INTRODUCTION

Face Recognition Attendance System (FRAS) is the complete system to record attendance without much interference in very less time. It saves time and effort of the teacher in the university where we only get one hours of time for one class. FRAS record attendance with the help of face recognition. It marks the attendance of all students in the class by obtaining data from the database and match with present face data and saves the result in the database (Excel Sheet). This system makes the attendance process easy and minimizes the interference of the teachers. Which provides them more time to teach and take full advantage of their period time.

The main objective of this project is to develop face recognition based automated student attendance system. In order to achieve better performance, the test images and training images of this proposed approach are limited to frontal and upright facial images that consist of a single face only. The test images and training images must be captured by using the same device to ensure no quality difference. In addition, the students must register in the database to be recognized. The enrolment can be done on the spot through the user-friendly interface.

Facial recognition attendance system is an application of facial recognition technology that enables organizations to automatically record the attendance of employees or students. This system uses a camera to capture the image of an individual's face and then uses algorithms to match the image to an existing database of faces to verify the identity of the individual.

Python is a popular programming language used for various applications, including machine learning and computer vision. With the availability of various open-source libraries, it has become easier to develop facial recognition systems using Python.To create a facial recognition attendance system using Python, we need to use various libraries like OpenCV, dlib, and face recognition. OpenCV is a popular library for computer vision, and dlib provides facial landmark detection and face recognition functionality. The face recognition library provides face recognition and identification features.The basic workflow for a facial recognition attendance system using Python involves capturing the image of an individual's face using a camera, processing the image to extract facial features, and matching the features with the existing database of faces to verify the identity. Once the identity is verified, the attendance can be recorded automatically.

Overall, facial recognition attendance systems using Python offer a reliable and efficient solution for attendance management in organizations. With the advancement of facial recognition technology and the availability of powerful Python libraries, it has become easier to develop such systems.

LITERATURE SURVEY

Facial recognition algorithms: The Eigenfaces algorithm is one of the earliest facial recognition algorithms and is based on Principal Component Analysis (PCA). Fisherfaces algorithm improves upon Eigenfaces by using Fisher Discriminant Analysis (FDA) to separate between different classes of faces. Local Binary Patterns (LBP) is another algorithm that uses texture information to recognize faces. Deep Learning-based approaches such as Convolutional Neural Networks (CNN) have shown promising results in facial recognition.

Face detection and tracking: Viola-Jones algorithm is a popular algorithm for face detection and tracking that uses Haar-like features and AdaBoost to classify faces. Histogram of Oriented Gradients (HOG) is another technique that uses gradient information to detect faces. Recently, CNN-based approaches have shown significant improvements in face detection and tracking.

Python libraries for facial recognition: OpenCV is a popular computer vision library that provides various functionalities for facial recognition, including face detection, recognition, and tracking. Dlib is another popular library that provides facial landmark detection and face recognition functionality. The face recognition library provides an easy-to-use API for face recognition and identification.

Performance evaluation: Accuracy, precision, recall, and F1 score are some of the commonly used metrics for evaluating the performance of facial recognition systems. The Labeled Faces in the Wild (LFW) dataset is a popular benchmark dataset for evaluating facial recognition systems.

Application scenarios: Facial recognition attendance systems are widely used in various industries, such as education, healthcare, and security. In education, facial recognition systems can help to automate attendance management and improve accuracy. In healthcare, facial recognition systems can be used to identify patients and ensure the correct medication is administered. In security, facial recognition systems can help to identify suspects and prevent crimes.

Ethical and privacy concerns: The use of facial recognition systems raises various ethical and privacy concerns, such as the potential for misuse, bias, and violation of privacy rights. It is essential to address these concerns and develop facial recognition systems that are transparent, fair, and respectful of privacy rights.

Overall, the literature survey provides a comprehensive understanding of the state-of-the-art in facial recognition attendance systems using Python and helps to identify the research gaps and potential areas for improvement

REQUIREMENT ANALYSIS

Introduction

The main purpose for preparing this software is to give a general insight into the analysis and requirement of the existing system or situation and for determining the operating characteristics of the system.

General Description

This project requires a computer system with the following software's:

1. Operating System - Windows 7 or later (latest is best)
2. Python 3.7 (including all necessary libraries)
3. Microsoft Excel 2007 or later
4. Google chrome (for cloud related services)

Specific Requirement

This face recognition attendance system project requires OpenCV-a python library with built-in LBPH face recognizer for training the dataset captured via the camera. Coming to the technical feasibility following are the requirements:

Hardware and Software Requirements:

Processor: Intel Processor IV (latest is recommended)

Ram: 4 GB (Higher would be good)

Hard disk: 40 GB

Monitor: RBG led

Keyboard: Basic 108 key keyboard

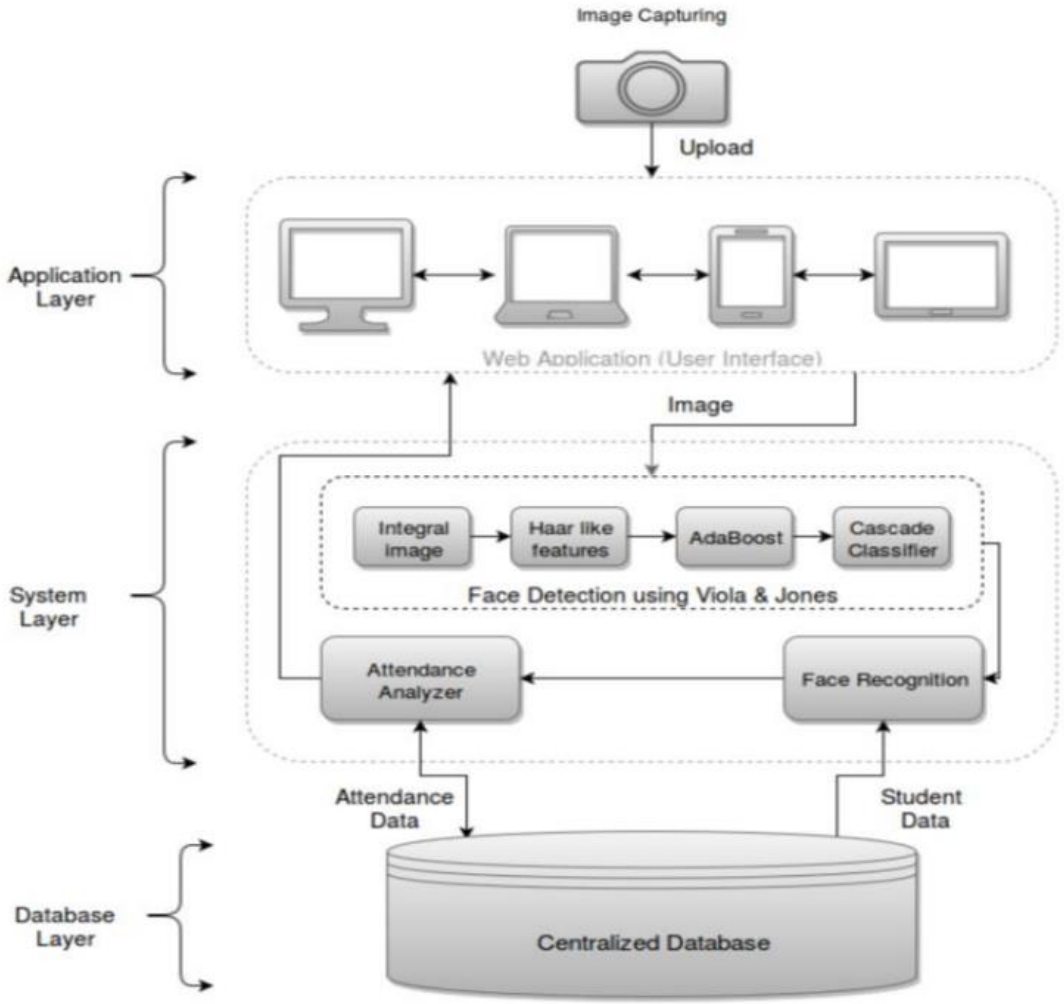
Mouse: Optical Mouse (or touchpad would work)

Camera: 1.5 Mega pixel (Or more)

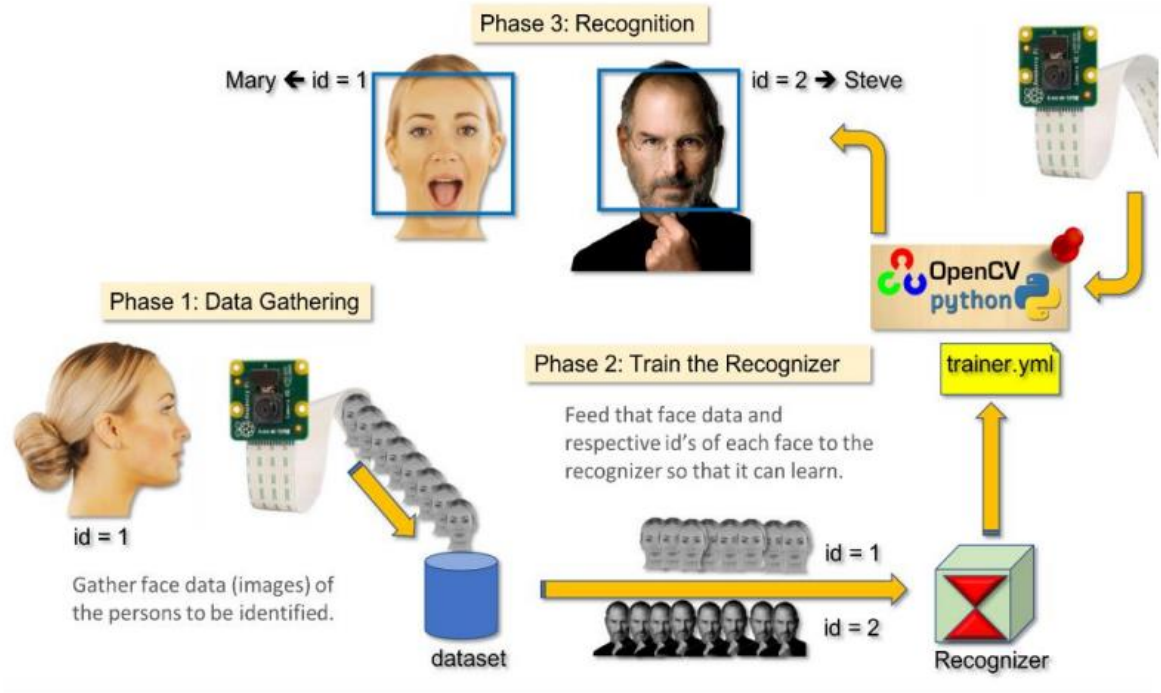
SYSTEM DESIGN

In the first step image is captured from the camera. There are illumination effects in the captured image because of different lighting conditions and some noise which is to be removed before going to the next steps. Histogram normalization is used for contrast enhancement in the spatial domain. Median filter is used for removal of noise in the image. There are other techniques like FFT and low pass filter for noise removal and smoothing of the images, but median filter gives good results.

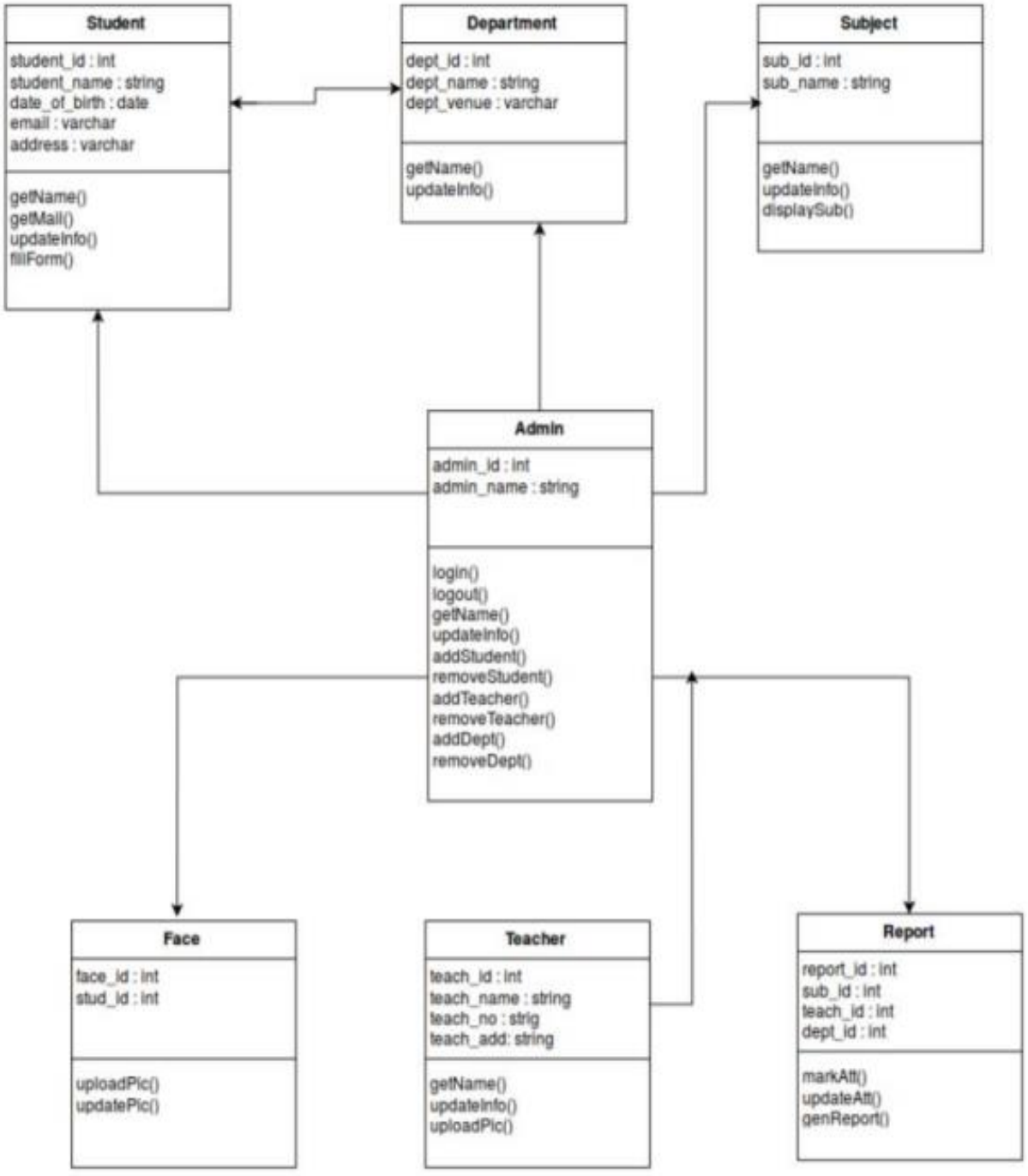
DATABASE DESIGN



DESIGN NOTATION



DETAILED DESIGN



CODING

1. Creating database

```
import sqlite3

conn = sqlite3.connect('database/database.db')
c = conn.cursor()
c.execute("CREATE TABLE students (UID text,student_name text, attendance text)")
conn.commit()
conn.close()
```

2. Training the Database

```
import os,cv2
import numpy as np
from PIL import Image

recognizer = cv2.face.LBPHFaceRecognizer_create()
detector = cv2.CascadeClassifier('cascades/haarcascade_frontalface_default.xml')

def getImagesAndLabels(path):
    #get the path of all the files in the folder
    imagePaths=[os.path.join(path,f) for f in os.listdir(path)]
    #create empth face list
    faceSamples=[]
    #create empty ID list
    Ids=[]
    #now looping through all the image paths and loading the Ids and the images
    for imagePath in imagePaths:
        #loading the image and converting it to gray scale
        pilImage=Image.open(imagePath).convert('L')
        #Now we are converting the PIL image into numpy array
        imageNp=np.array(pilImage,'uint8')
        #getting the Id from the image
        Id=int(os.path.split(imagePath)[-1].split(".")[1])
        # extract the face from the training image sample
        faces=detector.detectMultiScale(imageNp)
        #If a face is there then append that in the list as well as Id of it
        for (x,y,w,h) in faces:
            faceSamples.append(imageNp[y:y+h,x:x+w])
            Ids.append(Id)
    return faceSamples,Ids

faces,Ids = getImagesAndLabels('dataSet/')
s = recognizer.train(faces, np.array(Ids))
recognizer.write('trainer/trainer.yml')

if os.path.exists(os.getcwd()+'/trainer/trainer.yml'):
    os.system("python message_gui.py")
```

3. Writing database in excel:

```
import sqlite3 as sql
import win32com.client as win32
import os

os.remove('database/Students_Database_Sheet.xlsx')
conn = sql.connect('database/database.db')
c = conn.cursor()
#create out instance of excel
excelApp = win32.gencache.EnsureDispatch("Excel.Application")
#get the workbook
if not os.path.exists('database/Students_Database_Sheet.xlsx'):
    if not os.path.exists('database'):
        os.makedirs("database")
    ExcelWrkbook = excelApp.Workbooks.Add()
    ExcelWrkbook.SaveAs(os.getcwd()+'/database/Students_Database_Sheet.xlsx')
else:
    bookname = str(os.getcwd()+'/database/Students_Database_Sheet.xlsx')
    ExcelWrkbook = excelApp.Workbooks.Open(bookname)
#get the worksheet
Excelwrksheet = ExcelWrkbook.ActiveSheet
def get_data():
    c.execute("SELECT * FROM students ORDER BY UID")
    return c.fetchall()
students = get_data()
#creating the header
firstheaderCell = Excelwrksheet.Cells(1,1)
lastheaderCell = Excelwrksheet.Cells(1,3)
ExcelHeaderRange = Excelwrksheet.Range(firstheaderCell, lastheaderCell)
ExcelHeaderRange.Value = ('UID', 'Name', 'Attendance')
# get the length of a row
RowLength = len(students[0])
#print(RowLength)
#get the length of a column
ColLength = len(students)
#print(ColLength)
#define the first and last cell in out range
FirstCell = Excelwrksheet.Cells(2,1)
LastCell = Excelwrksheet.Cells(1+ColLength,RowLength)
#get the range
ExcelRange = Excelwrksheet.Range(FirstCell, LastCell)
ExcelRange.Value = students
#close and save the workbook
ExcelWrkbook.Close(True)
os.system("python message_gui.py")
conn.close()
```

4. Recognizer

```
import cv2
import os
import numpy as np
import time
import sys
import sqlite3 as sql

conn = sql.connect('database/database.db')
c = conn.cursor()

flag = 0
time_of_attendance = time.time()
face_cas = cv2.CascadeClassifier('cascades/haarcascade_frontalface_default.xml')
cap_video = cv2.VideoCapture(0)
recognizer = cv2.face.LBPHFaceRecognizer_create()
recognizer.read('trainer/trainer.yml')
font = cv2.FONT_HERSHEY_SIMPLEX
while True:
    ret, img = cap_video.read()
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cas.detectMultiScale(gray, 1.3, 7)
    for (x,y,w,h) in faces:
        roi_gray = gray[y:y+h,x:x+w]
        cv2.rectangle(img, (x,y), (x+w,y+h), (255,0,0))
        id,conf = recognizer.predict(roi_gray)
        cv2.putText(img,str(id)+" "+str(conf),(x,y-10),font,0.55,(120,255,120),1)
        if(conf<80):
            for row in c.execute("SELECT * FROM students WHERE UID = ?",(str(id),)):
                if(str(id) == row[0]):
                    c.execute("UPDATE students SET attendance= ? WHERE UID = ?",('Present',str(id)))
                    conn.commit()
                else:
                    os.system("python error_gui.py")
                    flag = flag+1
                    break;

    cv2.imshow('frame',img)
    period = 20
    if(flag == 10):
        os.system("python error_gui.py")
        break
    if(time.time()) > (time_of_attendance+period):
        break
    if(cv2.waitKey(100) & 0xFF == ord('q')):
        break


os.system("python Marking_attendance.py")

conn.close()
cap_video.release();
cv2.destroyAllWindows();
```

5. Marking attendance

```
import sqlite3 as sql
import win32com.client as win32
import os
from datetime import datetime

conn = sql.connect('database/database.db')
c = conn.cursor()

#create out instance of excel
excelApp = win32.gencache.EnsureDispatch("Excel.Application")

#get the workbook
if not os.path.exists('Attendance_Files/'+'Attendance'+str(datetime.now().date())+'.xlsx'):
    if not os.path.exists('Attendance_Files'):
        os.makedirs("Attendance_Files")
    ExcelWrkbook = excelApp.Workbooks.Add()
    ExcelWrkbook.SaveAs(os.getcwd()+'/Attendance_Files/Attendance'+str(datetime.now().date())+'.xlsx')
else:
    bookname = str(os.getcwd()+'/Attendance_Files/Attendance'+ str(datetime.now().date())+'.xlsx')
    ExcelWrkbook = excelApp.Workbooks.Open(os.getcwd()+'/Attendance_Files/Attendance'+ str(datetime.now().date())+'.xlsx')

#get the worksheet
Excelwrksheet = ExcelWrkbook.ActiveSheet

#get xl constants
xlRight = win32.constants.xlToRight
xlDown = win32.constants.xlDown

def get_data():
    c.execute("SELECT * FROM students ORDER BY student_name")
    return c.fetchall()

students = get_data()
#creating the header
firstheaderCell = Excelwrksheet.Cells(1,1)
lastheaderCell = Excelwrksheet.Cells(1,3)
ExcelHeaderRange = Excelwrksheet.Range(firstheaderCell, lastheaderCell)
ExcelHeaderRange.Value = ('ID', 'Name', 'Attendance')

# get the length of a row
RowLength = len(students[0])
#print(RowLength)

#get the length of a column
ColLength = len(students)
#print(ColLength)

#define the first and last cell in out range
FirstCell = Excelwrksheet.Cells(2,1)
LastCell = Excelwrksheet.Cells(1+ColLength,RowLength)

#get the range
ExcelRange = Excelwrksheet.Range(FirstCell, LastCell)
ExcelRange.Value = students

#close and save the workbook
ExcelWrkbook.Close(True)

#####testing bachi hai####
for row in c.execute("SELECT * FROM students"):
    c.execute("UPDATE students SET attendance = ?",('Absent',))
    conn.commit()

os.system("python message_gui.py")

conn.close()
```

6. GUI

```
from tkinter import *
from datetime import datetime
import os

root = Tk()
root.configure(background="ivory4")
root.title("AUTOMATIC ATTENDANCE MANAGEMENT USING FACE RECOGNITION")

label1 = Label(root, text="FACE RECOGNITION ATTENDANCE SYSTEM",font=("times new roman",20),bg="red4",fg="white",height=2)
label1.grid(row=0,rowspan=2,columnspan=2,sticky=N+E+W+S,padx=5,pady=5)

def c_dataset():
    os.system("python capture_database.py")

def write_e():
    os.system("python writing_in_excel.py")

def train_d():
    os.system("python training_dataset.py")

def m_attendance():
    os.system("python recognizer.py")

def v_attendance():
    os.startfile(os.getcwd()+"/Attendance_Files/attendance"+str(datetime.now().date())+'.xlsx')

def d_dataset():
    os.system("python delete_database.py")

def destroy():
    root.destroy()
button1 = Button(root, text="Create Dataset",font=("times new roman",20),bg="dodgerblue2",fg='white',command=c_dataset)
button1.grid(row=2,rowspan=2,column=0,sticky=N+E+W+S,padx=10,pady=10)

button2 = Button(root, text="Write in Excel",font=("times new roman",20),bg="dodgerblue2",fg='white',command=write_e)
button2.grid(row=2,rowspan=2,column=1,sticky=N+E+W+S,padx=10,pady=10)

button3 = Button(root, text="Train Dataset",font=("times new roman",20),bg="dodgerblue3",fg='white',command=train_d)
button3.grid(row=4,rowspan=2,column=0,sticky=N+E+W+S,padx=10,pady=10)

button4 = Button(root, text="Mark Attendance",font=("times new roman",20),bg="dodgerblue3",fg='white',command=m_attendance)
button4.grid(row=4,rowspan=2,column=1,sticky=N+E+W+S,padx=10,pady=10)

button5 = Button(root, text="View Attendance Sheet",font=("times new roman",20),bg="dodgerblue4",fg='white',command=v_attendance)
button5.grid(row=6,rowspan=2,column=0,sticky=N+E+W+S,padx=10,pady=10)

button6 = Button(root, text="Delete Dataset",font=("times new roman",20),bg="dodgerblue4",fg='white',command=d_dataset)
button6.grid(row=6,rowspan=2,column=1,sticky=N+E+W+S,padx=10,pady=10)

button7 = Button(root,text="Exit",font=('times new roman',20),bg="red4",fg="white",command=destroy)
button7.grid(row=8,columnspan=2,sticky=N+E+W+S,padx=10,pady=10)


root.mainloop()
```

TESTING

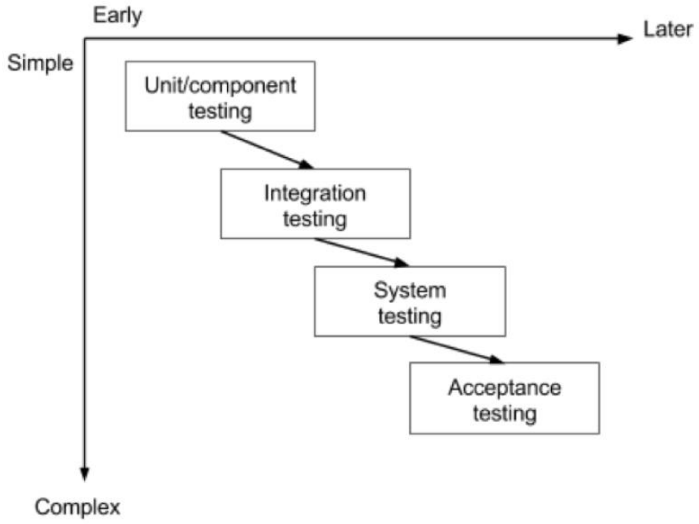
Functional Testing

Functional testing is done at every function level for example there is function named as assure_path_exists which is responsible to create directory for dataset, that function is tested if the directory is created or not. Similarly, all the functions are tested separately before integrating.

Structural Testing

Structural testing is performed after we integrated each module. After integrating the path creation function and capture image function then we tested that after capturing image the photos were saved in the correct directory that was created by assure_path_exists

Levels of Testing



Unit testing has been performed on the project by verifying each module independently, isolating it from the others. Each module is fulfilling the requirements as well as desired functionality. Integration testing would be performed to check if all the functionalities are working after integration.

Testing the project

Testing early and testing frequently is well worth the effort. By adopting an attitude of constant alertness and scrutiny in all your projects, as well as a systematic approach to testing, the tester can pinpoint any faults in the system sooner, which translates in less time and money wasted later.

IMPLEMENTATION

Implementation of the project

The complete project is implemented using python 3.7 (or later), main library used was OpenCV, SQLite3, pywin32, Pillow, ms excel is used for database purpose to store the attendance.

Capturing the dataset

The first and foremost module of the project is capturing the dataset. When building an on-site face recognition system and when you must have physical access to a specific individual to assemble model pictures of their face, you must make a custom dataset. Such a framework would be necessary in organizations where individuals need to physically appear and attend regularly. To retrieve facial images and make a dataset, we may accompany them to an exceptional room where a camcorder is arranged to (1) distinguish the (x, y)- directions of their face in a video stream and (2) store the frames containing their face to database. We may even play out this process over a course of days or weeks in order to accumulate instances of their face in:

1. Distinctive lighting conditions
2. Times of day
3. Mind-sets and passionate states to make an increasingly differing set of pictures
4. illustrative of that specific individual's face

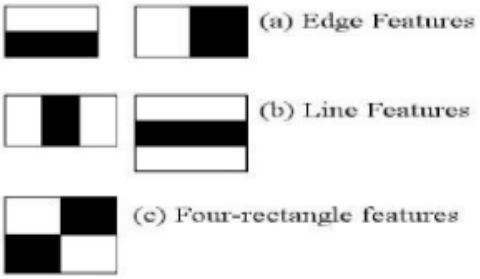
This Python script will:

1. Access the camera of system
2. Detect faces
3. Write the frame containing the face to database

Face detection using OpenCV:

In this project we have used OpenCV, to be precise, the Haar-cascade classifier for face detection. Haar Cascade is an AI object detection algorithm proposed by Paul Viola and Michael Jones in their paper "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001. It is an AI based methodology where a cascade function is prepared from a great deal of positive and negative pictures (where positive pictures are those where the item to be distinguished is available, negative is those where it isn't). It is then used to tell objects in different pictures. Fortunately, OpenCV offers pre-trained Haar cascade algorithms, sorted out into classifications (faces, eyes, etc.), based on the pictures they have been prepared on.

Presently let us perceive how this algorithm solidly functions. The possibility of Haar cascade is separating features from pictures utilizing a sort of 'filter', like the idea of the convolutional kernel. These filters are called Haar features and resemble that:



The idea is passing these filters on the picture, investigating one part at the time. At that point, for every window, all the pixel powers of, separately, white and dark parts are added. Ultimately, the value acquired by subtracting those two summations is the value of the feature extracted. In a perfect world, an extraordinary value of a feature implies it is pertinent. On the off chance that we consider the Edge (a) and apply it to the accompanying B&W pic:

We will get a noteworthy value, subsequently the algorithm will render an edge highlight with high likelihood. Obviously, the genuine intensities of pixels are never equivalent to white or dark, and we will frequently confront a comparable circumstance:

By and by, the thought continues as before: the higher the outcome (that is, the distinction among highly contrasting black and white summations), the higher the likelihood of that window of being a pertinent element.

To give you a thought, even a 24x24 window results in more than 160000 highlights, and windows inside a picture are a ton. How to make this procedure progressively effective? The arrangement turned out with the idea of Summed-area table, otherwise called Integral Image. It is an information structure and algorithm for producing the sum of values in a rectangular subset of a matrix. The objective is diminishing the amount of calculations expected to get the summations of pixel intensities in a window.

Following stage additionally includes proficiency and enhancement. Other than being various, features may likewise be unessential. Among the features we get (that are more than 160000), how might we choose which ones are great? The response to this inquiry depends on the idea of Ensemble strategy: by consolidating numerous algorithms, weak by definition, we can make a solid algorithm. This is cultivated utilizing Ad boost which both chooses the best features and prepares the classifiers that utilize them. This algorithm builds a strong classifier as a simple mix of weighted basic weak classifiers.

We are nearly done. The last idea which should be presented is a last component of advancement. Even though we decreased our 160000+ highlights to an increasingly reasonable number, the latter is still high: applying every one of the features on every one of the windows will consume a great deal of time. That is the reason we utilize the idea of Cascade of classifiers: rather than applying every one of the features on a window, it bunches the features into various phases of classifiers and applies individually. On the off chance that a window comes up short (deciphered:

the distinction among white and dark summations is low) the primary stage (which typically incorporates few features), the algorithm disposes it: it won't think about outstanding features on it. If it passes, the algorithm applies the second phase of features and continues with the procedure

Storing the data:

In order to store the captured dataset, we use SQLite. SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL database engine. SQLite is the most widely deployed SQL database engine in the world.

SQLite is renowned for its extraordinary element zero-design, which implies no intricate arrangement or organization is required. In SQLite, sqlite3 command is utilized to make another SQLite database. You don't have to have any unique benefit to make a database. Consider a situation when you have different databases accessible and you need to utilize any of them at once. SQLite ATTACH DATABASE command is utilized to choose a specific database, and after this command, all SQLite statements will be executed under the appended database.

SQLite CREATE TABLE statement is utilized to make another table in any of the given database. Making a primary table includes naming the table and characterizing its columns and every column's information type.

SQLite INSERT INTO Statement is utilized to include new tuples of information into a table in the database.

SQLite-Python

SQLite3 can be incorporated with Python utilizing sqlite3 module, which was written by Gerhard Haring. It furnishes an SQL interface agreeable with the DB-API 2.0 determination depicted by PEP 249. You don't have to introduce this module independently because it is delivered as a matter of course alongside Python version 2.5.x onwards.

To utilize sqlite3 module, you should initially make an association object that speaks to the database and afterward alternatively you can make a cursor object, which will help you in executing all the SQL explanations.

The face recognition systems can operate basically in two modes:

1. Verification or authentication of a facial image: it basically compares the input facial image with the facial image related to the user which is requiring the authentication. It is basically a 1x1 comparison.
2. dentification or facial recognition: it basically compares the input facial image with all facial images from a dataset with the aim to find the user that matches that face. It is basically a 1xN comparison.

There are different types of face recognition algorithms, for example:

1. Eigenfaces (1991)
2. Local Binary Patterns Histograms (LBPH) (1996)
3. Fisher faces (1997)
4. Scale Invariant Feature Transform (SIFT) (1999)
5. Speed Up Robust Features (SURF) (2006)

This project uses the LBPH algorithm.

Training Database (LBPH algorithm):

As it is one of the easier face recognitions algorithms, I think everyone can understand it without major difficulties.

Introduction: Local Binary Pattern (LBP) 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.

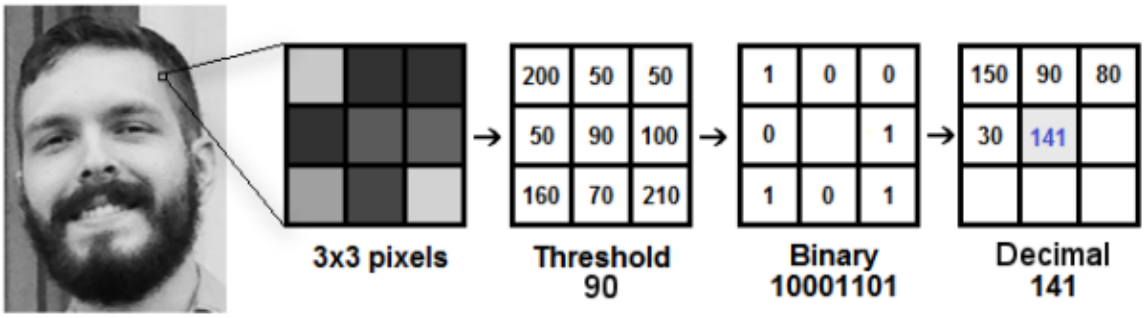
It was first described in 1994 (LBP) and has since been found to be a powerful feature for texture classification. It has further been determined that when LBP is combined with histograms of oriented gradients (HOG) descriptor, it improves the detection performance considerably on some datasets.

Using the LBP combined with histograms we can represent the face images with a simple data vector.

DETAILED EXPLANATION OF WORKING OF LBPH

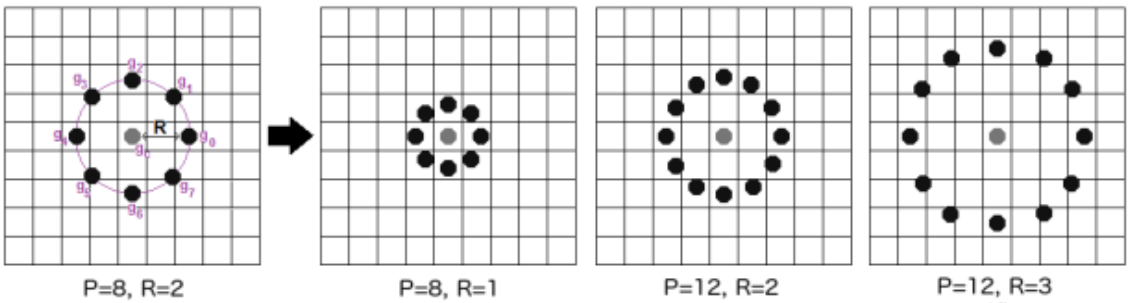
1. **Parameters:** the LBPH uses 4 parameters:
    - **Radius:** the radius is used to build the circular local binary pattern and represents the radius around the central pixel. It is usually set to 1.
    - **Neighbors:** the number of sample points to build the circular local binary pattern. Keep in mind: the more sample points you include, the higher the computational cost. It is usually set to 8.
    - **Grid X:** the number of cells in the horizontal direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
    - **Grid Y:** the number of cells in the vertical direction. The more cells, the finer the grid, the higher the dimensionality of the resulting feature vector. It is usually set to 8.
2. **Training the Algorithm:** First, we need to train the algorithm. To do so, we need to use a dataset with the facial images of the people we want to recognize. We need to also set an ID (it may be a number or the name of the person) for each image, so the algorithm will use this information to recognize an input image and give you an output. Images of the same person must have the same ID. With the training set already constructed, let's see the LBPH computational steps.
3. **Applying the LBP operation:** The first computational step of the LBPH is to create an intermediate image that describes the original image in a better way, by highlighting thefacial characteristics. To do so, the algorithm uses a concept of a sliding window, based on the parameter's **radius and neighbors**.

The image below shows this procedure:



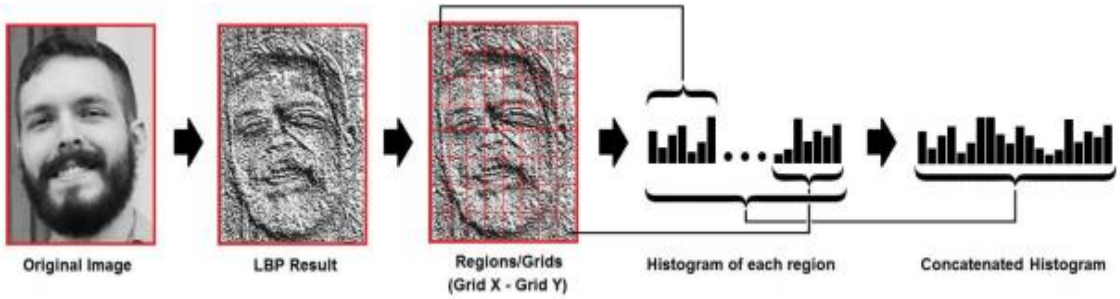
Based on the image above, let's break it into several small steps so we can understand it easily:

- Suppose we have a facial image in grayscale.
- We can get part of this image as a window of 3x3 pixels.
- It can also be represented as a 3x3 matrix containing the intensity of each pixel (0~255).
- Then, we need to take the central value of the matrix to be used as the threshold.
- This value will be used to define the new values from the 8 neighbors.
- For each neighbor of the central value (threshold), we set a new binary value. We set 1 for values equal or higher than the threshold and 0 for values lower than the threshold.
- Now, the matrix will contain only binary values (ignoring the central value). We need to concatenate each binary value from each position from the matrix line by line into a new binary value (e.g., 10001101). Note: some authors use other approaches to concatenate the binary values (e.g., clockwise direction), but the result will be the same.
- Then, we convert this binary value to a decimal value and set it to the central value of the matrix, which is a pixel from the original image.
- At the end of this procedure (LBP procedure), we have a new image which represents better the characteristics of the original image.
- Note: The LBP procedure was expanded to use a different number of radius and neighbors, it is called Circular LBP.



It can be done by using bilinear interpolation. If some data point is between the pixels, it uses the values from the 4 nearest pixels (2x2) to estimate the value of the new data point.

4. **Extracting the Histograms:** Now, using the image generated in the last step, we can use the Grid X and Grid Y parameters to divide the image into multiple grids, as can be seen in the following image:



Based on the image above, we can extract the histogram of each region as follows:

- As we have an image in grayscale, each histogram (from each grid) will contain only 256 positions (0~255) representing the occurrences of each pixel intensity.
- Then, we need to concatenate each histogram to create a new and bigger histogram. Supposing we have 8x8 grids, we will have 8x8x256=16.384 positions in the final histogram. The final histogram represents the characteristics of the image original image.

After detecting the face, image need to crop as it has only face nothing else focused. To do so Python Imaging Library (PIL) or also known as Pillow is used. PIL is a free library for python programming language that adds support for opening, manipulating, and saving many different images file formats. Capabilities of pillow:

- per-pixel manipulations,
- masking and transparency handling,
- image filtering, such as blurring, contouring, smoothing, or edge finding,
- image enhancing, such as sharpening, adjusting brightness, contrast or color,
- adding text to images and much more.

Face recognition

In this step, the algorithm is already trained. Each histogram created is used to represent each image from the training dataset. So, given an input image, we perform the steps again for this new image and creates a histogram which represents the image.

- So, to find the image that matches the input image we just need to compare two histograms and return the image with the closest histogram.

- We can use various approaches to compare the histograms (calculate the distance between two histograms), for example: Euclidean distance, chi-square, absolute value, etc. In this example, we can use the Euclidean distance (which is quite known) based on the following formula:

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

- So, the algorithm output is the ID from the image with the closest histogram. The algorithm should also return the calculated distance, which can be used as a 'confidence' measurement.
- We can then use a threshold and the 'confidence' to automatically estimate if the algorithm has correctly recognized the image. We can assume that the algorithm has successfully recognized if the confidence is lower than the threshold defined.

Conclusions

- LBPH is one of the easiest face recognition algorithms.
- It can represent local features in the images.
- It is possible to get great results (mainly in a controlled environment).
- It is robust against monotonic grey scale transformations.

It is provided by the OpenCV library (Open-Source Computer Vision Library).

Marking the attendance

The face(s) that have been recognized are marked as present into the database. Then the entire attendance data is written into an excel sheet that is dynamically created using pywin32 library of python. First, an instance is created for excel application and then new excel workbook is created and active worksheet of that file is fetched. Then data is fetched from database and written in the excel sheet.

Post implementation and software maintained

After the implementation of the software post implementation can be done by adding more features in the software like SMS tracking of the attendance of particular student for example if he is absent then a automatic scheduled SMS can be sent to their parent.

About the software maintenance part only the features can be maintained and improved over time and database could be large over time so a better data structure can be used for faster fetching of data could be done, for that purpose cloud storage can be used to minimize the latency of fetching data of a student.

CONCLUSION

In conclusion, face recognition attendance system in Python is a powerful technology that offers a reliable and efficient way to track attendance. It utilizes advanced algorithms to identify and verify an individual's identity through their facial features, making it a secure and accurate method for recording attendance.

By implementing this system, organizations can save time and resources that would otherwise be spent on manual attendance tracking. Furthermore, it can also help to reduce errors and eliminate the possibility of attendance fraud.

Python provides various libraries and tools that can be used to build a face recognition attendance system. Some of the popular libraries for face recognition include OpenCV, Dlib, and Face Recognition.

Overall, the face recognition attendance system in Python is a valuable tool that can help streamline attendance tracking processes and enhance security in various settings, including schools, offices, and other organizations.

FUTURE ENHANCEMENT

Facial recognition technology is continuously evolving and advancing, and there are several future enhancements that are likely to occur:

Improved accuracy

Facial recognition technology is already quite accurate, but there is always room for improvement. With the advancement of deep learning techniques and the availability of larger and more diverse datasets, facial recognition systems are likely to become even more accurate in the future.

Better privacy protection

As concerns about privacy and data security continue to grow, facial recognition technology will need to adapt to meet these challenges. Future enhancements may include stronger encryption, more secure storage methods, and better control over who has access to facial recognition data.

Increased speed and efficiency

As facial recognition technology becomes more widespread, there will be a growing need for faster and more efficient systems. Future enhancements may include improvements in hardware and software that enable faster processing and analysis of facial data.

Improved recognition in different environments

Facial recognition technology can sometimes struggle in low-light conditions, or when faces are partially obscured by clothing or accessories. Future enhancements may include improvements in lighting, camera technology, and algorithms that can better recognize faces under a variety of conditions.

Enhanced user experience

As facial recognition technology becomes more integrated into our daily lives, there will be a growing need for systems that are easy to use and intuitive. Future enhancements may include better user interfaces, voice recognition, and other features that make facial recognition technology more user-friendly.

BIBILOGRAPHY AND RESOURCE

LBPH - Local Binary Pattern Histogram (LBPH) algorithm is a simple solution on face recognition problem, which can recognize both front face and side face.

HDR - High-dynamic-range.

OpenCV - python library for computer vision.

Harcascade - A Harcascade is basically a classifier which is used to detect the object for which it has been trained for.

https://www.instructables.com/id/Real-time-Face-Recognition-an-End-to-end-Project/

https://pdfs.semanticscholar.org/b690/8248f52c917c8202e5bb1d39a19c0e018813.pdf

https://towardsdatascience.com/face-recognition-for-beginners-a7a9bd5eb5c2

https://www.tutorialspoint.com/

https://www.pyimagesearch.com/2018/06/11/how-to-build-a-custom-face-recognition-dataset

https://medium.com/dataseries/face-recognition-with-opencv-haar-cascade-a289b6ff042a

https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b

USER MANUAL

To run the software first we need to capture the image which we do using open cv video capture function that open camera for few seconds and capture only facial part of the image using frontal face haarcascades classifier, A haarcascades is classifier which is used to detect particular objects form source that is image or video, in our case it was frontal face detection so the cascade file contains the information about the facial symmetry of human face.

After capturing the face, the images need to train, for that there is a separate code that used for training the images what the trainer does is it set the image to the person who's the image is. For example, after training the images are named as user.UID.count.

And the last step is recognizing or mark attendance for that open the recognizer python file and it will open a camera window that will capture the person in the frame and the good thing about this algorithm is it can recognize more than 1 person in the frame depending on the lighting and image condition. After that the recognized person are marked present in the excel sheet as well as the database.

How to use the software

Step 1: Installation

Install the setup in the system. Allow the setup to be installed. A wizard opens. Choose the default download location and click Next. Continue the installation by clicking Next and at the end click Finish to close the wizard. After installation, a folder would appear in that folder one text file is present named dependencies. Open that text file and install 'python' software or one can download on its own from internet, any 3.x version would work. While installing python, choose the option of manual installation and then choose the option to set up the python in system environment. After installation is complete, follow the indexing and install the rest of dependencies. To install the dependencies open "command prompt". To open command prompt press "Window + S", Window Search will be open then type "cmd" in search box and choose option "run as administrator" from right menu shown up after searching. After opening "command prompt", Run setup.py by typing "py setup.py". Make sure system is connected to internet.

Step 2: Run the application(.exe)

After successful installation, a folder is created and inside the folder, run the application file(.exe). A UI opens.

Step 3: Capturing the dataset

In order to enroll a student into the database, click the capture dataset option that appears on the UI. The system runs a program which captures the facial images. Make sure the lighting is fair and your face is in front of the camera so that the system can capture and train the images properly.

Step 4: Training the dataset

After capturing the images, they need to be trained by applying face recognition algorithms like LBPH. This is performed by the train dataset option.

Step 5: Mark the attendance

Click the mark attendance option. The camera opens. Position your face in front of the camera and make sure the lighting is good.

Step 6: Attendance Sheet

To check if your attendance has been marked, click the attendance sheet option, an excel sheet opens with three fields-name, ID and attendance. The faces which were recognized by the system in the previous step are marked present, with the rest marked as absent.

Step 7: Exit

After the completion of the process of marking the attendance, click Exit to close the application.