displays an information message box.

```

### ### Main Function

```

```python
if __name__ == "__main__":
    root=Tk()
    obj=Train(root)
    root.mainloop()
```

- **Tk(**: Creates the main window.
- **Train(root)**: Creates an instance of the Train class.
- **mainloop(**: Starts the Tkinter event loop.

```

### ### Summary

This application uses Tkinter for the GUI, OpenCV for image processing and face recognition, and PIL (Pillow) for image manipulation. It provides a graphical interface to train a face recognition system using images from a specified directory. The GUI consists of labels, buttons, and images arranged using place() method to create an interactive user experience.

- face\_recognition.py

Sure! Let's break down the code step by step:

### ### Imports and Libraries

```

```python
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
from tkinter import messagebox
import mysql.connector

```

```

import cv2
import os
import numpy as np
from time import strftime
from datetime import datetime
```

- tkinter: Used for creating the GUI application.
- PIL (Pillow): Python Imaging Library, used for image processing tasks.
- mysql.connector: Connector for MySQL database.
- cv2 (OpenCV): Open Source Computer Vision Library, used for computer vision tasks like face detection and image processing.
- os: Provides functions to interact with the operating system.
- numpy: Library for numerical computations in Python.
- time: Module for working with times and dates.
- datetime: Module for working with dates and times.

Class Initialization

```python
class Face_Recognition:
    def __init__(self,root):
        self.root = root
        self.root.geometry("1530x790+0+0")
        self.root.title("face Recognition System")
```

- Face_Recognition Class: Represents the main application class for face recognition.
- self.root: Represents the main Tkinter window.
- geometry(): Sets the initial size and position of the main window.
- title(): Sets the title of the main window.

Title Label

```python
title_lbl=Label(self.root,text="FACE RECOGNITION",font=("times new roman",35,"bold"),bg="white",fg="green")
title_lbl.place(x=0,y=0,width=1530,height=45)
```

- Label Widget: Displays text in the GUI.
- bg: Background color.
- fg: Text color.
- place(): Positions the label within the GUI window.

First Image

```

```

```python
img_top=Image.open(r"college_images\face_rec.jpg")
img_top=img_top.resize((650,700),Image.ANTIALIAS)
self.photoimg_top=ImageTk.PhotoImage(img_top)

f_lbl=Label(self.root,image=self.photoimg_top)
f_lbl.place(x=0,y=55,width=650,height=700)
```

- Image and ImageTk: Used from PIL to work with images and display them in Tkinter.
- resize(): Resizes the images to fit into the GUI window dimensions.
- Label: Tkinter widget used to display images on the GUI.

Second Image

```python
img_bottom=Image.open(r"college_images\phone.jpg")
img_bottom=img_bottom.resize((950,700),Image.ANTIALIAS)
self.photoimg_bottom=ImageTk.PhotoImage(img_bottom)

f_lbl=Label(self.root,image=self.photoimg_bottom)
f_lbl.place(x=650,y=55,width=950,height=700)
```

- Image and ImageTk: Used from PIL to work with images and display them in Tkinter.
- resize(): Resizes the images to fit into the GUI window dimensions.
- Label: Tkinter widget used to display images on the GUI.

Button

```python
b1_1=Button(f_lbl,text="Face
Recognition",cursor="hand2",command=self.face_recog,font=("times new
roman",18,"bold"),bg="darkgreen",fg="white")
b1_1.place(x=365,y=620,width=200,height=40)
```

- Button Widget: Used to create a button.
- command: Function to be called when the button is clicked.
- cursor: Cursor style.
- font: Font settings.
- bg: Background color.
- fg: Text color.
- place(): Positions the button within the GUI window.

```

### ### Mark Attendance Method

```
```python
def mark_attendance(self,i,r,n,d):
    with open("attendance.csv","r+",newline="\n") as f:
        myDataList=f.readlines()
        name_list=[]
        for line in myDataList:
            entry=line.split(",")
            name_list.append(entry[0])
            if((i not in name_list) and (r not in name_list) and (n not in
name_list) and (d not in name_list)):
                now=datetime.now()
                d1=now.strftime("%d/%m/%Y")
                dtString=now.strftime("%H:%M:%S")
                f.writelines(f"\n{i},{r},{n},{d},{dtString},{d1},Present")
```
```

- **mark\_attendance()**: Method to mark attendance in a CSV file.
- **open()**: Opens a file.
- **readlines()**: Reads all lines from a file.
- **split()**: Splits a string into a list.
- **strftime()**: Converts a date and time to a string.
- **writelines()**: Writes a sequence of strings to a file.

### ### Face Recognition Method

```
```python
def face_recog(self):
    def
draw_boundary(img,classifier,scaleFactor,minNeighbors,color,text,clf):
    gray_image=cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
    features=classifier.detectMultiScale(gray_image,scaleFactor,minNeighbors)

    coord=[]

    for(x,y,w,h) in features:
        cv2.rectangle(img,(x,y),(x+w,y+h),(0,255,0),3)
        id,predict=clf.predict(gray_image[y:y+h,x:x+w])
        confidence=int((100*(1-predict/300)))

        conn=mysql.connector.connect(host="localhost",username="root",
password="Md#12345",database="face_recognizer")
        my_cursor=conn.cursor()

        my_cursor.execute("select Name from student where
Student_id="+str(id))
```



```

        n=my_cursor.fetchone()
        n= str(n[0]) if n else ""

        my_cursor.execute("select Roll from student where
Student_id="+str(id))
        r=my_cursor.fetchone()
        r=str(r[0]) if r else ""

        my_cursor.execute("select Dep from student where
Student_id="+str(id))
        d=my_cursor.fetchone()
        d=str(d[0]) if d else ""

        my_cursor.execute("select Student_id from student where
Student_id="+str(id))
        i=my_cursor.fetchone()
        i=str(i[0]) if i else ""

        if confidence>77:
            cv2.putText(img,f"ID:{i}",(x,y-
75),cv2.FONT_HERSHEY_COMPLEX,0.8,(255,255,255),3)
            cv2.putText(img,f"Roll:{r}",(x,y-
55),cv2.FONT_HERSHEY_COMPLEX,0.8,(255,255,255),3)
            cv2.putText(img,f"Name:{n}",(x,y-
30),cv2.FONT_HERSHEY_COMPLEX,0.8,(255,255,255),3)
            cv2.putText(img,f"Department:{d}",(x,y-
5),cv2.FONT_HERSHEY_COMPLEX,0.8,(255,255,255),3)
            self.mark_attendance(i,r,n,d)
        else:
            cv2.rectangle(img,(x,y),(x+w,y+h),(0,0,255),3)
            cv2.putText(img,"Unknown Face",(x,y-
5),cv2.FONT_HERSHEY_COMPLEX,0.8,(255,255,255),3)

        coord=[x,y,w,h]

        return coord

def recognize(img,clf,faceCascade):
    coord=draw_boundary(img,faceCascade,1.1,10,(255,255,255),"Face",cl
f)

    return img

faceCascade=cv2.CascadeClassifier("haarcascade_frontalface_default.xml
")
clf=cv2.face.LBPHFaceRecognizer_create()
clf.read("classifier.xml")

video_cap=cv2.VideoCapture(0)

```

```

while True:
    ret,img=video_cap.read()
    img=recognize(img,clf,faceCascade)
    cv2.imshow("Welcome To Face Recognition",img)

    if cv2.waitKey(1)== ord('q'):
        break
video_cap.release()
cv2.destroyAllWindows()
'''

- face_recog(): Method to perform face recognition using OpenCV.
- draw_boundary(): Draws a rectangle around detected faces and displays associated information.
- cv2.cvtColor(): Converts an image from one color space to another (in this case, from BGR to grayscale).
- classifier.detectMultiScale(): Detects objects in the input image.
- cv2.rectangle(): Draws a rectangle on the image.
- cv2.putText(): Writes text on the image.
- mysql.connector.connect(): Connects to a MySQL database.
- my_cursor.execute(): Executes a SQL query.
- cv2.face.LBPHFaceRecognizer_create(): Creates a LBPH (Local Binary Patterns Histograms) face recognizer.
- clf.predict(): Predicts the label and confidence of the input sample.
- if confidence>77: Checks if the confidence level is high enough to recognize the face.
- mark_attendance(): Marks the attendance of the recognized face in a CSV file.

### Main Function

'''python
if __name__ == "__main__":
    root=Tk()
    obj=Face_Recognition(root)
    root.mainloop()
'''

- Tk(): Creates the main window.
- Face_Recognition(root): Creates an instance of the Face_Recognition class.
- mainloop(): Starts the Tkinter event loop.

### Summary

This application uses Tkinter for the GUI, OpenCV for face detection and

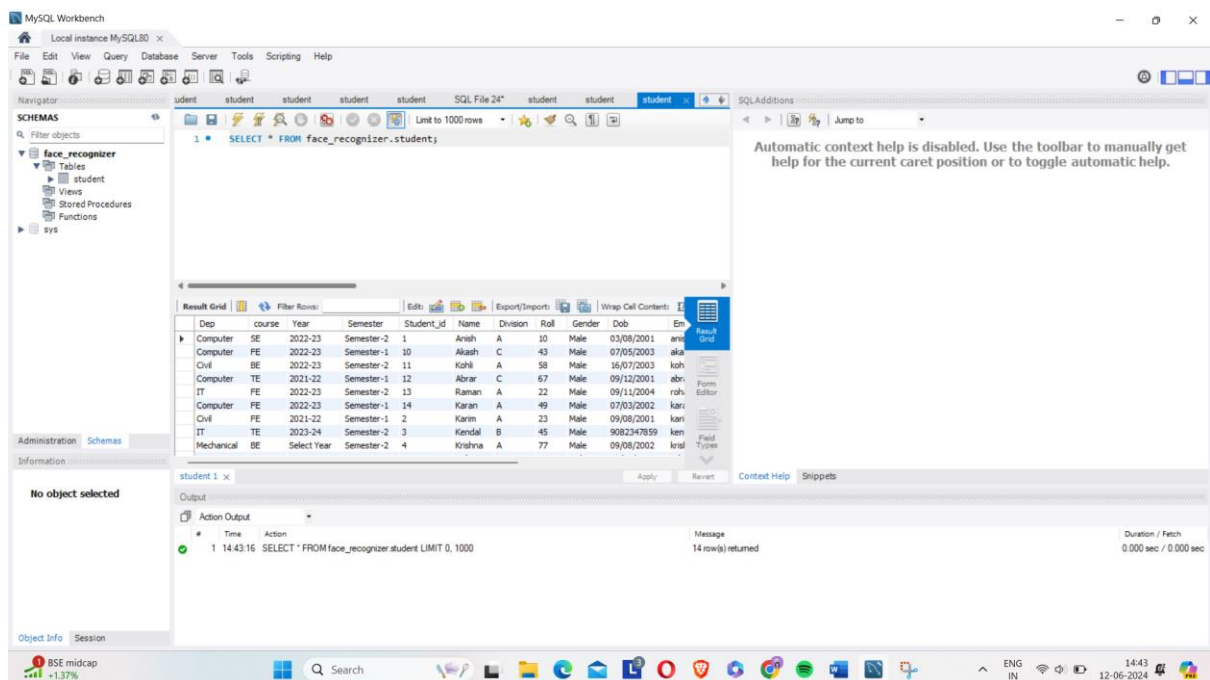
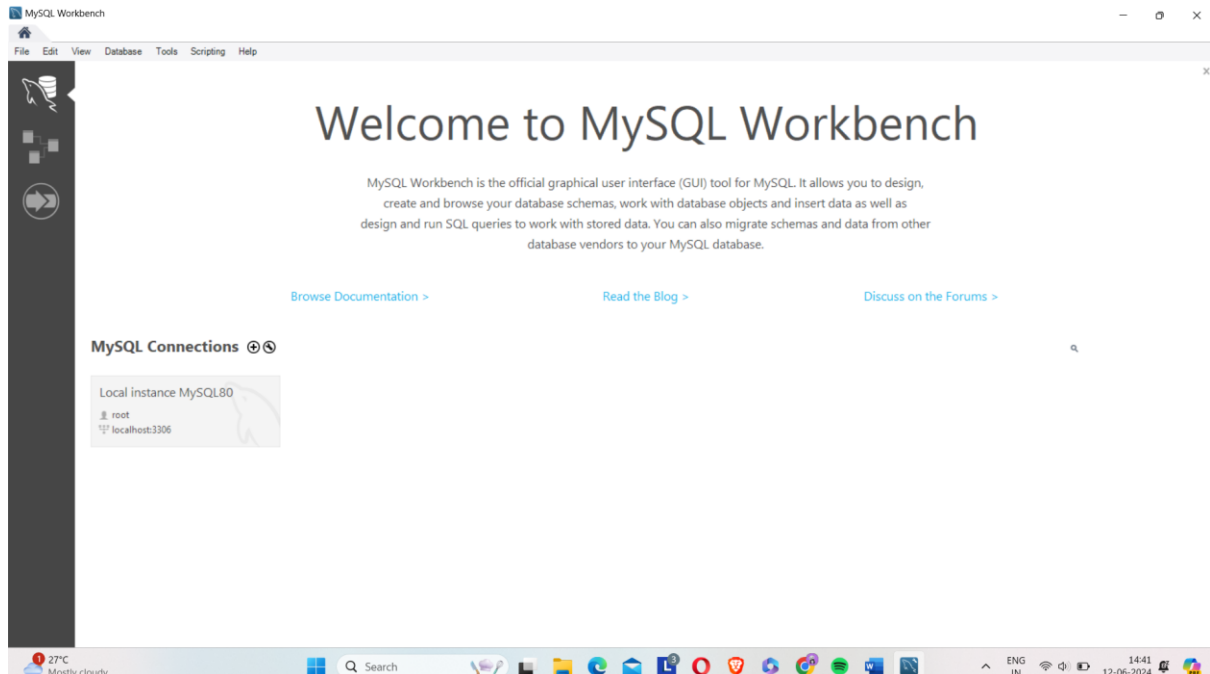
```

recognition, and MySQL for database operations. It provides a graphical interface to perform face recognition in real-time using a webcam. The GUI consists of labels, buttons, and images arranged using the `place()` method to create an interactive user experience. Detected faces

WEB INTERFACE

Back-end Server

MySQL Workbench has been used for the Back-end Server.

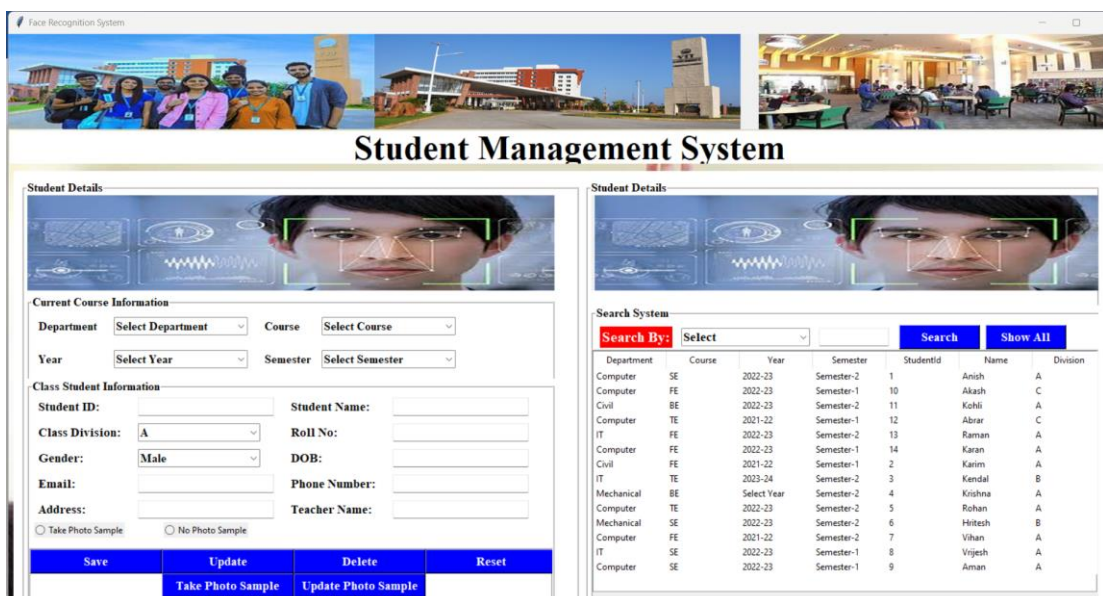


Webapp Preview

1. Landing Page



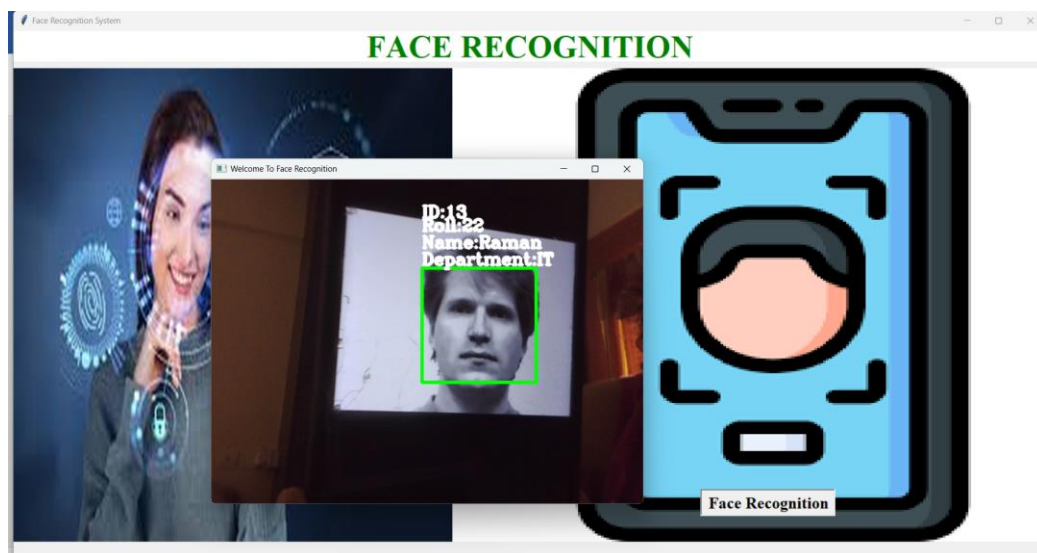
2. Student Management System



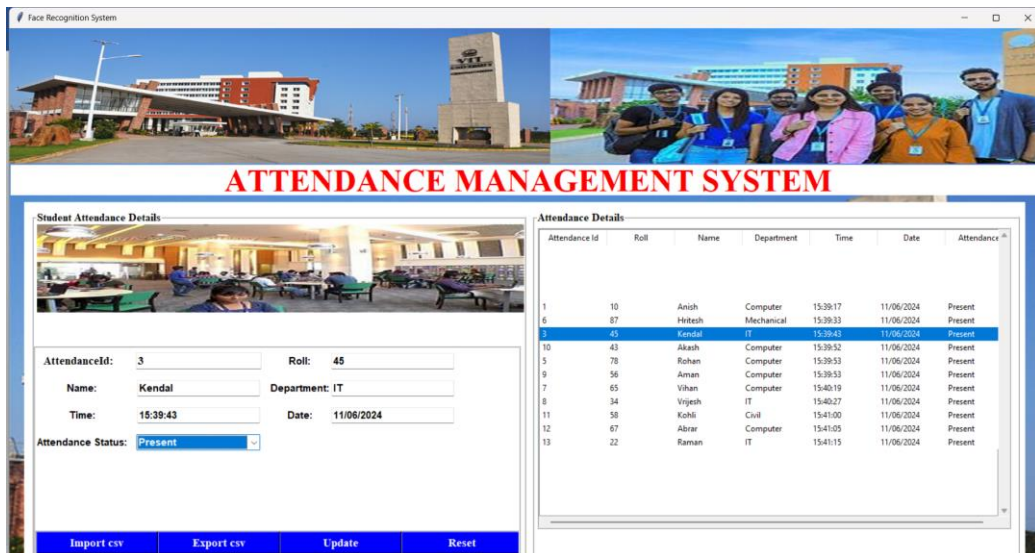
3. Train Data Set



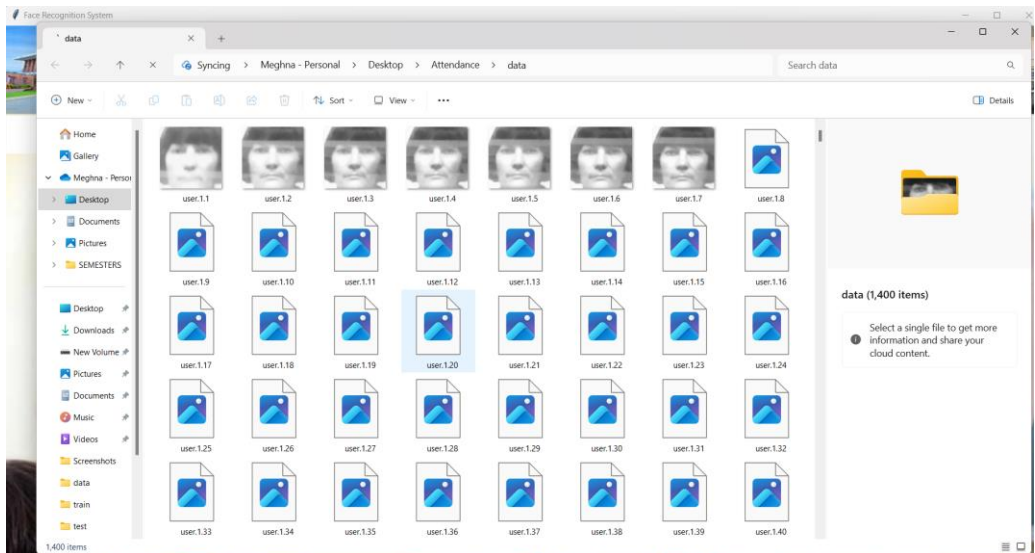
4. Face Recognition



5. Attendance Management System



6. Photos



RESEARCH PAPER DETAILS

Paper Title: Facial Frontiers: Unveiling the Potential of LBPHs and Haar Cascades in Facial Recognition for Enhanced School Security

Authors:

- **Students:** Krish Goel, Meghna Das, Aastha Kumar, Godavari Tanmayi, T Suraj Kumar
- **Faculty Supervisor:** Dr. Suganya R.
- **Faculty Co-Supervisor:** Dr. Subbulakshmi T.

Journal Name: International Journal For Multidisciplinary Research (IJFMR)

Volume & Issue: Volume 6, Issue 4 (July-August 2024)

DOI: <https://doi.org/10.36948/ijfmr.2024.v06i04.24527>

Published on: 2024-07-12

Paper Link: <https://www.ijfmr.com/research-paper.php?id=24527>

Issue Page Link: <https://www.ijfmr.com/publication-archive.php?volume=6&issue=4>

SUMMARY

Imports and Libraries

The project starts by importing necessary Libraries and Modules:

- **Tkinter:** For building the GUI application.
- **PIL:** To handle image processing tasks, like resizing and displaying images.
- **cv2 (OpenCV):** For Computer Vision tasks, including Face Detection and Recognition.
- **os:** For interacting with the operating system, such as reading directories and file operations.
- **numpy:** For numerical computations.
- **mysql.connector:** For connecting to and interacting with MySQL databases.
- **time.strftime:** For formatting timestamps.
- **datetime.datetime:** For manipulating dates and times.

Face_Recognition Class Initialization

- This class initializes the main application window (**root**) with a specific geometry and title.
- It sets up a title label at the top of the window with the text "FACE RECOGNITION" in a large, bold font and green color.

Displaying Images

- **1st Image:** Loads an image (**face_rec.jpg**) from a file, resizes it to 650x700 pixels, and displays it on the left side of the window.
- **2nd Image:** Loads another image (**phone.jpg**), resizes it to 950x700 pixels, and displays it on the right side of the window.
- A button labeled "Face Recognition" is placed at the bottom of the right image for initiating the Face Recognition process.

mark_attendance Method

- This method marks attendance when a face is recognized.
- It reads and writes to a CSV file (**attendance.csv**).

- Before marking attendance, it checks if the student's information is not already in the CSV file.
- It records the student's ID, roll number, name, department, current date, and time when marking attendance.

face_recog Method

- This method performs the core Face Recognition functionality using a webcam feed.
- **draw_boundary Function:** Draws rectangles around detected faces and labels them with student information (ID, roll number, name, department).
- **recognize Function:** Calls **draw_boundary** and then displays the image with rectangles and labels.
- Uses a pre-trained LBPH Face Recognizer (**clf**) and a Haar Cascade classifier (**faceCascade**) for Face Detection.
- Opens a webcam feed using OpenCV (**video_cap**) and continuously detects faces in real-time.
- If a recognized face's confidence level is above a threshold (77%), it marks attendance using the **mark_attendance** method.
- The process stops when the user presses the 'q' key.

Main Function

- Initializes the Tkinter window (**root**), creates an instance of the **Face_Recognition** class (**obj**), and starts the GUI event loop.

CONCLUSION

1. Purpose and Objective

- The primary goal of this project is to automate attendance management through real-time Face Recognition.
- It leverages Computer Vision techniques to detect and recognize faces, marking attendance for recognized individuals.

2. Technological Stack

- **Tkinter:** Used for building the Graphical User Interface (GUI), providing a user-friendly environment to initiate Face Recognition.
- **OpenCV:** Provides functionalities for Face Detection (**Haar Cascade Classifier**) and Recognition (**LBPH Face Recognizer**).
- **PIL (Pillow):** Handles Image Processing tasks like loading, resizing, and displaying images.
- **MySQL Connector:** Interacts with a MySQL database to manage student information.
- **Python Libraries:** **os**, **numpy**, **datetime** for general file operations, numerical computations, and date/time manipulations.

3. Application Interface

- **GUI Layout:** The Application Interface is divided into two sections, displaying images related to Face Recognition and a button to start the Recognition process.
- **Images:** Two images are displayed: one for the Face Recognition concept and another for the Application context (e.g., an Attendance System).
- **Button:** A "Face Recognition" button allows users to initiate the recognition process.

4. Face Recognition Process

- **Initialization:** The application initializes with a main window defined by **Tkinter** and sets the title and the geometry.
- **Face Detection:** Utilizes a **Haar Cascade Classifier** to detect faces in real-time from a webcam feed (**video_cap**).
- **Face Recognition:** Uses a pre-trained **LBPH Face Recognizer (clf)** to recognize faces based on a dataset previously trained and saved as **classifier.xml**.

- **Attendance Marking:** When a recognized face is detected with confidence above a threshold, attendance is marked by recording the student's ID, roll number, name, department, date, and time into a CSV file (**attendance.csv**).

5. Database Integration

- **MySQL Connection:** Connects to a MySQL database (**face_recognizer**) to fetch student information (ID, name, roll number, department) during Face Recognition.
- **Attendance Logging:** Logs attendance data directly into a CSV file (**attendance.csv**), ensuring easy retrieval and management of attendance records.

6. Enhancements and Future Developments

- **Database Integration:** Replace CSV file storage with MySQL database storage for better scalability, data management, and security.
- **User Interface:** Enhance the GUI with additional features like student information display, attendance statistics, and error handling.
- **Performance Optimization:** Optimize Face Recognition algorithms and database operations for improved system performance with larger datasets.
- **Security Measures:** Integrate additional security measures, such as access control and data encryption, to protect sensitive attendance information.

7. Applications

- **Educational Institutions:** Automate classroom attendance to improve efficiency and accuracy, reducing administrative workload.
- **Corporate Environments:** Track employee attendance for payroll and security purposes, enhancing workplace management.
- **Public Events:** Manage participant registration and attendance, ensuring accurate records for event organizers.

8. Conclusion

- The Face Recognition project demonstrates the practical application of Computer Vision and database technologies in automating attendance management.
- It offers a scalable solution suitable for educational institutions, corporate offices, and public events.
- By refining and optimizing the system, it can streamline attendance tracking processes, improve security measures, and enhance overall user experience.

FUTURE SCOPE

1. Database Integration

- **MySQL Integration:** Replace CSV file storage with a MySQL database to manage student information and attendance records.
- **Data Security:** Implement Data Encryption and Access Control measures to protect sensitive information.

2. User Interface Enhancements

- **Interactive GUI:** Develop a more interactive and intuitive User Interface using modern design principles.
- **Attendance Statistics:** Display attendance statistics, graphs, and reports for better Data Visualization.
- **Error Handling:** Implement robust error handling and feedback mechanisms for better user experience.

3. Performance Optimization

- **Algorithm Optimization:** Optimize Face Recognition algorithms to improve accuracy and speed, especially with larger datasets.
- **Real-Time Processing:** Implement real-time video processing for smoother and faster Face Detection and Recognition.

4. Enhanced Features

- **Live Updates:** Provide real-time updates and notifications for attendance marking and system status.
- **Mobile Integration:** Develop a mobile application to access attendance data and notifications remotely.

5. Integration with Biometrics

- **Multimodal Biometrics:** Integrate fingerprint or iris recognition for multimodal biometric authentication.
- **Voice Recognition:** Explore voice recognition for enhanced security and accessibility.

6. Scalability and Deployment

- **Cloud Integration:** Deploy the application on cloud platforms for scalability and accessibility.
- **Distributed Systems:** Implement a distributed system architecture for handling multiple recognition endpoints.

7. Security Enhancements

- **Face Mask Detection:** Integrate face mask detection to enforce safety protocols during attendance marking.
- **Anomaly Detection:** Implement anomaly detection to identify and flag suspicious activities.

Potential Modifications

1. Enhanced Database Connectivity

- **Migration to NoSQL Databases:** Consider using NoSQL databases like MongoDB for flexible schema and scalability.

2. Integration with IoT Devices

- **IoT Sensors:** Integrate IoT sensors to automate attendance marking based on physical presence in specific locations.

3. Machine Learning Integration

- **Deep Learning Models:** Explore Deep Learning models like CNNs for more accurate Face Recognition.

4. Cross-Platform Compatibility

- **Web Interface:** Develop a web-based interface for accessing attendance records and managing the system remotely.

5. Continuous Integration and Deployment (CI/CD)

- **Automation:** Implement CI/CD pipelines for automated testing, deployment, and updates.

6. Feedback Mechanism

- **User Feedback:** Incorporate user feedback mechanisms to continuously improve the application based on user experiences.

7. Regulatory Compliance

- **GDPR and Data Privacy:** Ensure compliance with data privacy regulations like GDPR for handling personal data.

REFERENCES

Facial Recognition

1. Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
2. Ahonen, T., Hadid, A., & Pietikäinen, M. (2006). Face recognition with local binary patterns. *European Conference on Computer Vision (ECCV)*.
3. Dalal, N., & Triggs, B. (2005). Histograms of oriented gradients for human detection. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
4. Turk, M., & Pentland, A. (1991). Eigenfaces for recognition. *Journal of Cognitive Neuroscience*.
5. Belhumeur, P. N., Hespanha, J. P., & Kriegman, D. J. (1997). Eigenfaces vs. Fisherfaces: Recognition using class specific linear projection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
6. Liu, C., Shum, H. Y., & Zhang, C. (2002). A two-stage approach to hallucinating faces: Global parametric model and local nonparametric model. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
7. Zhou, Z. H., & Chellappa, R. (2012). Unconstrained face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
8. Yang, M., Zhang, L., Feng, X., & Zhang, D. (2010). Fisher discrimination dictionary learning for sparse representation. *IEEE Transactions on Image Processing*.

Attendance Management Systems

9. Ogunleye, G., Olakanmi, O., & Aniebonam, C. (2013). Development of an automatic attendance system using face recognition. *International Journal of Advanced Computer Science and Applications*.
10. Bhosale, P. R., & Jagdale, R. S. (2016). Automated attendance system using face recognition. *International Journal of Computer Science and Information Technologies*.
11. Ullah, A., Shahzad, M. A., & Wahid, A. (2017). Student attendance system based on face recognition using raspberry pi. *IEEE Access*.
12. Singh, R., & Venugopal, K. R. (2016). Automated attendance management system using face recognition. *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)*.

Biometric Technologies

13. Jain, A. K., Ross, A., & Prabhakar, S. (2004). An introduction to biometric recognition. *IEEE Transactions on Circuits and Systems for Video Technology*.

14. Prabhakar, S., Pankanti, S., & Jain, A. K. (2003). Biometric recognition: Security and privacy concerns. *IEEE Security & Privacy Magazine*.
15. Rattani, A., & Verma, S. (2014). A review on fingerprint recognition system. *International Journal of Advanced Research in Computer and Communication Engineering*.
16. Das, P., & Majumder, S. (2018). Palmprint recognition system: A review. *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*.

Machine Learning and Deep Learning

17. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*.
18. Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.
19. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2015). Going deeper with convolutions. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.
20. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*.

IoT and Mobile Integration

21. Gope, P., & Uddin, M. Z. (2013). A survey of mobile cloud computing: Architecture, applications, and approaches. *Wireless Personal Communications*.
22. Xu, L. D., He, W., & Li, S. (2014). Internet of things in industries: A survey. *IEEE Transactions on Industrial Informatics*.
23. Khairnar, S. R., & Bhide, N. S. (2014). An efficient student attendance system using QR codes and GSM technology. *2014 International Conference on Signal Propagation and Computer Technology (ICSPCT)*.

Security and Privacy

24. Alzahrani, S. M. (2018). Face recognition for security applications. *International Journal of Computer Science and Network Security*.
25. Weymann, C., Haase, M., & Eisert, P. (2016). Overview and outlook on 3D face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.
26. Singh, R., & Venugopal, K. R. (2015). Privacy preserving face recognition using multimodal biometrics. *2015 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*.

System Integration and Optimization

27. Thakur, N., & Sharma, S. (2014). Design and implementation of attendance management system using face recognition. *2014 International Conference on Computing for Sustainable Global Development (INDIACom)*.

28. Li, M., Li, X., Shen, C., & Yang, J. (2018). Real-time face recognition and tracking using deep learning. *IEEE Transactions on Circuits and Systems for Video Technology*.
29. Nguyen, D. T., Nguyen, T. T., Nguyen, V. H., & Nguyen, T. M. (2017). A framework for cloud-based student attendance system using face recognition. *2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*.

Ethical and Legal Considerations

30. Jain, A. K., & Dass, S. C. (2019). Can biometrics be spoofed? A lesson in security. *IEEE Signal Processing Magazine*.