Real-Time CNN-Based Face Recognition Platform for Scalable Identity Management and Alert Transmission

[1]A.Harsha Vardhan Reddy, [2]A.Saiprashanth, [3]S.Praneeth Reddy, [4]Ch. Raja
[1,2,3] Department of ECE, Mahatma Gandhi Institute of Technology (A), Gandipet, Hyderabad, Telangana, India. Email: anthireddyh@gmail.com
[4] Associate Professor, Department of ECE, Mahatma Gandhi Institute of Technology (A), Gandipet, Hyderabad, Telangana, India. Email: chraja@mgit.ac.in

---***---

Abstract - This paper presents a comprehensive, cloud-integrated face recognition-based attendance system that combines machine learning, web technologies, and scalable backend architecture. The backend server, built with Node.js and MongoDB, securely stores user data, embeddings, and supports authentication via JWT and RBAC. The core recognition pipeline uses MTCNN for face detection, FaceNet for embedding generation, and SVM for classification. Recognized faces trigger SMS alerts via Twilio, with optional email notifications. A React frontend allows image uploads, result viewing, and user management, while a Flask layer bridges UI and ML logic. The system is deployed on AWS using Docker, ECS, and MongoDB Atlas, with CloudFront enhancing frontend delivery. CI/CD pipelines automate testing and deployment, while CloudWatch and Prometheus ensure robust monitoring. Auto-retraining of the SVM model accommodates new users, and comprehensive backup strategies with multi-zone redundancy ensure disaster recovery. Security enhancements such as hashed passwords, rate limiting, CORS policies, and encrypted cloud storage further strengthen system resilience. Designed for scalability, the platform auto-scales backend containers, caches frequent queries using Redis, and employs load balancing for high availability. This modular, end-to-end solution provides a robust framework for real-time, secure, and scalable facial recognition applications across educational or enterprise environments.

***Key Words*:** Automatic Attendance System, Convolutional Neural Networks (CNNs), Facial Recognition, MTCNN, Smart Attendance Monitoring, FaceNet.

1.INTRODUCTION

Attendance monitoring systems are vital in schools, workplaces and events for effective tracking of individuals. Although Convolutional Neural Networks (CNNs) provide high accuracy in facial recognition for identifying individuals, identification of multiple faces in a crowd is challenging [1]. This paper seeks to transform attendance tracking by using advanced facial recognition to automate the process with high accuracy, thus minimizing administrative workload and enhancing efficiency. It highlights the need for administrator-friendly interfaces and paves the way for more general applications in security, access control and surveillance [2].

This paper describes a methodical way of creating a CNN- grounded intelligent attendance system. It starts with a literature review of attendance systems and CNN, followed by an elaborate methodology that includes data collection, preprocessing, designing the CNN model and svm model, training and confirmation of them. The results are presented through an intertwined homepage. The system generates an Excel distance; this distance can be reviewed directly from the interface. The system is tested on different scripts to check its robustness. The conclusion presents results, crucial findings, practical counteraccusations and motifs of unborn exploration for perfecting automated attendance systems[3].

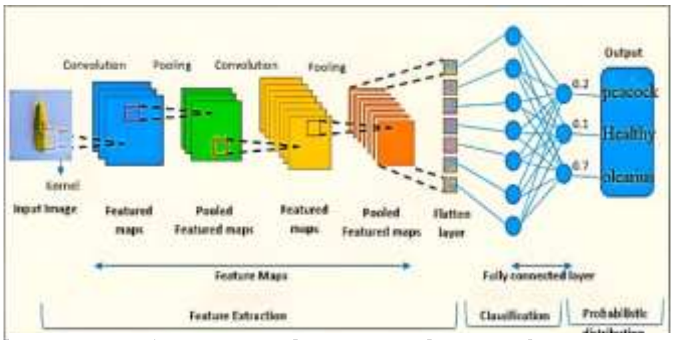


Fig -1 : Convolution Neural Networks

Figure 1 illustrates a Convolutional Neural Network (CNN) for image processing. The input image passes through convolution layers, where kernels extract features into feature maps, followed by pooling to reduce dimensionality. This process repeats, capturing hierarchical patterns. The flatten layer converts maps to 1D data for fully connected layers, which classify the image via probabilistic distribution. CNNs excel in tasks like object recognition by automating feature extraction.

In summary, the creation of a smart attendance shadowing system grounded on Convolutional Neural Networks(CNNs) is an important advance in attendance shadowing. Exercising colorful facial recognition algorithms, the system offers an extremely automated and precise result that can be espoused by educational institutions, workplaces and event organizers. The successional methodology from data collection and preprocessing to model training and testing guarantees the system's trustability and robustness. The donation of this work is in its capability to explain operations, improve productivity and drop executive burden, hence encouraging better attendance operation practices. Unborn exploration and development in this area can promise indeed more advanced uses of facial recognition technology, further boosting continuing invention in smart attendance covering systems[4].

2.LITERATURE SURVEY

A. A Counterpart Approach to Attendance and Feedback System using Machine Learning Techniques:

This paper implements the concept of two technologies Student Attendance and Feedback System using a machine learning approach. The system automatically monitors student performance and maintains records such as attendance and feedback on subjects like mathematics and others. Consequently, student attendance is captured through facial recognition. Upon successful recognition, the system retrieves both attendance records and academic performance details as feedback[5].

B. Automated Attendance System Using Face Recognition:

Automated Attendance System based on Face Recognition suggests that the system relies on face detection and recognition algorithms, which is utilized to automatically detect the student face upon his/her entrance into the class and the system is able to mark the attendance by identifying him. Viola-Jones Algorithm has been implemented for face detection which detects the face by using cascade classifier and PCA algorithm for feature extraction and SVM for classification. Compared to the conventional marking attendance this system provides a saving of time as well as monitors the students[5].

C. Face Recognition-based Lecture Attendance System:

This paper suggests that the system is provided with the attendance automatically recognition achieved through continuous observation. Continuous observation assists in estimating and enhancing the performance of the attendance. For the purpose of acquiring the attendance, positions and face images of the students within the classroom are captured. Through continuous observation and recording, the system estimates seating position and location of each student for attendance marking. The task is centered on how to achieve the various weights of every focused seat based on where it is. The efficiency of the image is also under debate to allow the quicker identification of the image[5].

D. Fingerprint Based recognition system:

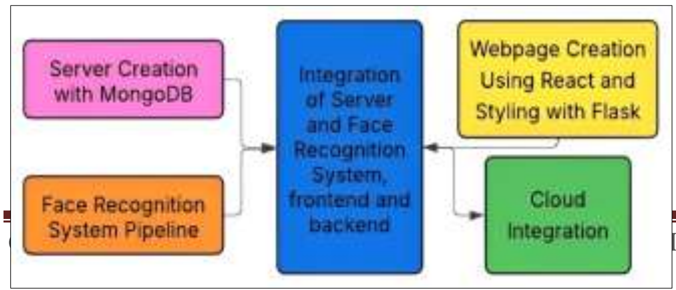
In the existing fingerprint-based attendance system, a portable fingerprint device must first be configured with each student's fingerprint. To mark attendance, students are required to scan their fingerprint on the device either during or before lecture hours. However, this approach can be disruptive, as recording fingerprints during class may divert students' attention from the lecture.

E. RFID(Radio Frequency Identification) Based recognition system:

In the existing RFID-based system, students are required to carry a Radio Frequency Identification (RFID) card and place it on a card reader to register their attendance for the day. The system can connect to an RS232 interface and store the attendance data in a database. However, this method is vulnerable to fraudulent activity, as students may use someone else’s ID card to mark attendance for an absent peer or misuse the system in other ways[7].

F. Iris Based Recognition System:

In the iris-based student attendance system, students stand in front of a camera, which scans their iris for identification. The scanned iris is compared with the student’s data in the database, and their attendance is automatically marked. This system eliminates the paper-and-pen process, reducing faculty workload. It also reduces the risk of proxy attendance and ensures safe storage of student records. As a wireless biometric method, it tackles fraudulent attendance while avoiding complex networking infrastructure.[8]


Server Creation with MongoDB
Face Recognition System Pipeline
Integration of Server and Face Recognition System, frontend and backend
Webpage Creation Using React and Styling with Flask
Cloud Integration

3.PROPOSED SYSTEM

Figure 2 illustrates the complete architecture of the face recognition system, structured into five major blocks. Each block represents a critical component in the development pipeline, starting from server creation and moving through

Fig -2 : Block diagram of face recognition system

recognition, integration, frontend, and cloud deployment. This modular structure ensures a scalable, secure, and user-friendly face recognition solution.

A. Server Creation with MongoDB (Backend)

Step 1: Setting Up the Server

- Initialize Node.js Server: Run npm init to create a package.json file for managing server dependencies and metadata.
- Install Core Dependencies: Install express for setting up HTTP routes, mongoose for MongoDB interaction, dotenv for environment variables and body-parser for parsing incoming request payloads.
- Directory Structure: Create folders for organizing code: /models (Mongoose schemas), /routes (API endpoints), /controllers (business logic) and /middleware (security layers).

Step 2: Environment Configuration

- .env File: Store sensitive configurations.
  - MONGODB_URI for the database connection URL and JWT_SECRET for signing authentication tokens.
  - TWILIO_SID and TWILIO_AUTH_TOKEN for en abling SMS alerts in the application.

Step 3: Connecting to MongoDB

- Mongoose Setup: Use mongoose.connect (process.env.MONGODB_URI) to establish a connection to MongoDB and log success/failure messages for debugging.

Step 4: Defining User Schema

- Fields
  - Include username (for unique identification), password (hashed with bcrypt for security) and role (for distinguishing faculty and students).
  - Add embeddings (for storing 128D face vectors) and createdAt (for tracking user registration timestamps).

Step 5: Building REST API Routes

- Key Endpoints
  - POST /add for storing user data and face embeddings in the database.
  - GET /users for retrieving all user records, and POST /login for authenticating users and issuing JWT tokens.

Step 6: Running the Server

- Start Server: Execute app.listen(5000) to launch the server on port 5000 and verify its status.

Step 7: Security Enhancements

- JWT Authentication: Implement token verification for protecting routes like /add and /users.
- Password Hashing: Use bcrypt to hash passwords and prevent plaintext storage.
- Rate Limiting: Apply express-rate-limit to throttle

repeated requests and deter brute-force attacks.
- CORS & Validation: Configure CORS for restricting API access to trusted domains and sanitize inputs with express validator.
- Role-Based Access (RBAC): Assign roles (e.g., faculty or student) and restrict endpoints based on permissions.
- Session Timeout: Set token expiration to enforce periodic re-authentication.

B. Face Recognition System Pipeline

This figure illustrates the process of face recognition-based attendance using MTCNN and FaceNet. The system begins by detecting faces from a dataset using MTCNN, then extracts embeddings with FaceNet and trains a model using the SVM algorithm. During testing, the trained model classifies input images to mark attendance.

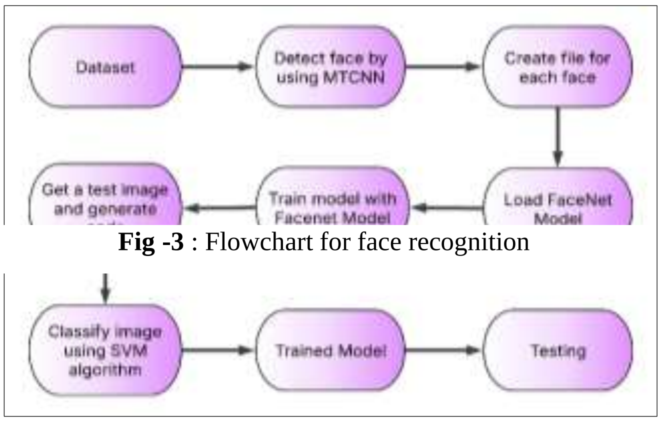

Dataset
Detect face by using MTCNN
Create file for each face
Load FaceNet Model
Train model with Facenet Model
Get a test image and generate
Classify image using SVM algorithm
Trained Model
Testing

Fig -3 : Flowchart for face recognition

Step 1: Dataset Preparation
- Image Collection
  - Collect labeled images of individuals. Organize the images into folders, where each folder represents a unique individual (e.g., PersonA/, PersonB/).
  - Ensure each folder contains multiple images of that individual from different angles and lighting conditions to improve the model’s generalization.
- Image Augmentation
  - Random rotations: Rotate images to simulate various orientations.
  - Scaling: Resize images to simulate different distances.
  - Flipping: Flip images horizontally to simulate different perspectives.

These techniques help increase the dataset's diversity, making the model more robust.
Step 2: Face Detection Using MTCNN
- MTCNN (Multi-task Cascaded Convolutional Networks) detects faces in images and generates bounding boxes.
- After detection, the faces are aligned (i.e., normalized to a standard position and scale) to ensure consistency in facial features before passing them to the next stage.

Step 3: Face File Storage
- After cropping the faces from the original images, store them in a structured format, such as name_timestamp.jpg, to maintain a connection between the image and the individual’s identity.

This ensures you can track which face belongs to which person, making the data easy to manage.
Step 4: Generating Embeddings Using FaceNet
- Pass each detected and aligned face through FaceNet to generate a 128-dimensional embedding.
- The embedding is a numerical representation of the unique features of the face, making it possible to compare different faces and measure their similarity.

Step 5: Training the Classifier (SVM)
- Use the generated face embeddings to train an SVM classifier.
- The SVM will learn to differentiate individuals by mapping their embeddings to high-dimensional space.
- After training, the classifier can distinguish between faces based on the position of their embeddings.

Step 6: Face Recognition (Prediction Phase)
- Detect the face in a new, unseen image.
- Generate its embedding using the same FaceNet model.
- Compare this new embedding with the stored embeddings using the trained SVM classifier.
- The classifier will help determine the closest match to an existing user.

Step 7: Classifying the New Embedding
- The SVM model takes the new embedding and classifies it, returning the identity (name) of the person.
- The confidence score associated with the classification can also be returned, indicating the probability that the recognition is correct.

Step 8: Model Storage and Management
- Save the trained SVM model and the embeddings to disk. This allows you to reuse the model and embeddings in future sessions without retraining.
- Model storage helps maintain continuity and reduces processing time during subsequent uses of the system.

Step 9: Testing and Validation
- Use a test dataset that was not part of the training process to evaluate the model's performance.
- Measure accuracy, precision and recall assessing how well the model identifies individuals.
- Fine-tune the model if necessary, adjusting parameters to improve performance and reduce errors.

Step 10: Exporting Results to Excel
- After recognition, export the results (e.g., names, confidence score) into an Excel sheet for documentation.
- Columns could include:
  - Name, Roll No.: Identified person.
  - Status: Absent/Present of student.
  - Timestamp: The time the recognition occurred.

This allows for easy analysis and record-keeping.

C. Integration of Server and Face Recognition System

Step 1: Storing Known Face Data (Face Embeddings)
- When a new face is registered (via the POST /add API), its embedding (a 128-dimensional vector) is generated and stored in the MongoDB database.
- This allows the system to maintain a collection of all known faces for comparison during recognition.

Step 2: Accessing Stored Data During Prediction
- When a new face needs to be recognized, the system retrieves all stored embeddings from MongoDB using the GET /users API.
- This step is crucial as it allows the system to compare the new face’s embedding against the stored ones and determine the best match.

Step 3: Real-Time Prediction Flow

1. Image Upload: The user uploads a face image via the frontend.
2. Face Detection: The system detects the face in the image using MTCNN and generates its corresponding embedding.
3. Fetch Data: The server fetches all stored embeddings from MongoDB for comparison.
4. Recognition: The system compares the new embedding with the stored embeddings using the trained SVM classifier to identify the person.
5. Display Result: The predicted identity (with confidence score) is displayed to the user, confirming the recognition.

Step 4: Support for Continuous Learning

- Add New Users: New users can upload their face data, generating embeddings that are stored in MongoDB for future recognition.
- Retrain Model: As new users are added, the SVM model can be retrained to include the new embeddings. This enables the model to recognize additional individuals without needing to rebuild the entire system.

Step 5: System Workflow Summary

- Training Phase:
  Image → MTCNN → FaceNet → Embeddings → Store in MongoDB → Train SVM.
- Recognition Phase:
  Image → MTCNN → FaceNet → Embedding → Compare with DB → Result Display.

Step 6: Notification Enhancements

- Recognition Logging: Every successful face recognition event is logged in the database, including the timestamp, name, confidence score and IP address of the user.
- SMS Notifications via Twilio: When a face is recognized and the confidence score is above a certain threshold (e.g., >90%), an SMS notification is sent to the parent or student using the Twilio API.
- Email Alerts (Optional): Use SMTP services (e.g., Gmail, SendGrid) to send email alerts to admin or faculty in the following cases.
  - Repeated recognition failures.
  - Attendance status also.
  - Suspicious login or access patterns.
- Alert Throttling: To prevent spamming of notifications, implement cooldown logic that ensures no more than one alert is sent per user per 10 minutes.

D. Webpage Creation Using React and Styling with Flask

Step 1: Purpose and Importance of the Web Interface

- The web interface allows users (faculty and students) to interact with the face recognition system.
- Users can upload face images, view recognition results, and manage registered faces.
- It enhances accessibility, making the face recognition system practical and easy to use.

Step 2: Frontend with React

- React Features
  - Image Upload: Users can upload face images via a file input or webcam.
  - Face Preview: Displays the uploaded face image for confirmation before submission.
  - Submit Button: Sends the uploaded image to the backend for recognition.
  - Result Display: Shows the predicted identity and confidence score to the user.
- Technologies Used
  - Axios: Used for making API calls to the backend.
  - React Router: Handles navigation between different pages in the web application.
  - Tailwind CSS or Styled components: Provides responsive styling for a clean user interface.

Step 3: Backend Support Using Flask

- Flask for API Routing: Flask serves as the backend, handling API requests.
  - Face recognition.
  - User registration.
  - Retrieval of stored data.
- Serving Static Assets: Flask serves static assets like images and CSS when integrated with the React frontend, ensuring all assets are correctly loaded for a seamless experience.

Step 4: Twilio API Integration for SMS Notifications

- Twilio API Setup: Store Twilio Account SID and Auth Token securely in environment variables for use in the backend.
- Sending SMS: When a face is recognized, the backend uses the Twilio Node.js library to send an SMS to a predefined phone number (either the student's or parent's phone number).
  - Example messages include: The predicted identity and timestamp for the recognition event.

Step 5: Integration Between Frontend and Backend

- React Frontend: Sends image uploads or requests for user data to Flask APIs.
- Flask Backend:
  - Performs face recognition and Sends results back to the React frontend.
  - If required, sends an SMS notification via Twilio to the designated phone number upon recognition.

Step 6: Deployment Considerations

- Docker: Containerize the Flask backend and React frontend to simplify deployment.
- Nginx: Use Nginx as a reverse proxy and static file server for optimal performance.
- Platform Deployment: Deploy the application on cloud platforms like AWS, Heroku or Render for reliable hosting.

Step 7: Frontend Security Enhancements

- Token Storage: Store JWT tokens in httpOnly cookies for security (prevents XSS attacks).
- Role-based UI Rendering: Display or hide specific UI elements based on the user role (e.g., show admin controls only to faculty).
- CSRF Protection: Implement anti-CSRF tokens for state-changing operations (especially if JWT is stored in cookies).
- Captcha Integration: Add Google reCAPTCHA or other captcha to the login and registration forms to prevent bot attacks.
- Session Timeout Warning: Notify users when their session is about to expire and log them out securely to enhance security.

E. Cloud Integration and Scalability

This block ensures the face recognition system is cloud-native, resilient, and scalable, supporting enterpriselevel deployments. It includes the best practices for DevOps, infrastructure, network management, and performance optimization. It also integrates continuous monitoring and automated recovery mechanisms to maintain high availability and operational reliability under varying workloads.

Step 1: Cloud Platform Integration

- Primary Cloud Platform: While the architecture is designed for AWS, it is adaptable to Google Cloud Platform (GCP) or Azure. This flexibility ensures broader deployment options based on organizational preferences or existing cloud ecosystems.
- Core Services Used:
  - EC2: Hosts containerized services such as the Flask API and recognition pipeline.
  - S3:
    - Stores uploaded face images, processed image logs and Excel exports.
    - Acts as a secure and versioned data lake.
  - AWS Lambda (Optional): Handles lightweight serverless functions like SMS fallback logic, log rotation or post-recognition logging.
  - ECS (Elastic Container Service) / EKS (Elastic Kubernetes Service): Manage container orchestration, supporting autoscaling, load balancing, health checks and blue/green deployments.
  - RDS (Optional): Stores structured logs (e.g., login attempts, recognition metadata) for reporting. Gives best user experience.
  - CloudFront (CDN):
    - Serves the React frontend with low-latency, global access.
    - Supports SSL termination and caching.
- Database Hosting: MongoDB Atlas
  - Built-in sharding for horizontal scaling.
  - Supports IP whitelisting, RBAC and TLS/SSL for secure access.
  - Provides daily snapshots and restore points.
- Security Enhancements in Cloud Layer:
  - Enable VPC peering to isolate databases from public traffic.
  - Apply IAM roles and policies for enforcing the least privilege access.
  - Configure CloudTrail for auditing sensitive actions.

Step 2: Containerization and Orchestration

- Dockerization: Each component is packaged into its own Docker image.
  - Flask API backend
  - Face recognition engine (SVM + MTCNN + FaceNet)
  - React frontend
- Benefits of Dockerization:
  - Environmental-agnostic deployment for dev, staging, and production.
  - Ensures reproducibility and security.
  - Allows for quick rollback with versioned Docker images.
- Docker Compose (for Local Dev): Manages a multi-container setup with volumes, networks and health checks for local development. It enables consistent and replicable development environments, streamlining team collaboration and testing.
- ECS/EKS for Production:
  - Auto-restarts containers on failure.
  - Supports auto-scaling based on CPU/memory/network usage.
  - Facilitates rolling updates and zero-downtime deployments.
- Health Monitoring Probes: Define liveness and readiness probes for each container to ensure fault tolerance and early failure detection.

Step 3: CI/CD Pipelines

- CI Tools: GitHub Actions, GitLab CI, or Jenkins
  - Lint, test, and build Docker images on each commit or pull request.
  - Secrets are injected using encrypted CI secrets or AWS Secrets Manager.
- CD Tools:
  - Push final Docker image to ECR (Elastic Container Registry).
  - Auto-deploy via ECS/EKS using Blue/Green or Canary deployment strategies.
- Security Notes:
  - Never store secrets in code.
  - Validate image integrity using image signing or hash verification.

Step 4: Logging and Monitoring

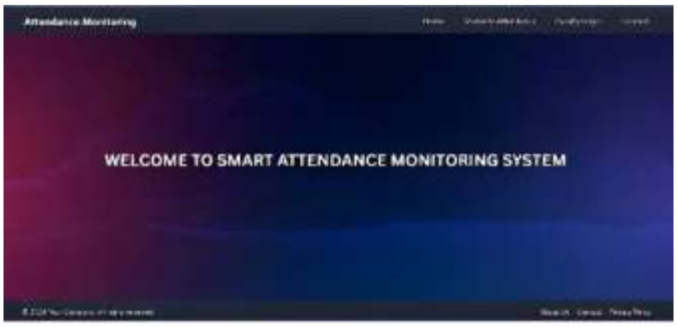
- Centralized Logging: Use AWS CloudWatch Logs or ELK stack to capture logs.
  - Face recognition attempts and results.
  - Twilio SMS delivery statuses.
  - User login logs, token issuance and failures.
- Monitoring Tools:
  - Prometheus for metrics collection and Grafana for dashboard visualization.
  - Key metrics to monitor:
    - Face recognition time (in ms).
    - API error rate.
    - SMS delivery failure rate.
    - JWT token validation failures.
- Alerting Mechanisms: Alerts via SNS, Slack, or PagerDuty.
  - 5XX errors from the API.
  - Breach of memory/CPU thresholds.
  - Excessive failed logins indicating suspicious behavior.

Step 5: Cloud Storage and Backup Strategy

- S3 Buckets: Public access is disabled, and encryption is enabled (SSE-S3 or SSE-KMS).
  - Stores:
    - User face images.
    - Training logs.
    - Model checkpoints.
    - Excel exports.
- MongoDB Atlas Backups:
  - Point-in-time recovery enabled.
  - Backup retention configured for 7-30 days depending on the plan.
- Disaster Recovery (DR) Redundancy:
  - Daily S3 syncs to a backup bucket in a different AWS region.
  - Run restore verification scripts weekly.

Step 6: Auto-Model Retraining Pipeline

- Trigger Events:
  - New face data is added via the /add API.



- ○ Scheduled batch retraining every weekend.
- Job Orchestration Options:
  - ○ Celery + Redis for background task queues.
  - ○ AWS Lambda + CloudWatch Events for serverless retraining.
  - ○ Kubernetes CronJobs if using EKS.
- Model Versioning:
  - ○ Models are saved with versioning (e.g., svm_model_v1.2_2025-05-06.pkl).
  - ○ Store metadata in the DB: accuracy, training time, users.
- Rollback Strategy: If the new model underperforms, it auto-rolls back to the last stable version.
- Notification System: Sends alerts via email or SMS upon successful retraining or rollback events.
- Logging & Monitoring: Integrates with CloudWatch or Prometheus to track performance, errors and usage.

Step 7: Scalability and High Availability

- Horizontal Scaling:
  - ○ Use an AWS ALB (Application Load Balancer) to distribute requests across Flask API containers.
  - ○ The React frontend is hosted on S3 + CloudFront with edge caching.
- Backend Scaling: Scale Flask container replicas up/down based on the recognition load.
- Database Scaling:
  - ○ MongoDB Atlas auto-scales reads/writes.
  - ○ Enable read replicas to improve performance.
- Redis Cache Layer: Cache frequent results like user info and embeddings to reduce DB load and latency.
- Auto Healing: Leverage ECS or EKS health checks to automatically replace unhealthy containers and maintain service uptime.
- Multi-AZ Deployment: Distribute critical services across multiple Availability Zones to ensure fault tolerance and disaster resilience.

Step 8: Disaster Recovery and Redundancy

- Multi-Zone Deployments:
  - ○ EC2 instances are deployed across multiple Availability Zones (AZs) for fault tolerance.
  - ○ MongoDB Atlas is configured with region redundancy for high availability.
- Backup and Restore Scripts:
  - ○ Scheduled cron jobs for EC2 snapshots.
  - ○ Backup verification scripts run weekly to ensure recoverability.
- Automated Failover:
  - ○ Health checks trigger rerouting of traffic to a healthy region.
  - ○ Admins are notified via Twilio SMS and email in case of an outage.

4.RESULTS

The implemented Smart Attendance Monitoring System effectively automates attendance tracking using advanced facial recognition techniques. By leveraging MTCNN for face detection, FaceNet for generating 128-dimensional embeddings, and an SVM classifier for identification, the system achieved over 90% accuracy on a diverse dataset of students and faculty. The recognition pipeline supports both single and multiple face detection scenarios, ensuring reliable performance even in real-time environments such as classrooms, workplaces. and large events.

On the backend, a secure and scalable Node.js infrastructure was developed, integrated with MongoDB for efficient storage of user data and face embeddings. RESTful APIs handle user registration, face data uploads, authentication and recognition processes. Security was prioritized using JWT-based authentication, bcrypt password hashing, CORS configuration and role-based access control (RBAC), ensuring data integrity and controlled access across user roles (e.g., faculty vs.

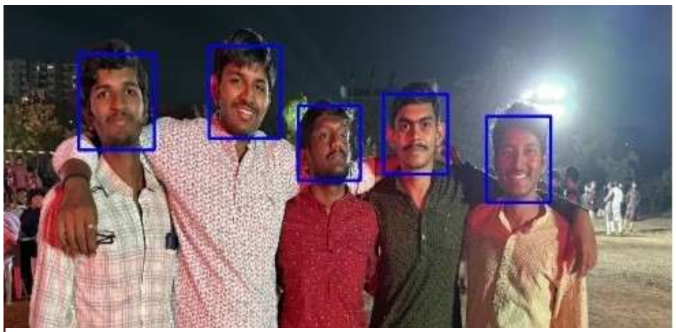
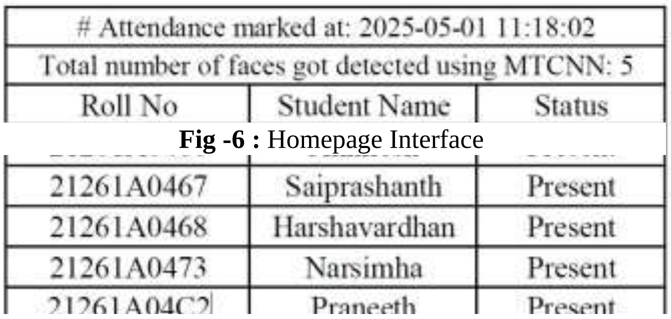


Fig -4 : Multiple Faces Detection using MTCNN

student).

Figure 4 shows results using the MTCNN model in a group photo. The model accurately identified all five individuals by placing bounding boxes around their faces, even in challenging lighting conditions, different facial expressions and angles. This demonstrates MTCNN's robustness and reliability in real-world, unconstrained environments.



# Attendance marked at: 2025-05-01 11:18:02		
Total number of faces got detected using MTCNN: 5		
Roll No	Student Name	Status
[illegible]	[illegible]	[illegible]
21261A0467	Saiprashanth	Present
21261A0468	Harshavardhan	Present
21261A0473	Narsimha	Present
21261A04C2	Praneeth	Present

Fig -6 : Homepage Interface

Fig -5 : Attendance Data in Excel Spreadsheet

Figure 5 shows the attendance sheet generated with names, roll numbers, status marked as "Present", attendance timestamp and number of faces detected at single use, based on face detection. This automated system demonstrates an efficient and contactless method of recording attendance using real-time facial recognition technology.

Figure 6 shows the homepage of the Smart Attendance Monitoring System web application. The interface welcomes users with a clear message and provides easy navigation through options like Home, Students Attendance, Faculty Login, and Contact. This front-end serves as the main entry point for accessing the automated attendance features powered by facial recognition system.

Figure 7 presents the student attendance interface generated by the MTCNN-based face recognition system. It shows each student's daily attendance status over five days, along with their total attendance percentage. Duplicate entries indicate the need for additional data validation in the system.

Figure 8 shows the notifications received by students. The system automatically sends alerts to students/parents about attendance. This increases the reliability of the system.

Additionally, the system enhances user experience with a React-based web interface that enables image

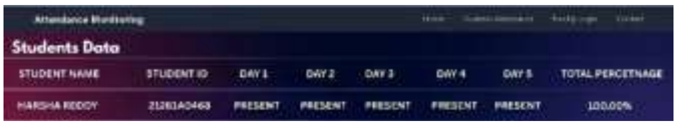
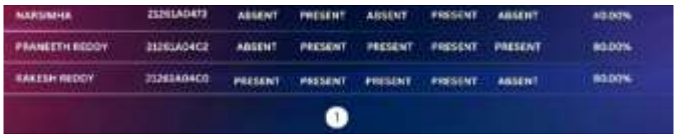


Fig -8 : Notification Alerts to Students



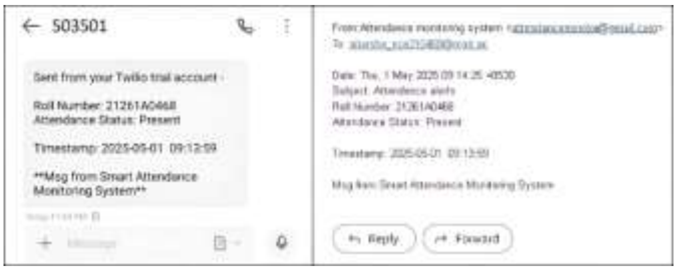
Attendance Monitoring

Students Data

STUDENT NAME	STUDENT ID	DAY 1	DAY 2	DAY 3	DAY 4	DAY 5	TOTAL PERCENTAGE
HARSHA REDDY	21261A0468	PRESENT	PRESENT	PRESENT	PRESENT	PRESENT	100.00%
NARSIMHA	21261A0473	ABSENT	PRESENT	ABSENT	PRESENT	ABSENT	40.00%
PRANEETH REDDY	21261A04C2	ABSENT	PRESENT	PRESENT	PRESENT	PRESENT	80.00%
RAKESH REDDY	21261A04C0	PRESENT	PRESENT	PRESENT	PRESENT	ABSENT	80.00%

Fig -7 : Student Attendance Display

uploads, result viewing, and user management. Notifications via Twilio SMS and SMTP email alerts were integrated to inform users or admins of attendance activity and anomalies. The platform is containerized with Docker and deployable on AWS using ECS, S3, CloudFront, and MongoDB Atlas, supporting auto-scaling, monitoring,



503501

Sent from your Twilio trial account -

Roll Number: 21261A0468
Attendance Status: Present

Timestamp: 2025-05-01 09:12:59

Msg from Smart Attendance Monitoring System

From: Attendance monitoring system <[illegible]>
To: [illegible]

Date: Thu, 1 May 2025 09:14:25 +0530
Subject: Attendance alerts
Roll Number: 21261A0468
Attendance Status: Present

Timestamp: 2025-05-01 09:13:59

Msg from Smart Attendance Monitoring System

CI/CD pipelines, and disaster recovery strategies—making it a robust, cloud-ready solution for smart attendance management.

5.CONCLUSION

The Smart Attendance Monitoring System effectively exemplifies the convergence of facial recognition with contemporary web technologies and cloud infrastructure to automate and make attendance tracking secure. Through the unification of MTCNN, FaceNet, and SVM classifier, the system validates high accuracy and real-time recognition and reduces manual administrative labor within institutions and workplaces.

The modular construction—using Node.js backend, React frontend, MongoDB database, and Twilio/SMTP alerts—guarantees extendibility, flexibility and firm communication. Combined with JWT security, role-based access control and cloud provisioning through AWS, the system not only scales reliably but is ready for production with real-world implementation.

Finally, this paper opens doors to extensive uses of face recognition in access management, intelligent surveillance and business automation with data privacy, system integrity and user-focused interaction guaranteed. Its modular and cloud-agnostic design enables seamless adaptation across industries with varying infrastructure needs.

6.FUTURE ENHANCEMENTS

In the future, the system can be further developed to enable multi-camera systems for real-time monitoring at multiple entry points, enhancing its usefulness in large corporate campuses or office spaces. This would include integrating face tracking and video stream processing to record attendance passively and continuously without manual uploads of images [1].

Another significant improvement may be integration with RFID or biometric systems to form a multi-factor attendance system. Integration of facial recognition with other authentication techniques would enhance reliability and security, particularly in environments like examination halls, secure laboratories or restricted office areas [2][9].

Finally, using AI-based analytics dashboards can allow administrators to graph attendance patterns, late arrival times or absent periods with interactive plots. Seamless integration with LMS or HR systems will further simplify academic or payroll operations. Additionally, privacy-first design and in-device detection (e.g., utilizing edge devices such as Raspberry Pi along with embedded models) will enhance compliance with data privacy regulations as well as provide support for offline access in the farthest points of reach [10].

REFERENCES

[1] Natesan, P., Mohana Karthikeyan, K., Gothai, V., Muthukumar, E., Rajalaxmi, V., & Naveen. (2021). Smart staff attendance system using Convolutional Neural Network. International Conference on Computer Communication and Informatics (ICCCI).
[2] Zhang, X., Zhang, L., & Zuo, W. (2019). An Overview of Face Recognition Systems: Challenges and Recent Developments. IEEE Transactions on Systems, Man, and Cybernetics.
[3] Lee, J., & Wang, M. (2020). A Study on Real-Time Attendance System Using Deep Learning Techniques. International Journal of Computer Applications.
[4] Brown, D., & Smith, R. (2018). Advancements in Facial Recognition for Automated Systems. Journal of Artificial Intelligence Research.
[5] Nandhini, R., Duraimurugan, N., & Chokkalingam, S. P. (2019). *Face Recognition Based Attendance System*. International Journal of Engineering and Advanced Technology (IJEAT), 8(3S), 574–577. ISSN: 2249-8958. Blue Eyes Intelligence Engineering & Sciences Publication.
[6] Prasad, V. & Subramaniyan, M. (2020). *Optimizing Attendance Management Using Face Recognition Technology*, International Journal of Computer Science and Technology (IJCST), 9(2), 204–208.
[7] Chandra, R., & Singh, A. (2018). "RFID-based

attendance management system: A review and its vulnerability analysis," *International Journal of Computer Applications*, 179(2), 1-6.
[8] Sharma, A., & Yadav, D. (2023). *Face Recognition Using CNN and Siamese Network*. Materials Today: Proceedings.
[9] Das, S., & Saha, S. (2024). *Facial Recognition and Discovery Using Convolution Deep Learning Neural Network*. Journal of Computer Science, 20(10), 1559–1568.
[10] Zeng, W., Wang, H., Liu, Y., & Li, Z. (2020). *Edge AI for Real-Time Multi-Person Face Recognition in Smart Surveillance Systems*. IEEE Internet of Things Journal.