

**School of Computer Science and Engineering**

#### J Component report

**Programme : M.Tech(Int)**

**Course Title : Digital Image Processing**

**Course Code : SWE1010**

**Slot : E2**

**Title : Face Recognition Based Attendance**

**Team Members : HARIGANESH M – 21MIS1004**

**SANTHOSHKUMAR M – 21MIS1023**

**Faculty: Dr.D.Saraswathi Sign:**

**Date: 21/07/2023**

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| S.No | Title | PAGENO |
| 1 | Abstract | 3 |
| 2 | Problem statement | 3 |
| 3 | Objectives | 4 |
| 4 | Introduction | 6 |
| 5 | Literature review | 7 |
| 6 | System Architecture | 16 |
| 7 | Module Description | 17 |
| 8 | Sample code |  |
| 9 | Results and Discussion |  |
| 10 | Conclusion |  |
| 11 | References |  |

# **Face Recognition Based Attendance**

**Abstract**

The provided Python project aims to implement face recognition and attendance tracking using a webcam. It consists of three main parts: "Add face," "app face," and "test face." The "Add face" script captures facial images from the webcam, creating a dataset for training the face recognition model. The "app face" script generates a Streamlit web app that refreshes automatically, displaying counting results (FizzBuzz) and an attendance table with recognized faces and timestamps. The "test face" script utilizes the K-Nearest Neighbors algorithm for real-time face recognition and updating attendance records. The project offers an accessible and efficient system for managing attendance through facial recognition.

**Introduction**

The provided Python code presents a simple yet effective face recognition and attendance tracking system. It consists of three major components: "Add face," "App face," and "Test face." In the "Add face" script, the code captures facial images through the computer's webcam, creates a dataset of faces, and stores them in pickle files for future use. The "App face" script builds a Streamlit web application with auto-refresh capabilities, featuring a counter and a dynamic attendance table. Lastly, the "Test face" script implements real-time face recognition using the K-Nearest Neighbors algorithm to match detected faces with the stored dataset. Once recognized, the system updates the attendance record with the individual's name and timestamp, which is then saved to a CSV file. This system offers a user-friendly and efficient solution for automated face recognition and attendance management, making it applicable in various settings like classrooms, offices, or events.

# **Project Background**

The provided Python code implements a face recognition and attendance tracking system, which serves as a helpful tool for various applications, such as automated attendance management in schools, offices, or events. The main objective of this system is to accurately detect and recognize human faces in real-time using the webcam. The "Add face" script allows users to add their faces to a dataset for later recognition. The "App face" script creates a Streamlit web application with dynamic features to display a counter and an attendance table. The "test face" script utilizes the K-Nearest Neighbors algorithm to recognize faces by matching them against the stored dataset. When a face is recognized, the system records the person's name and timestamp in the attendance record, which is then saved to a CSV file. Overall, this system provides a user-friendly and efficient solution for automated face recognition and attendance management, making it a valuable tool in diverse settings.

# **Problem Statement**

The problem is to develop a real-time face recognition and attendance management system using computer vision techniques and machine learning algorithms. The system should be able to capture face images from the webcam, detect and crop faces using Haar cascades, and save these images along with corresponding labels. Then, a K-Nearest Neighbors (KNN) model should be trained on the collected face data. The system should continuously process live video feed, perform face recognition using the trained KNN model, and display the recognized person's name on the screen in real-time. Additionally, the system should allow users to record attendance by pressing a specific key, which logs the recognized person's name and timestamp in a CSV file named "Attendance\_" followed by the current date. Finally, a Streamlit web application should be created to display the attendance data in a tabular format for the current date.

# **Algorithms**

**Haar Cascades for Face Detection**

Haar cascades are a machine learning-based object detection technique used to find specific objects, such as faces, in images or video frames. The Haar cascades work by looking for simple patterns of image features called Haar-like features, which resemble rectangles with different intensity values. These features are used to identify regions in the image that potentially contain the target object (faces in our case). The algorithm then moves these features across the image and checks for matches using a pre-trained classifier. When it finds a match, it detects and marks the region as a face. In our project, Haar cascades are used to detect faces from the live video feed captured by the webcam.

1. **Nearest Neighbors (KNN) for Face Recognition**

K-Nearest Neighbors (KNN) is a simple and effective machine learning algorithm used for classification tasks, such as face recognition. In the context of our project, KNN is used to recognize the identity of a person based on their face. To achieve this, the algorithm is trained on a dataset of face images along with their corresponding labels (names of the people). When a new face is captured and detected, the KNN algorithm finds the 'k' closest faces in the training dataset to the new face based on some similarity metric (usually distance in feature space). It then assigns the label of the majority of these 'k' neighbors to the new face, considering it as the recognized identity. In our project, KNN is employed to recognize individuals based on the previously captured and labeled face images stored in the data directory.

**Data Collection and Preprocessing Steps:**

1. **Face Image Capture:** The process starts by capturing face images from the webcam in real-time. The cv2.VideoCapture function is used to access the webcam, and a loop is set up to continuously read frames from the video stream.
2. **Face Detection:** For each frame, the Haar cascade classifier is applied to detect faces. The cv2.CascadeClassifier loads the pre-trained Haar cascades XML file, which contains the features required for face detection. The detectMultiScale function is used to detect multiple faces in the grayscale version of the frame.
3. **Crop and Resize Faces:** When a face is detected, the code marks a rectangle around the face in the original frame and then crops the region containing the face. The cropped face image is resized to a fixed size of 50x50 pixels using cv2.resize. This step ensures that all face images have the same size for consistent training.
4. **Data Collection:** The resized face images are collected into an array called faces\_data. In the provided code, up to 100 face images are collected for each person. The data collection loop continues until either 100 images are collected for the person or the user presses the 'q' key to stop the collection.
5. **Labeling Faces:** During data collection, the user is prompted to enter their name. The name is used as the label for the collected face images. For each face image collected, the corresponding label (name) is recorded.
6. **Data Storage:** The collected face images and corresponding labels are stored in separate pickled files. The face images are saved in a numpy array called faces\_data, and the labels are saved in a list called names. These files will be used later for training the KNN algorithm.
7. **Data Preprocessing:** Before training the KNN algorithm, the face images in faces\_data are converted into feature vectors. Each face image is flattened into a 1D array to represent a single feature vector. This process is done using the flatten method and reshaping the data to have dimensions (100, -1) to ensure each image has the same size.

At this point, the data collection and preprocessing steps are complete. The Haar cascades were used to detect faces in the webcam frames, and the collected face images, along with their corresponding labels, are saved for training the KNN algorithm. The KNN algorithm will use this preprocessed data to recognize faces in the real-time webcam feed and perform attendance logging when necessary.