

AI-Based Face Recognition Attendance System

KHUSHI RAJ, NANDITHA S D, AKHIRANANDAN AND DEBRA CHEPKEMOI

Department of Civil Engineering, FET, JAIN (Deemed-to-be University),
Bengaluru-562112, Karnataka, India

Abstract

This paper presents an AI-powered face recognition attendance system that automates attendance taking using real-time facial detection and recognition. The system utilizes deep learning (a CNN-based face model) to detect faces from a live video stream, encode them into numerical embeddings, and match them against a database of known students. Attendance is recorded with subject, date, and time, while preventing duplicate entries per subject per day. The design ensures high accuracy and efficiency, reducing manual workload and eliminating proxy attendance. Key components include data collection, face encoding, recognition, and attendance logging. Performance is evaluated using metrics such as accuracy, precision, recall, and processing speed.

Privacy and ethical considerations are discussed, and a comprehensive implementation plan is outlined for deployment.

Keywords: Face Recognition, Attendance System, CNN, OpenCV, Automation, Biometrics

I. Introduction

Attendance management is a critical administrative function in education, traditionally handled by manual registers or RFID scanners. These methods are time-consuming and vulnerable to fraud (e.g. proxy attendance). With advances in artificial intelligence and computer vision, face recognition has emerged as a promising solution. State-of-the-art face recognition systems (e.g. FaceNet[1], ArcFace[2], DeepFace[3]) achieve accuracy comparable to humans on challenging benchmarks. This project leverages such techniques to design an automated attendance system. The goal is a contactless, real-time system that identifies students via their faces and logs their attendance accurately, thus enhancing efficiency and security in classrooms.

II. Literature Review

Prior work includes biometric attendance using fingerprints or RFID, which require physical contact or tokens. Face recognition systems have been developed, for example by Suresh *et al.* [1] using Python and OpenCV, and Jadhav *et al.* [2] using deep learning approaches. Research shows that deep CNNs trained on large datasets (e.g. FaceNet[1]) produce robust facial embeddings. ArcFace[2] introduced an angular-margin loss that further improved recognition accuracy. Face_recognition, a Python library by Geitgey, wraps the dlib toolkit and provides an easy API for face detection and encoding (128-dimensional embeddings[4]). OpenCV also offers classical methods (Haar cascades[5]) for face detection. Combining these advances, modern systems can reliably detect and recognize faces under varied conditions. However, practical deployment for attendance requires addressing issues like lighting, pose variation, and data privacy, as noted in recent surveys and ethical discussions[6][7].

III. Methodology

The attendance system follows a multi-stage pipeline:

- i. **Data Collection:** Capture facial images of each student using a webcam. Multiple images per student (under different lighting and poses) are stored in a structured dataset.
- ii. **Face Encoding:** For each collected image, detect the face region and compute a numerical encoding using a pretrained face recognition CNN. The result is a 128-dimensional feature vector representing that face[4].
- iii. **Recognition:** In real time, webcam frames are processed to detect faces. Each detected face is encoded into a 128-D vector and compared against the stored encodings of all registered students. The comparison uses Euclidean distance: if the distance is below a set tolerance (e.g. 0.6), it is considered a match[8].
- iv. **Attendance Logging:** Upon a successful match, the student's name is recorded along with the subject, current date, and time. Duplicate entries are prevented by checking existing records for that student and subject on the same day. This workflow ensures subject-wise attendance tracking and automatically logs timestamps while avoiding duplicates.

IV. System Architecture

Example of a face recognition attendance interface (for illustration). The system architecture consists of sequential modules as shown below:

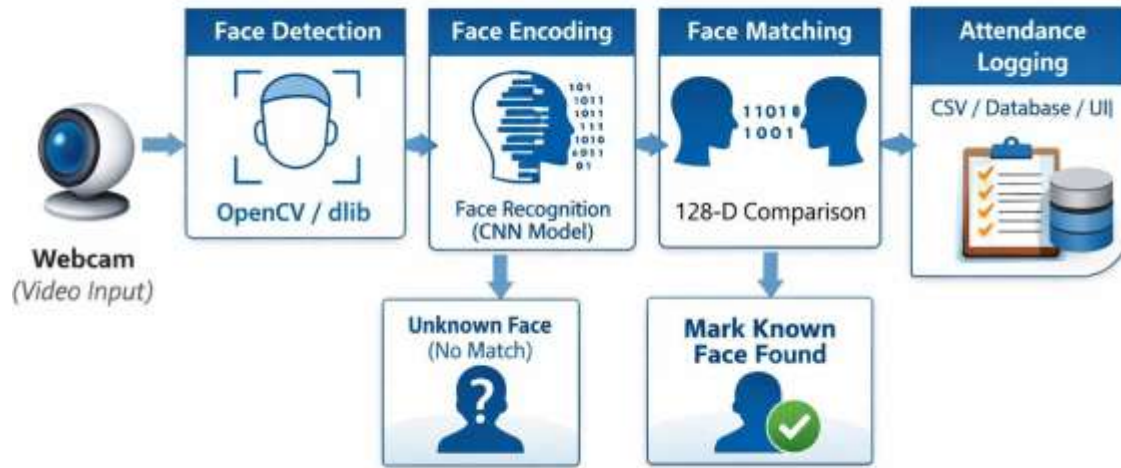
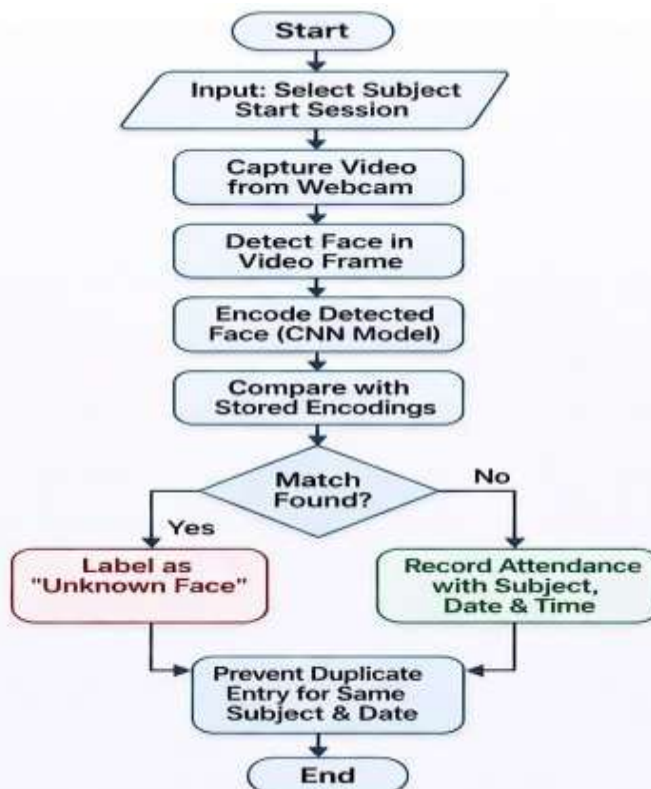


Figure 1

The system follows a pipeline architecture consisting of input, processing, and output stages.

- Input is captured through a webcam
- Faces are detected using OpenCV
- Features are encoded using a CNN model
- Matching is performed using feature comparison
- Attendance is recorded in a CSV file or database

The system also differentiates between known and unknown faces, ensuring only registered users are marked present.



Each step operates in sequence: the webcam feed is processed by OpenCV/dlib to find faces, then a CNN (ResNet-

based) model encodes the face, and matching logic determines identity. Known faces result in attendance marking, while unknown faces are ignored.

V. Data Collection

Students' facial data are collected in advance. The `collect_data.py` script captures multiple images of each student via webcam. Users input a student's name, and the program saves a series of face images (e.g. 20–50 pictures) to a directory `dataset/[Name]/`. Images should include slight variations in angle and lighting to improve robustness. No public dataset is used; each class/institution collects its own data. *Assumption*: No specific constraint on dataset size, but more images typically yield better accuracy.

VI. Preprocessing & Face Encoding

Captured images undergo face detection and alignment. The system uses the `face_recognition` library (which wraps `dlib`) for detection. It offers two modes: a fast HOG-based detector (CPU-friendly) and a CNN-based detector (more accurate, GPU-accelerated)[9]. Detected face regions are cropped (and optionally aligned to a canonical orientation).

Next, each face is passed to a pretrained face recognition model (`dlib`'s ResNet-34) to compute a **128-dimensional encoding**[4]. This numeric vector uniquely represents the facial features. The library function `face_encodings(image)` returns one or more 128-D arrays. These embeddings are stored (in `encode_faces.py`) along with the associated names (in `encodings.pickle`). In real-time recognition, the same encoding process is applied to live video frames.

A key parameter is the **tolerance** threshold. By default, `face_recognition.compare_faces` uses a tolerance of 0.6 (distance units) for best trade-off[8]. Distances below 0.6 are treated as matches. This threshold determines the False Accept Rate (FAR) and False Reject Rate (FRR).

VII. Face Matching & Thresholding

For each detected face, its encoding is compared to the database of known encodings. The system computes the Euclidean distance between the new encoding and each stored encoding[8]. If the minimum distance ≤ 0.6 , the corresponding student is recognized. If no match is close enough, the face is labeled "Unknown" and not marked.

Once identified, the student's name and subject are written to `attendance.csv` along with date and time. Before writing, the system checks whether this student has already been marked present for that subject **on the same date** to prevent duplicates. This is typically implemented by reading the CSV (or DB) and verifying no existing entry for (Name, Subject, Date). This duplicate-prevention logic ensures each student is counted only once per subject per day.



Fig. Face Recognition

attendance - Excel (Product Activation Failed)												
FILE HOME INSERT PAGE LAYOUT FORMULAS DATA REVIEW VIEW												
Calibri 11 A+ Wrap Text General												
B I U Bold Italic Underline Font Alignment Number Conditional Formatting												
Clipboard Font Alignment Number												
L18												
A	B	C	D	E	F	G	H	I	J	K	L	M
1	Khushi	24-01-2026	54:18.2									
2	Khushi	Maths	01-04-2026	02:47:18	Present							
3	Khushi	English	09-04-2026	12:22:06	Present							
4												
5												

Fig. Attendance Sheet

VIII. Implementation Overview

The system is implemented in Python with three main scripts:

- **collect_data.py:** Captures images via webcam. *Input:* prompts user for student name. *Process:* Detects face frames and saves multiple images into a folder named after the student. *Output:* a directory dataset/Name/ filled with face images.
- **encode_faces.py:** Processes the collected dataset. *Input:* reads all images under dataset/. *Process:* Detects faces and computes encodings for each image. *Output:* a file encodings.pickle containing lists of 128-D encoding vectors and corresponding names.
- **recognize.py:** The live attendance loop. *Input:* webcam stream and encodings.pickle; user-entered subject name. *Process:* For each video frame, detect faces, encode them, compare with known encodings. If a match is found, check the CSV for duplicates, then append a new attendance record (Name, Subject, Date, Time) to attendance.csv. Also display identified names on the video feed. *Output:* updated attendance.csv file and on-screen results.

A helper script **fix_dlib.py** may be included to resolve common library issues (on some systems).

Files written/used: - encodings.pickle: Binary file of known face embeddings. - attendance.csv: Log of attendance records (Name, Subject, Date, Time). - .gitignore: to exclude large data and pickles.

All scripts use libraries: OpenCV (opencv-python), face_recognition(with dlib), and NumPy. No code is shown here, but the logic flow is as described.

IX. Evaluation Plan

The system will be evaluated using these metrics and experiments:

- **Accuracy:** $(TP+TN)/(Total)$. Measures overall correctness.
- **Precision & Recall:** Precision = $TP/(TP+FP)$, Recall = $TP/(TP+FN)$ [10][11]. High precision means few false matches; high recall means few missed recognitions.
- **F1-Score:** Harmonic mean of precision and recall[12].
- **False Accept Rate (FAR) & False Reject Rate (FRR):** Standard biometric measures. FAR is the rate at which impostors are incorrectly accepted; FRR is the rate genuine users are not recognized.
- **Latency:** Time per face/frame processing (ms).
- **Throughput:** Frames per second (fps) processed.

Planned experiments:

Experiment	Dataset/Setup	Metrics	Expected Outcome
1. Baseline Recognition	30–50 students × 20 images each	Accuracy, Precision, Recall	High accuracy (~90–95% on test set)
2. Lighting Variation	Same subjects under poor/bright lighting	Accuracy, FRR	Some accuracy drop under poor lighting
3. Pose Variation	Faces at slight angles or with expressions	Accuracy, FRR	Slight increase in FRR for extreme angles
4. Detector Comparison	Use HOG vs CNN face detection	Detection rate, Latency	CNN improves detection of rotated faces; HOG faster on CPU[9]
5. Threshold Tuning	Vary tolerance 0.4–0.7	FAR vs FRR	Lower tol → lower FAR, higher FRR
6. Multi-face Scenario	Class photos (multiple faces)	Throughput, Accuracy	Recognizes multiple faces; slight fps drop
7. Speed Benchmark	Measure on CPU vs GPU	Latency (ms), FPS	GPU (with CNN model) yields higher FPS than CPU

Each experiment will record the chosen metrics. For example, precision/recall will confirm identification quality, and latency will ensure real-time operation (target >10 fps). The threshold study will identify the optimal tolerance (face_recognition uses 0.6 by default[8]).

X. Experimental Setup

- **Hardware:** No specific constraint. We assume a typical PC (e.g. Intel i5/i7 CPU, optionally an NVIDIA GPU for CNN). Tests can be run on Linux/Windows. GPU (e.g. CUDA) speeds up the CNN detector but is not strictly required (HOG can run on CPU)[9].
- **Software:** Python 3.8+, libraries: opencv-python, face_recognition (built on dlib), numpy, pandas(for CSV handling). Example requirements.txt entries (min versions):

```
opencv-python>=4.5 face_recognition>=1.3 dlib>=19.22 numpy>=1.19 pandas>=1.2
```

- **Dataset:** Create a custom dataset. Aim for 20–50 images per student. Encourage variation: some images in dim light, others in sunlight or fluorescent light; some facing directly, others with slight head tilt. Optionally use data augmentation (rotation $\pm 15^\circ$, brightness adjustment). No external dataset needed, but ensure the dataset reflects real classroom conditions.
 - **Sample Size:** Starting with 30–50 students provides a realistic classroom. Larger scale will require more compute. If no specific constraint, we can assume building for mid-size classes.
 - **Pose/Landmark Data:** Use dlib's 5-point or 68-point landmarks internally (handled by the library) to align faces for encoding.
 - **Software Tools:** OpenCV's CascadeClassifier (Haar) can be used for initial detection; dlib's CNN detector for better accuracy if GPU available[9].
 - **Performance Testing:** Measure on this hardware and dataset. Report fps and memory usage.
- No extreme edge-case constraints were given; we assume normal classroom environment without occlusions (e.g. no masks) for baseline.

XI. Limitations

- **Lighting and Environment:** Recognition accuracy drops in very low or uneven lighting. The system relies on clear frontal faces (occlusions like masks/hats can fail detection).
- **Pose Variation:** Extreme head angles (side profiles) are not reliably recognized. The model is trained mostly on frontal images[9].
- **Dataset Quality:** The system's accuracy heavily depends on collecting good- quality training images. Poor enrollment (blurry, partial faces) can cause errors.
- **Compute Load:** CNN face detection is computationally intensive. On limited hardware, performance may be slow. Using the HOG detector speeds up CPU performance but at some accuracy cost[9].
- **False Positives/Negatives:** No system is perfect; some non-registered faces might occasionally match (low FAR) or registered faces might not match (FN). Tolerance tuning mitigates but does not eliminate these.
- **Scalability:** The CSV approach may become inefficient for very large classes or multiple simultaneous classes. A database is recommended for larger scale.
- **Bias and Fairness:** As with all face models, there is a risk of demographic bias (e.g. by race/gender). Care must be taken to use diverse training data.

XII. Privacy and Ethics

Face recognition involves personal biometric data, so privacy is paramount. In deployment: - **Data Security:** Store encodings securely; no plain images publicly accessible. Use encryption and access controls on the CSV/DB. - **Consent:** Obtain student (or parent) consent before capturing faces. Make attendance usage transparent. - **Regulations:** Comply with relevant laws (e.g. India's data protection regulations or international GDPR-like policies). Laws such as BIPA (Illinois) or CCPA (California) restrict biometric data use[6]. - **Purpose Limitation:** Use face data only for attendance. Do not share or repurpose the data without permission. - **Transparency:** Keep logs of system use. Allow students to review attendance records. - **Mitigation:** Use "privacy by design" practices (e.g. strip identifying

information where possible)[7].

Ethical deployment means balancing security with respect for privacy. All stakeholders should be informed and data handled responsibly.

XIII. Deployment & Future Scope

- **Database Integration:** Replace CSV with a SQL database (MySQL/PostgreSQL). This enables concurrency and queries. *Sample Schema:*

```
CREATE TABLE Students (
student_id INT PRIMARY KEY AUTO_INCREMENT, name VARCHAR(100) NOT NULL
);
CREATE TABLE Attendance (
attendance_id INT PRIMARY KEY AUTO_INCREMENT, student_id INT NOT NULL,
subject VARCHAR(50), datetime DATETIME,
UNIQUE (student_id, subject, DATE(datetime)),
FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
```

The UNIQUE constraint on (student_id, subject, DATE) prevents duplicates per day.

- **Web Dashboard:** Develop a web interface to view attendance records and analytics. Technologies like Flask or Django can display CSV/database data graphically.
- **API Backend:** Use FastAPI or Flask to create a REST API for recognition (e.g. send images, receive ID). This allows mobile or web apps to integrate the system.
- **Role-Based Access:** Implement user accounts (faculty, admin) so that only authorized staff can view/modify attendance.
- **Analytics:** Add features to compute attendance percentages, generate reports, and send notifications for absentees.
- **Cloud Deployment:** Host the system on cloud platforms (AWS/GCP). For example, use an EC2 instance with GPU for recognition and S3/RDS for storage. Cloud scaling can handle many classrooms.
- **Model Updates:** Explore more advanced models (e.g. ArcFace updates, one- shot learning for new students) and real-time face tracking to improve robustness.

These enhancements will evolve the prototype into a production-ready system with full functionality.

XIV. Face Recognition Models Comparison

Model/Library	Accuracy (LFW)	Speed (Inference)	Ease-of- Use	License	Use Case
FaceNet [13] (Google)	99.63% (LFW)	GPU- accelerated (ResNet), moderate on CPU	Medium (TF/Keras)	Apache 2.0	High- precision applications; research; large-scale deployment
ArcFace [2] (InsightFace)	~99.5– 99.8% (LFW)	GPU (ResNet100), heavy	High (PyTorch)	MIT License	State-of-art accuracy; large datasets; industry deployments
DeepFace [3] (Facebook)	97.35% (LFW)	Slow (9-layer net)	Low (not public)	Proprietary	Historical baseline;

Model/Library	Accuracy (LFW)	Speed (Inference)	Ease-of- Use	License	Use Case
					not readily available
face_recognition/dlib[14]	99.38% (LFW, 1-100 jitters)	Fast on CPU (HOG), faster with GPU (CNN)	Very High (Python API)	MIT License	Rapid prototyping; education; small-medium datasets

- *Accuracy:* FaceNet and ArcFace lead; face_recognition's ResNet model is also very high (99.38% with 100 jitters[14]). DeepFace is lower.
- *Speed:* face_recognition's HOG mode is CPU-fast; CNN mode needs GPU. FaceNet/ArcFace typically use GPU.
- *Ease:* face_recognition (Python) is easiest to use. ArcFace/FaceNet require more setup (TensorFlow/PyTorch).
- *License:* face_recognition/dlib (open), FaceNet (open), ArcFace (open MIT), DeepFace (closed).
- *Use:* face_recognition is great for quick development. FaceNet/ArcFace for maximum accuracy at scale. DeepFace mostly historical.

XV. Evaluation Plan (Tabular)

Experiment	Dataset/Conditions	Metrics	Expected Outcome
Baseline Recognition	Custom dataset (~50 persons, 20 imgs each)	Accuracy, Precision, Recall, F1	High accuracy (>90%); balanced precision & recall
Lighting Variation	Same faces, dim vs bright lighting	Accuracy drop, FRR, FAR	Lower accuracy under dim light; adjust threshold
Pose Variation	Faces with slight left/right tilt	Accuracy, FRR	Minor performance drop; CNN detector helps a bit
Threshold Tuning	Vary tolerance 0.4–0.7 on same data	FAR vs FRR curve	Lower tol → lower FAR (safer), higher tol → lower FRR
Multi-face Scenario	Group photo with 5–10 people	Recognition rate, FPS	Should find all known faces; slight throughput decrease
Detector Comparison	HOG vs CNN face detection (same setup)	FPS, detection rate	HOG: faster but may miss angled faces; CNN: slower, finds more faces[9]
Experiment	Dataset/Conditions	Metrics	Expected Outcome
Throughput Stress Test	Continuous stream, high frame rate (~30fps)	Max FPS, CPU/GPU usage	Measure sustainable frame rate; expect higher FPS on GPU
Database vs CSV Logging	Simulate many entries (>1000)	Insert time, query time	DB scales better (indexed) than CSV append

Each experiment uses defined dataset scenarios and records metrics. Expected results guide fine-tuning (e.g. tolerance selection). Benchmarking the HOG vs CNN detectors will clarify trade-offs[9]. Overall, we expect >90% accuracy with

optimized settings.

XVI. Project Structure & Requirements

Folder Layout:

```
face_attendance_project/
|
|— collect_data.py          # Collect webcam images for each student
|— encode_faces.py         # Compute and save face embeddings
|— recognize.py            # Real-time recognition and attendance logging
|— fix_dlib.py             # Optional dlib patch (Windows fix)
|— encodings.pickle        # (Generated) known face embeddings
|— attendance.csv          # (Generated) attendance log file
|— requirements.txt        # Python dependencies
```

requirements.txt (example versions):

```
opencv-python>=4.5.3 face_recognition>=1.3.0 dlib>=19.22.0 numpy>=1.19.5 pandas>=1.1.5
fastapi>=0.70.0          # if using API backend uvicorn>=0.15.0      # if using API server
```

These packages can be installed via pip.

XVII. Sample SQL Schema

To replace CSV storage, a relational schema could be:

```
CREATE TABLE Students (
student_id INT PRIMARY KEY AUTO_INCREMENT,
name VARCHAR(100) NOT NULL
);
CREATE TABLE Attendance (
attendance_id INT PRIMARY KEY AUTO_INCREMENT, student_id INT NOT NULL,
subject VARCHAR(50) NOT NULL,
datetime DATETIME NOT NULL, UNIQUE(student_id, subject, DATE(datetime)),
FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
```

The UNIQUE(student_id, subject, DATE(datetime)) constraint ensures a student is not marked twice for the same subject on the same date (duplicate prevention).

References

- [1] F. Schroff, D. Kalenichenko, J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," *CVPR*, 2015[1][13].
- [2] J. Deng *et al.*, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," *CVPR*, 2019[2].
- [3] Y. Taigman *et al.*, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," *CVPR*, 2014[3].
- [4] "face_recognition: Recognize Faces in Images," Python library documentation (Adam Geitgey)[4][9].
- [5] OpenCV Documentation: Face Detection using Haar Cascades[5].
- [6] V. Suresh *et al.*, "Facial Recognition Attendance System Using Python and OpenCV," *Journal of Software Eng. & Simulation*, vol.5, no.2, 2019.
- [7] R. Jadhav *et al.*, "AI-Powered Facial Recognition Attendance System Using Deep Learning and CV," *IJRSL*, vol.XII, no.IX, 2025.
- [8] S. Smitha, P.S. Hegde, A. Rashid, *Face Recognition based Attendance Management System*, Yenepoya Inst. Tech., 2016.