Efficient Attendance Management System using Real-Time Face Recognition with OpenCV

Abstract:

The traditional manual attendance management system is time-consuming and prone to errors. Therefore, an efficient attendance management system that utilizes real-time face recognition using OpenCV is proposed in this report. The proposed system captures the image of the individual and processes it using OpenCV to identify the person in real-time. The system compares the image of the individual with the images stored in the MongoDB database and records the attendance of the individual if the person is recognized. The system is capable of handling large amounts of data and can be easily integrated into existing attendance management systems. This will eliminate the need for manual attendance recording also it eliminates the possibility of errors that may occur during manual attendance recording.

Hardware requirements:

- Computer with a multi-core processor (i5 or better)
- Webcam with HD resolution or better
- Sufficient RAM (8GB or more)

Software requirements:

- Operating system: Windows, Linux or Mac OS
- Python (3.7 or higher) installed on the system
- OpenCV library installed on the system
- Face recognition libraries such as dlib, face_recognition, etc.
- IDEs such as PyCharm, Anaconda, Jupyter Notebook or any other preferred IDE
- MongoDB database : For storing images and attendance report

Diagram:

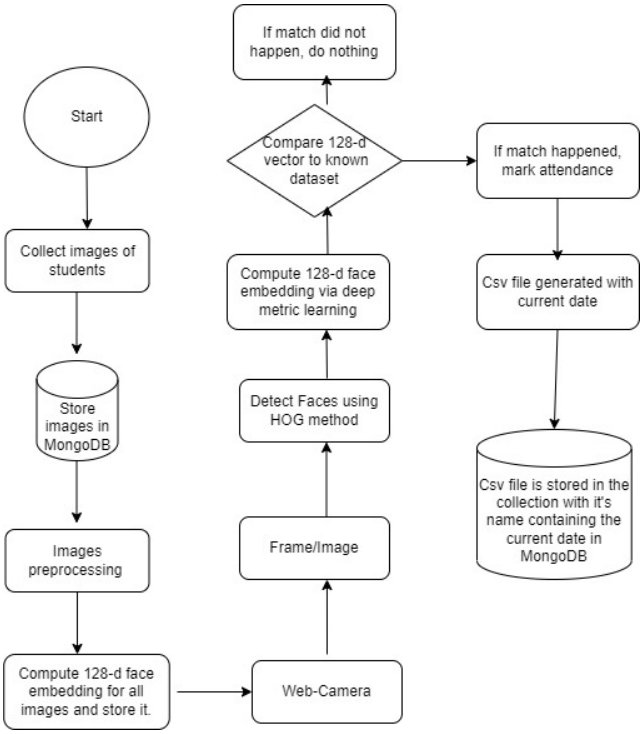

Figure 1: It represents the block or flowchart diagram of the complete functionality of the project.

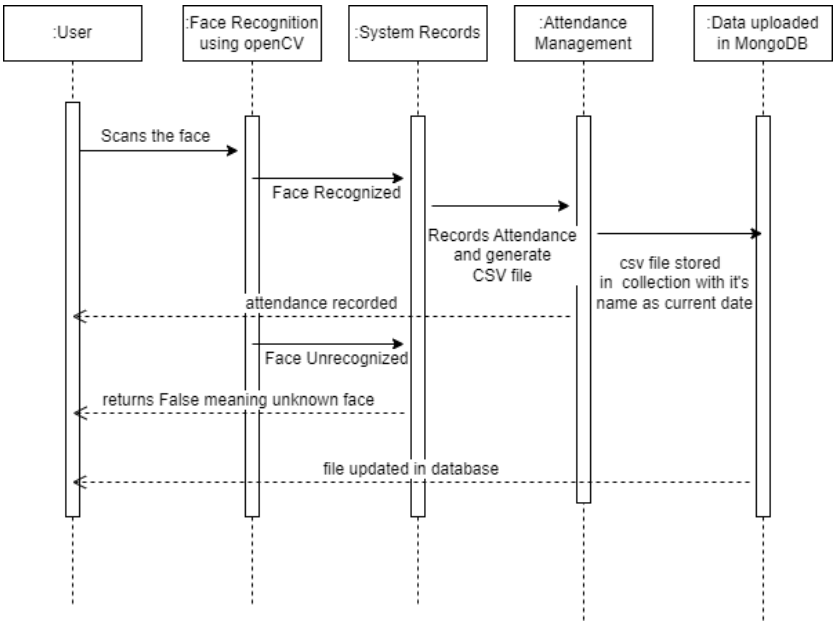

Figure 2: It represents the sequence diagram of the project

Working principle of Face Recognition or techniques used to recognize face :

Step 1: Finding all the Faces – Here in order to detect faces from frame/image Histogram of Oriented Gradients (HOG) method is used. This involves dividing the image into small cells, computing the gradient direction and magnitude within each cell, grouping the cells into larger blocks, and normalizing the gradients within each block to obtain a descriptor vector. By sliding the detection window over the entire image, all regions containing faces can be identified for further processing and recognition.

Step2: Posing and Projecting faces – Here, the faces localized from first step are aligned and projected into a common space for feature extraction and comparison. Here an algorithm called **face landmark estimation is used to determine 68 specific points that exists on face.** This involves first estimating the pose of each face, which refers to its orientation and position relative to the camera. The face is then transformed to a normalized position using affine transformations, such as scaling, rotation, and translation. Next, the face is projected into a common space using techniques such as Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). This results in a feature vector for each face that captures its unique characteristics in a standardized manner.

Step3: Encoding Faces - It involves extracting unique features from an input image of a face and converting them into a mathematical representation, typically using convolutional neural networks which learns to reliably generate 128 measurements for each person. This mathematical representation is known as a face embedding and represents the essential features of a face in a lower-dimensional feature space. The face embeddings can be compared using distance metrics to determine if two faces belong to the same person or not.

Step 4: Finding the person's name from the encoding – In this last step ,we train any ML classifier like linear SVM classifier that can take in the measurements from a new test image and tells which known person is the closest match . The result of the classifier is the name of the person.

How Data or images are stored in backend MongoDB database:

In order to store images in MongoDB, GridFS method is used. It divides the file into smaller chunks and stores them as separate documents. GridFS uses two collections: one for storing the file chunks and another for storing metadata associated with the file, such as filename, content type, etc. GridFS is useful when working with larger images that exceed the document size limit that is 16MB.

GridFS stores files in two collections:

- chunks stores the binary chunks.
- files stores the file's metadata.

Chunk's Collection:

Documents in this collection have the following form:

```
{
  "_id" : <ObjectId>,
  "files_id" : <ObjectId>,
  "n" : <num>,
  "data" : <binary>
}
```

A document from the chunks collection contains the following fields:

chunks._id : The unique ObjectId of the chunk.

chunks.files_id : The _id of the "parent" document, as specified in the files collection.

chunks.n : The sequence number of the chunk. GridFS numbers all chunks, starting with 0.

chunks.data : The chunk's payload as a BSON Binary type.

File's Collection:

Each document in the files collection represents a file in GridFS.

```
{
  "_id" : <ObjectId>,
  "length" : <num>,
  "chunkSize" : <num>,
  "uploadDate" : <timestamp>,
  "md5" : <hash>,
  "filename" : <string>,
  "contentType" : <string>,
  "aliases" : <string array>,
  "metadata" : <any>,
}
```

Why MongoDB is chosen as backend for my project and not Relational Database (RDB) ?

Below is the table which shows the comparison between RDB and MongoDB for storing and retrieving images :

Criteria	Relational Database (RDB)	MongoDB
Data structure	Structured data in tables with predefined schemas	Unstructured and semi-structured data
Image storage	Images need to be encoded as binary data and stored in BLOB or LONGBLOB columns	Images can be stored as binary data directly within a document
Retrieval performance	Retrieving images can be slow and inefficient due to the need for encoding and decoding	Efficient storage and retrieval of large files such as images using GridFS

Scalability	Limited scalability due to the need for predefined schema and strict data consistency	Highly scalable with automatic sharding and flexible schema
Querying	SQL-based queries with joins and relationships	Querying is based on document structure and supports a variety of operators
Use cases	Suitable for transactional systems that require strict data consistency and integrity	Suitable for systems that handle large amounts of unstructured data, such as image and video sharing platforms

MongoDB connection using python :

```
import face_recognition
import cv2
import numpy as np
import csv
import os
from datetime import datetime
from pymongo import MongoClient
from gridfs import GridFS
from PIL import Image
import pandas as pd
```

```
Connection_string="mongodb://localhost:27017"
client=MongoClient(Connection_string)
client.drop_database('online_attendance')
```

```
dbname=client['online_attendance']
fs = GridFS(dbname,collection='students_pics')
```

Snapshots of the data stored in MongoDB on executing the project:

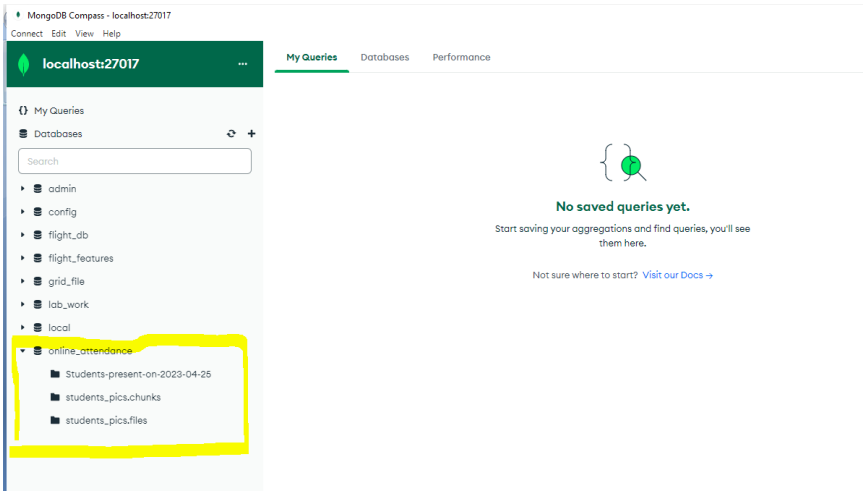

Figure 3: This represents the database “online_attendance” and 3 collections created inside it.

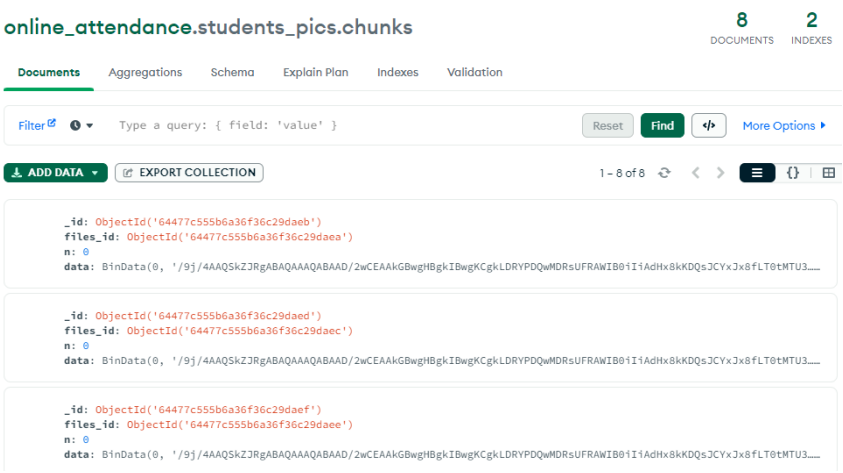

Figure 4: This represents the collection “online_attendance.student_pics.chunks” .

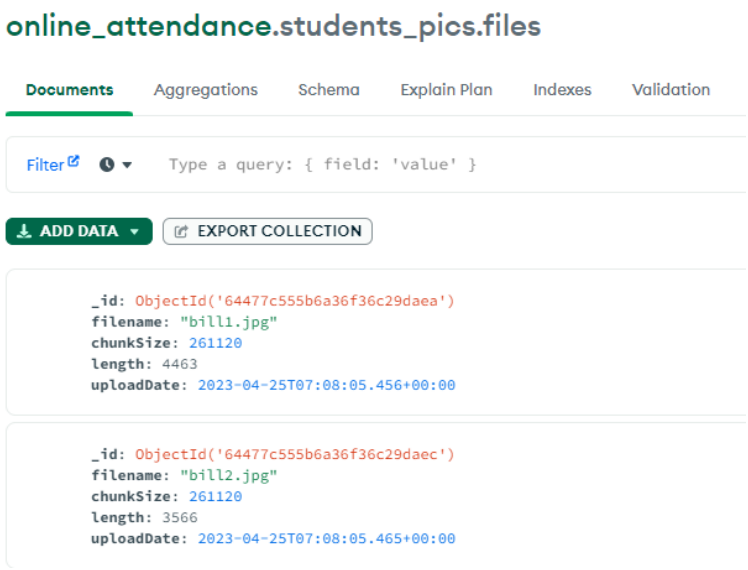

Figure 5: This represents the collection “online_attendance.students_pics.files” .

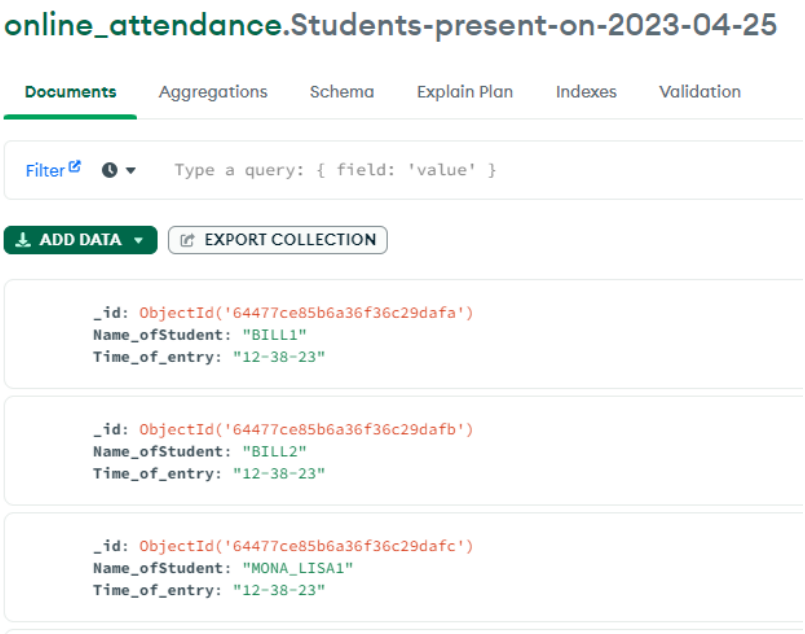

Figure 6: This represents the collection “online_attendance.Students-present-on-2023-04-25” where a new collection with current date is created containing the attendance report of students present on that particular date.