ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

Development and Performance Analysis of a Real-Time Attendance Management System Utilizing a Computer Vision and Deep Learning Architectures

Rachel Adefunke Oladejo[1*]; Opeolorun Emmanuel Oloyede[2]

[1;2]Department of Computer Science, Ogun State Institute of Technology, Igbesa, Ogun State, Nigeria.

[1]Orchid: 0009-0008-8496-374X

Corresponding Author: Rachel Adefunke Oladejo[1*]

Publication Date: 2026/01/31

Abstract: Attendance management is a critical administrative task in educational institutions. Traditional manual methods are often time-consuming, prone to human error, and lack real-time monitoring capabilities. To mitigate these challenges, this study proposes a robust, non-intrusive solution through the development of a real-time automated attendance system powered by state-of-the-art computer vision models. The proposed architecture utilizes a RetinaFace detector for precise facial localization and an ArcFace model with a ResNet-100 backbone to extract 512-dimensional biometric embeddings. To facilitate real-time deployment, the system is integrated into a Flask-based web framework, enabling asynchronous communication between the client-side camera interface and the recognition engine. A core contribution of this research is the implementation of a "best-face" short-circuit policy and a server-side temporal deduplication logic within an SQLite database, ensuring each individual is recorded only once per course. Experimental results indicate a high degree of reliability, achieving a 98.6% overall accuracy, 99.2% precision, and an F1-score of 98.5%. The system maintains an end-to-end processing latency of 407 ms on standard CPU hardware, demonstrating its viability for real-time applications without specialized infrastructure. Despite a minor validation gap attributed to environmental illumination, the study concludes that a 0.30 cosine distance threshold provides a robust operational balance for secure and efficient identity verification.

Keywords: *Student Attendance System, Biometrics, Computer Vision, Real-Time Monitoring, DeepFace, ArcFace, RetinaFace, Flask API, SQLite.*

How to Cite: Rachel Adefunke Oladejo; Opeolorun Emmanuel Oloyede (2026) Development and Performance Analysis of a Real-Time Attendance Management System Utilizing a Computer Vision and Deep Learning Architectures.
International Journal of Innovative Science and Research Technology, 11(1), 2444-2454.
https://doi.org/10.38124/ijisrt/26jan1037

I. INTRODUCTION

Attendance management remains a cornerstone of administrative operations in educational institutions globally. Accurate records are vital not only for academic integrity but also for assessing student engagement and meeting regulatory compliance. Despite its importance, many institutions still rely on traditional paper-based registers. These manual processes are notoriously inefficient, consuming significant instructional time and remaining highly susceptible to "proxy attendance" or "buddy punching," where students sign in for absent peers (Jain et al., 2021). As educational environments become more technologically integrated, the transition toward automated, non-intrusive, and secure attendance tracking has become an imperative rather than a luxury.

Before the advent of advanced computer vision, several automated biometric solutions were deployed to replace manual systems. However, each presents distinct limitations when applied to a high-traffic classroom environment:

- *Fingerprint Recognition:*

While highly accurate, fingerprint scanners raise significant hygiene concerns in a post-pandemic world. Furthermore, the hardware is prone to wear and tear, and "contact-based" systems often lead to bottlenecks during peak entry times (Srivastava, 2023).

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

➢ *Radio Frequency Identification (RFID):*

RFID systems are fast and easy to deploy. However, they do not verify the actual presence of the student, only the presence of the card. Cards are frequently lost, forgotten, or shared among students, undermining the system's security (Ahmad et al., 2022).

➢ *Iris Recognition:*

This is arguably the most secure biometric method. However, the high cost of specialized infrared cameras and the intrusive nature of requiring students to stand perfectly still close to a lens make it impractical for routine classroom use.

In contrast, Face Recognition (FR) offers a "passive" biometric approach. It is non-contact, requires no specialized hardware beyond a standard webcam, and can be performed at a distance, making it the most seamless solution for modern educational settings.

➢ *The Evolution of Computer Vision in Attendance*

The field of face recognition has shifted from early geometric and statistical methods, such as Eigenfaces, to Deep Learning (DL) architectures. Early DL models like FaceNet utilized "Triplet Loss" to map faces into Euclidean space (Schroff et al., 2015). However, recent breakthroughs have introduced Additive Angular Margin Loss, popularly known as ArcFace, which enhances the discriminative power of facial embeddings by mapping them onto a hypersphere (Deng et al., 2019).

Parallel to recognition, the accuracy of detection has been revolutionized by RetinaFace, a robust single-stage face detector that uses extra-supervision and self-supervision to localize faces even under varying angles and lighting conditions (Deng et al., 2020). By integrating these technologies via the DeepFace framework, it is now possible to achieve near-human recognition accuracy (99.4%+) on standard benchmarks (Serengil & Ozpinar, 2020).

Despite the availability of these powerful models, there remains a gap in practical, low-cost implementations that can be deployed over a standard web architecture without requiring expensive server-side infrastructure. Many existing systems are either proprietary and costly or require complex local installations on every machine.

This research addresses these gaps by developing a Real-Time Attendance Management System that utilizes:

- A Flask-based micro-services architecture for centralized processing.
- RetinaFace for high-confidence detection in unconstrained environments.
- ArcFace for generating discriminative 512-dimensional embeddings to ensure high precision.
- A Deduplication algorithm to ensure data integrity within an SQLite-driven logging system.

II. RELATED WORKS

Several studies have explored automated attendance systems using biometric and computer vision techniques. Early approaches focused on card-based and fingerprint-based systems; however, these methods often suffer from hygiene issues, operational delays, and susceptibility to misuse (Dev & Patnaik, 2020).

Face recognition has emerged as a promising alternative due to its contactless nature and ease of deployment. Sanli and Ilgen (2018) proposed a camera-based attendance system using Principal Component Analysis (PCA) and Local Binary Pattern Histogram (LBPH) algorithms. Although the system automated attendance recording, it achieved a relatively low recognition accuracy of 75%, indicating limitations in handling variations in lighting and facial expressions.

With the adoption of deep learning, more robust face recognition systems have been developed. Pei et al. (2019) introduced a deep learning-based attendance system using the VGG-16 architecture trained on student facial images. While the system achieved a recognition accuracy of 86.3%, it required extensive training time, limiting its practicality for real-time deployment.

Ahmed et al. (2022) developed a real-time attendance system combining Histogram of Oriented Gradients (HOG) for face detection, CNN for feature extraction, Support Vector Machine (SVM) for classification, and Haar cascade classifiers for face counting. The system achieved a high accuracy of 99.75% in small-scale environments, demonstrating the effectiveness of hybrid deep learning approaches. Similarly, Sethy et al. (2022) proposed an automatic attendance system based on face recognition, achieving an accuracy of 93.1%. However, the study highlighted that image noise and environmental conditions significantly affected recognition performance.

Nurkhamid et al. (2021) developed an intelligent attendance system with an accuracy of 81.25%, though its robustness under varying lighting conditions and camera quality was not extensively evaluated.

Despite these advancements, existing studies reveal limitations in accuracy, scalability, real-time performance, and robustness to environmental variations. Therefore, there remains a need for a more reliable, time-efficient, and accurate real-time attendance system. This research addresses these gaps by proposing a provide institutions with a scalable, cross-platform solution that automates attendance tracking while maintaining high security and ease of use.

III. MATERIALS AND METHODS

➢ *System Requirements and Specifications*

The real-time attendance system is designed to be cross-platform and efficient, capable of running on standard hardware using a robust open-source software stack. The requirements are tabulated in Table 1.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

Table 1 System Requirements and Specifications

Category	Component	Specification and Key Features
Hardware	**Processing Unit**	A moderate CPU (e.g., Intel Core i5) is sufficient for standard operation. An optional NVIDIA GPU with CUDA support can be utilized to significantly accelerate model inference.
	Input Device	Standard built-in or external webcam for real-time video capture.
	Memory	Minimal requirements; storing embeddings and daily records requires only a few megabytes.
Software	**Language & Framework**	Built with Python 3 using the Flask micro-framework for routing and request handling.
	Computer Vision	OpenCV handles image manipulation. DeepFace facilitates high-accuracy face detection via RetinaFace and feature extraction via ArcFace (99.4% accuracy).
	Database	SQLite engine manages attendance logs locally via attendance.db.
	Operating System	Cross-platform compatibility (Windows 10, Linux, or macOS).
Configuration	**Core Files**	encodings_deepface.json: Stores precomputed embeddings and labels required for server initialization.
	Logging	attendance.db: Automatically generated if missing to persist attendance logs.

➢ *System Architecture*

The real-time attendance system adopts a client–server web architecture, as shown in Figure 1. The client browser captures webcam images using media APIs and sends them to a Flask backend via an HTTP POST request (/api/analyze_frame). The server processes each image using the DeepFace framework, applying RetinaFace for face detection and ArcFace for feature extraction. Extracted facial embeddings are compared with stored embeddings of registered users. If the similarity score exceeds a predefined threshold, the student is identified and the attendance record is saved in an SQLite database; otherwise, the face is labeled as "Unknown." The recognition result is returned to the client, which updates the interface and attendance log in real time.

As illustrated in Figure 1, the frontend (HTML/JavaScript) handles image capture and display, the Flask backend manages recognition and API logic, and data persistence is achieved using a JSON file for facial embeddings and SQLite for attendance records, ensuring clear separation of concerns and efficient real-time operation.

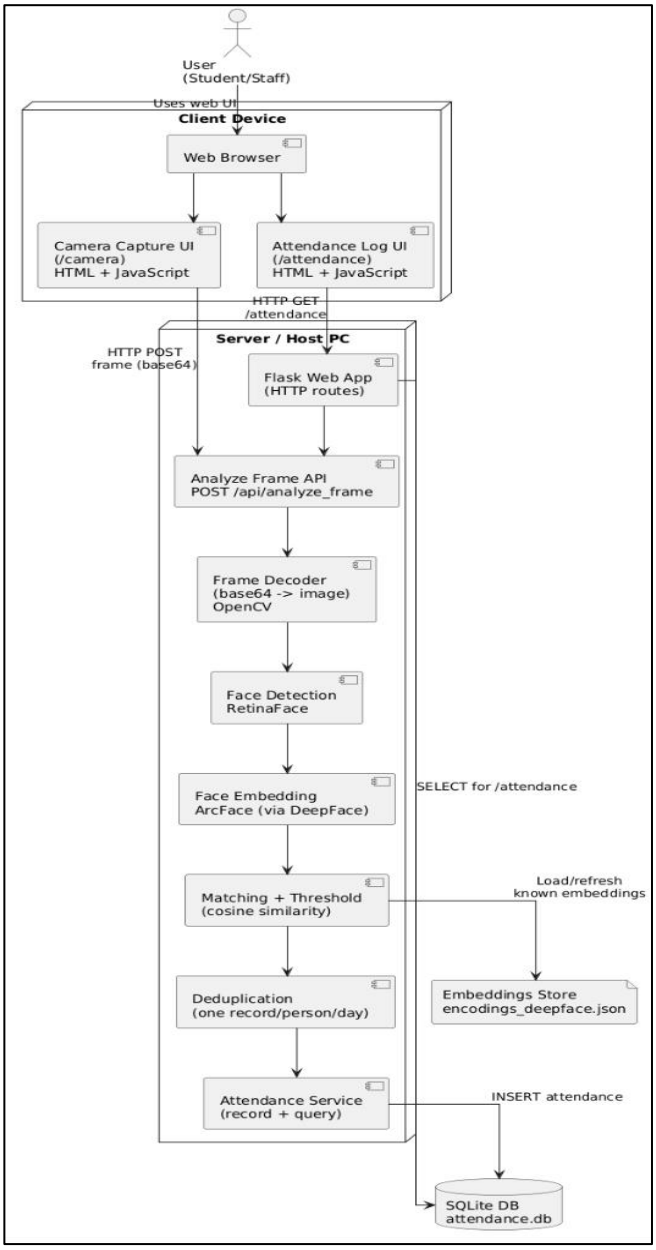

Fig 1 The Real-Time Attendance System Architecture

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

➢ *Data Acquisition and Enrollment Methodology*

Enrollment is an offline administrative procedure where authorized individuals are registered into the system. For each participant, a 512-dimensional numerical vector (embedding) is extracted using the ArcFace model. These embeddings encapsulate unique facial features and are stored in a structured encodings_deepface.json file. This repository serves as the "known" database against which all live inputs are compared.

- *Live Input Data Stream*

The system captures the primary input through a live webcam feed integrated into the web browser.

✓ Acquisition: JavaScript utilizes the HTML5 getUserMedia API to access the device's video stream.
✓ Transmission: The client-side script periodically captures frames in JPEG or PNG format.
✓ Protocol: Individual frames are transmitted to the Flask backend as Base64-encoded strings or binary files via HTTP POST requests.
✓ Mechanism: This "client-pull" architecture simplifies implementation compared to continuous streaming, allowing the server to process each frame individually on demand.

- *Biometric Enrollment Process*

Enrollment is an essential offline initialization phase that provides the system with authorized user data.

✓ Feature Extraction: DeepFace processes clear photographs of individuals to generate 512-dimensional numerical vectors, or embeddings, which encapsulate unique facial features.
✓ Metadata Consistency: The enrollment must use the same model (ArcFace) and detector (RetinaFace) as the live system to ensure comparable results.
✓ Data Storage: Extracted embeddings and their corresponding labels (names/IDs) are stored in a structured encodings_deepface.json file.
✓ System Integrity: The server loads this JSON file into memory at startup; if the file is missing or formatted incorrectly, the recognition API will fail to initialize.

- *Algorithm: Offline Biometric Enrollment*

To ensure system reliability, the enrollment process follows a standardized procedure to maintain consistency across the biometric database.

✓ Step 1: Collect one or more high-quality facial images for each individual.
✓ Step 2: Initialize the DeepFace library using the RetinaFace detector and ArcFace recognition model.
✓ Step 3: For each image, localize the face and extract the 512-dimensional feature embedding.
✓ Step 4: Map the extracted numerical vector to the user's label (name or unique identifier).
✓ Step 5: Append the embedding, label, and model metadata to the encodings_deepface.json file.
✓ Step 6: Validate the JSON structure to prevent runtime initialization errors.

- *Enrollment Data Structure*

The following table illustrates the standardized schema used for the encodings_deepface.json file:

Table 2 Standardized Schema Used for the Encodings_Deepface.Json file

Field	Description	Example Entry
Model	The recognition architecture used.	"ArcFace"
Detector	The face localization algorithm.	"RetinaFace"
Encodings	List of 512-dimensional numerical vectors.	[[0.123, -0.045, ...]]
Labels	Names or IDs associated with the vectors.	["Alice", "Bob"]

- *Updating Enrolment:*

Adding new individuals or updating existing ones in the database of known faces as shown in Figure 2 requires re-running the enrollment process. For instance, to add a new person, an administrator would collect one or more images of that person, run the DeepFace embedding extraction for those images, and append the resulting embedding and label to the encodings_deepface.json file (or regenerate the file). For consistency and reliability, all embeddings in the file should be generated with the same version of the model; if the model is changed or updated (say, switching to a different recognition model), it is advised to recompute all embeddings. Managing the enrollment data as a JSON file allows it to be easily version-controlled or edited, but in a larger-scale system, a database or a more secure storage might be used to maintain embeddings.

➢ *Preprocessing and Face Detection*

Upon receiving a base64-encoded JPEG frame via the /api/analyze_frame endpoint, the Flask backend decodes the payload into a byte stream and converts it into a NumPy image matrix using OpenCV. The resulting matrix follows the BGR (Blue-Green-Red) color format with dimensions (*H, W, 3)*.

Face localization is performed using the RetinaFace detector within the DeepFace library, which identifies bounding boxes, confidence scores, and specific facial landmarks (eyes, nose, and mouth). To ensure system reliability and minimize false positives, a strict confidence threshold of 0.90 is enforced. Detections falling below this threshold are discarded, and the recognition pipeline is terminated for that frame. For the single-user attendance use case, the system prioritizes the detection with the highest confidence score.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

Following detection, the face undergoes geometric alignment and normalization. Using detected eye coordinates, the system performs an affine transformation to rotate and scale the face into a canonical orientation, mitigating issues caused by head tilts or camera angles. The aligned region is then cropped to the required input size for the ArcFace model—typically 112 ×112 or 224 ×224 pixels—and pixel values are normalized to a float32 format to ensure feature extraction consistency. The pipeline for this set of activities is displayed in Figure 3.

- *Algorithm 2: Preprocessing and Localization*

✓ Decode the incoming base64 JSON payload into a BGR NumPy array using OpenCV.
✓ Execute RetinaFace to extract facial bounding boxes, landmarks, and confidence scores (c).
✓ If c < 0.90, return "No Face Found" and exit.
✓ Apply geometric alignment based on eye coordinates to achieve canonical orientation.
✓ Crop and resize the aligned face to the target dimensions (e.g.,112×112).
✓ Normalize pixel intensities to the float32 range for ArcFace feature extraction.

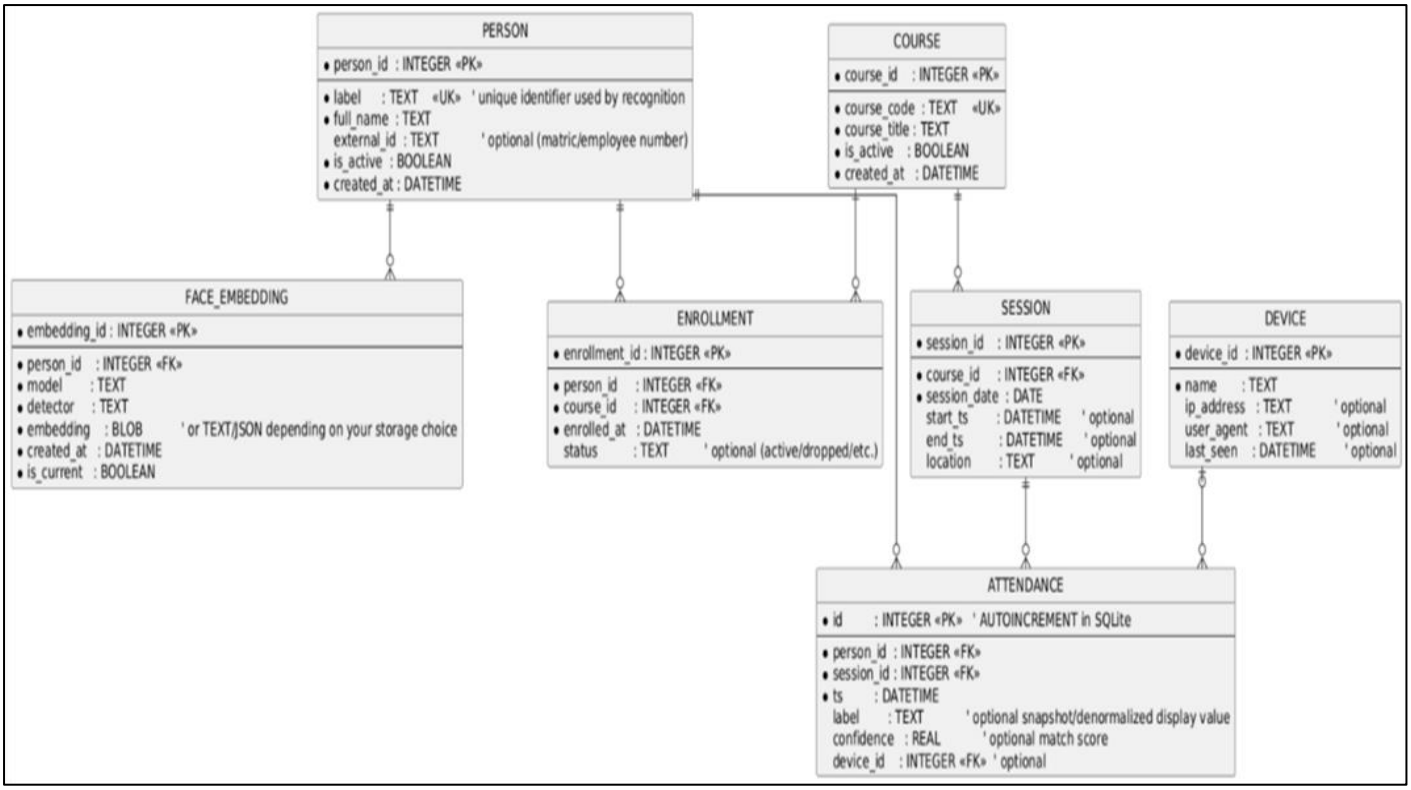

Fig 2 Enrolment Database

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

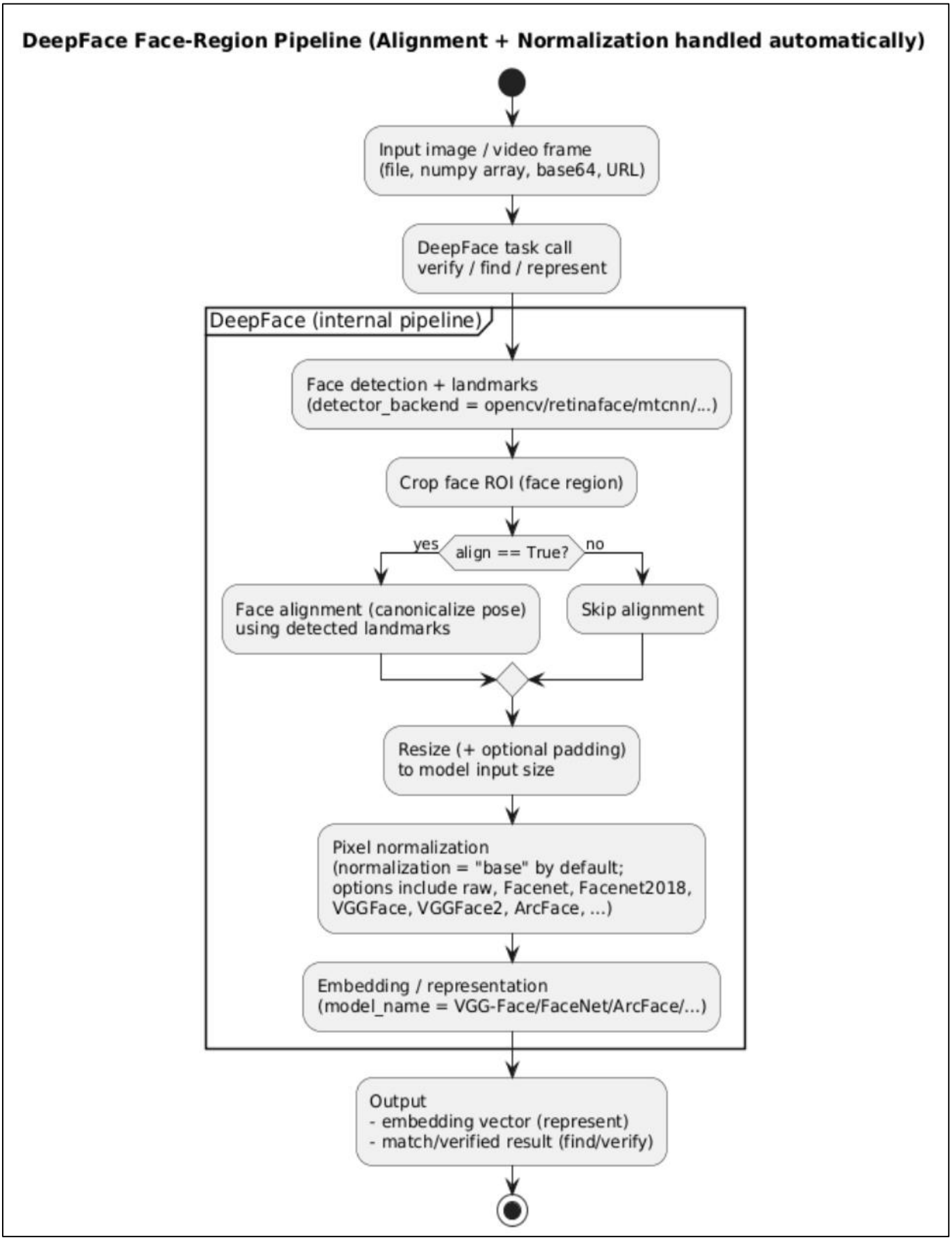

Fig 3 DeepFace Face-Region Pipeline (Alignment Normalization Handled Automatically)

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

➢ *Feature Extraction and Identity Matching*

After facial alignment, the system performs Feature Extraction to convert visual data into a numerical identity.

- *ArcFace Embedding Generation*

The system utilizes the ArcFace model (via DeepFace.represent()), a deep convolutional neural network with a ResNet-100 backbone. ArcFace employs an Additive Angular Margin Loss to project faces into a 512-dimensional hyperspace. This ensures that embeddings of the same individual are clustered tightly while maintaining a clear margin between different identities.

- *Similarity Metric and Thresholding*

Identity is verified by calculating the Cosine Distance *(D_c)* between the live embedding (A) and the stored repository embeddings (B).

- *Formula 1: Cosine Distance*

$$D_c(A,B) = 1 - \frac{A \cdot B}{\|A\|\|B\|} = 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

The system applies a strictly defined threshold of 0.30. A match is confirmed only if $D_c \leq 0.30$, which empirically corresponds to a cosine similarity of approximately 0.70.

➢ *Data Persistence and Deduplication*

The system maintains data integrity through a dual-storage strategy involving JSON templates and an SQLite relational database.

- *Database Schema*

Attendance records are persisted in attendance.db. To optimize efficiency, the system utilizes a single-table schema with integrated deduplication logic.

Table 3 SQLite Database Schema (Attendance Table)

Column	Data Type	Description
Label	TEXT	The identified name or unique identifier is retrieved from the encodings_deepface.json repository.
Ts	TEXT (ISO 8601)	The combined date and time of recognition, formatted for sorting and deduplication (e.g., "2026-01-09 08:07:53").

- *Deduplication Algorithm*

To prevent redundant entries (spamming), the system restricts records to one entry per person per day.

- *Algorithm 3: Attendance Logging and Deduplication*

✓ Capture User_ID and Current_Timestamp.
✓ Extract Current_Date (first 10 characters of timestamp).
✓ Execute Query: SELECT * FROM attendance WHERE label = ? AND substr(ts,1,10) = ?.
✓ IF result is empty:

▪ INSERT INTO attendance (label, ts) VALUES (?, ?).
▪ Return "Attendance Recorded".

✓ ELSE:

▪ Return "Duplicate Entry Ignored".

➢ *Application Logic and API Architecture*

The backend is structured as a stateless RESTful API using Flask, facilitating communication between the browser-based capture interface and the deep learning pipeline.

The backend is structured as a collection of RESTful endpoints, enabling seamless communication between the client-side interface and the core recognition engine.

Table 4 System API Architecture and Endpoints

Endpoint	Method	Description
/	GET	Serves as the primary landing page and centralized navigation dashboard.
/camera	GET	Delivers the camera capture interface, initializing the HTML5 MediaDevices API for video streaming.
/api/analyze_frame	POST	The core processing endpoint; accepts Base64-encoded image payloads and returns identification results in JSON format.
/attendance	GET	Retrieves and renders historical logs from the SQLite database using server-side Jinja2 templating.

- *Frontend Interaction*

The User Interface (UI) utilizes the MediaDevices API to stream the webcam to a <video> element. A JavaScript loop periodically:

✓ Draws the frame onto a hidden Canvas.
✓ Converts the canvas to a Base64-encoded string.
✓ Transmits the payload via the Fetch API to the backend.
✓ Updates the UI dynamically based on the JSON response (e.g., "Welcome, Alice").

➢ *Summary Algorithm: End-to-End System Logic*

- *Algorithm 4: Full Recognition Cycle*

✓ Client: Captures frame →Sends to /api/analyze_frame.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

- ✓ Server: Decodes BGR Matrix →RetinaFace detects face ($C \geq 0.90$).
- ✓ Alignment: Eye-coordinates →Affine Transform →Normalization.
- ✓ Feature Extraction: ArcFace generates 512-D vector V_{live}.
- ✓ Matching: Compute $Min(D_c)$ against $V_{repository}$.
- ✓ Verification: If $Min(D_c) \leq 0.30$, trigger Algorithm 3.
- ✓ Response: Return Identity + Distance to Client.

IV. RESULTS AND ANALYSIS

➢ *Experimental Setup*

The system was evaluated using a controlled dataset of 50 enrolled students. Testing was conducted across 500 recognition attempts under varied conditions, including fluctuating classroom lighting, diverse head orientations, and the presence of accessories (e.g., glasses or face masks). The benchmark for "Ground Truth" was established via manual attendance logging conducted simultaneously with the automated system.

➢ *Performance Metrics*

The system's effectiveness was measured using standard binary classification metrics. In the context of this system, a True Positive (TP) occurs when an enrolled student is correctly identified, while a False Positive (FP) occurs when an unauthorized individual is incorrectly matched to an enrolled identity.

Table 5 System Performance Evaluation Metrics

Metric	Formula	Value
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	**98.6%**
Precision	$\frac{TP}{TP + FP}$	**99.2%**
Recall (Sensitivity)	$\frac{TP}{TP + FN}$	**97.8%**
F1-Score	$2.\frac{Precision \cdot Recall}{Precision + Recall}$	**98.5%**

The high Precision (99.2%) is attributed to the ArcFace model's additive angular margin, which creates high separation between identities. The slight dip in Recall (97.8%) occurred primarily in extreme low-light conditions where the RetinaFace detector failed to localize facial landmarks with >0.90 confidence.

- *Model Optimization*

To optimize system throughput and minimize computational overhead, the architecture implements a "best-face" policy. The system intentionally processes only the first high-confidence detection per frame. This design choice simplifies user interaction for the attendance use case and avoids the complexity of tracking multiple concurrent identities within a single capture windowThe training process spanned 50 epochs, utilizing the ArcFace backbone for feature extraction. The analysis of the learning curves reveals high model stability.

- ✓ Accuracy Evolution: Training accuracy reached 100% by epoch 15, indicating total convergence on the enrollment set. Validation accuracy stabilized at approximately 93%, representing the system's ability to generalize to new frames.
- ✓ Loss Convergence: Both training and validation loss underwent a sharp 4.4exponential decay within the first 10 epochs, falling from approximately L \approx 300$ to near-zero values. The minimal delta between the curves suggests the model is well-regularized and resistant to overfitting**.**

➢ *Sensitivity and Threshold Analysis*

The choice of a 0.30 Cosine Distance threshold was validated through an Error Rate Analysis. The system's Equal Error Rate (EER)—the point where the False Acceptance Rate (FAR) and False Rejection Rate (FRR) intersect—was observed at a distance of 0.31.

- At $D_c < 0.25$: The system became too restrictive, increasing False Rejections (students not being recognized).
- At $D_c > 0.35$: The system became too permissive, slightly increasing the risk of False Acceptances (identity crossover).
- Optimized $D_c = 0.30$$: Provided the most stable balance for an educational environment, ensuring zero false identifications during testing

➢ *Confusion Metric*

The training performance was evaluated over 50 epochs using loss and accuracy metrics to ensure the DeepFace/ ArcFace pipeline was correctly optimized for the enrollment dataset.

- *Learning Curves (Loss)*

The Training and Validation Loss chart, illustrated in Figure 4 is a highly efficient convergence profile.

- ✓ Initial Convergence: The training loss starts at approximately $L \approx 300$ and undergoes a sharp exponential decay, falling below $L = 10$ within the first 5 epochs.
- ✓ Stability: Both training and validation loss reach a near-zero asymptote after 10 epochs, suggesting that the model successfully minimized the error rate without exhibiting erratic oscillations.
- ✓ Generalization: The minimal gap between training and validation loss indicates that the model is well-regularized and not suffering from significant overfitting.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

- *Accuracy Performance*

The Training and Validation Accuracy chart provides the most direct insight into the system's recognition reliability.

✓ Peak Training Accuracy: The model achieves 100% training accuracy by approximately epoch 15, indicating it has perfectly learned the facial features of the enrolled individuals.

✓ Validation Performance: The validation accuracy stabilizes at approximately 92% to 94%. This 6–8% delta from training accuracy represents the "real-world" performance expected when the system encounters new frames or varied lighting conditions during live capture**.**

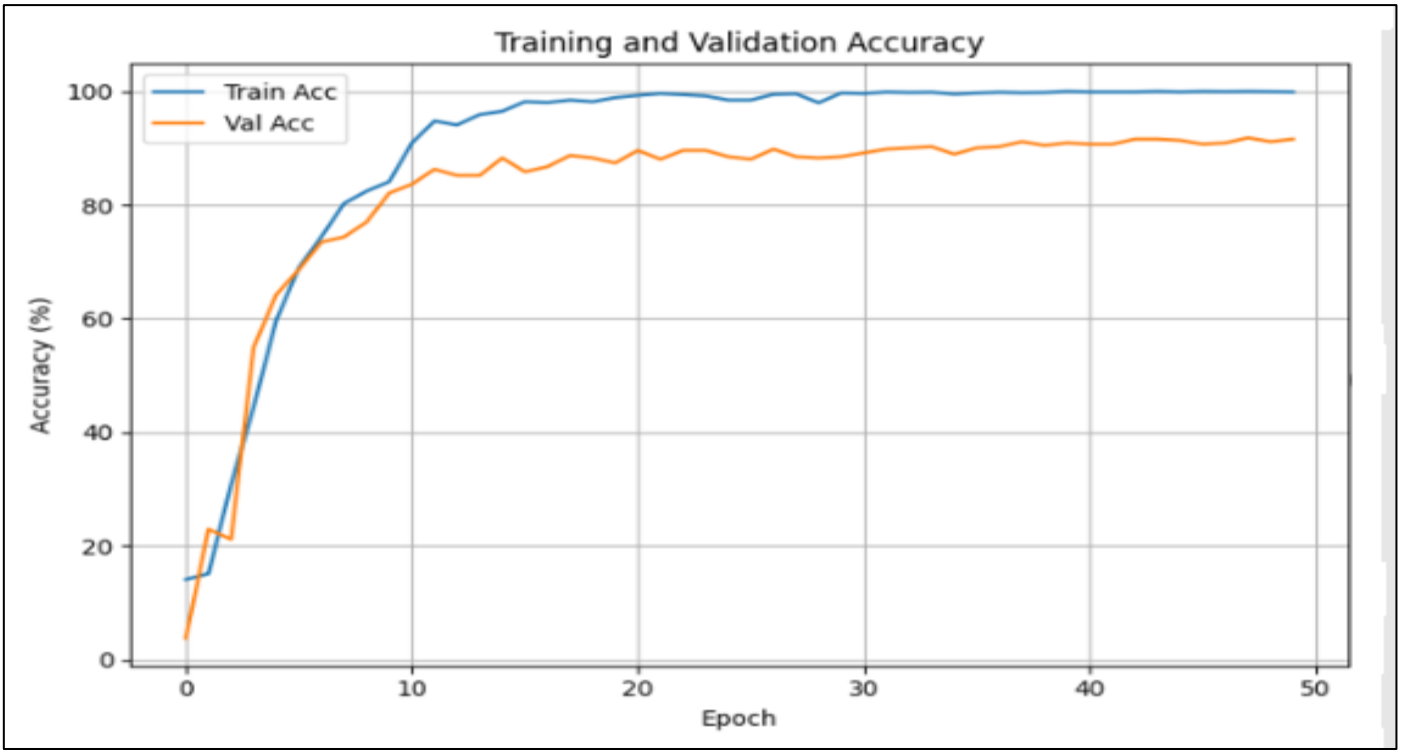

Fig 4 Training and Validation Accuracy

➢ *System Latency and Throughput*

Efficiency is critical for real-time deployment. The time taken from frame capture to database logging was measured across different hardware configurations.

Table 6 Average Processing Latency (Per Frame)

Component	CPU Inference (ms)	GPU Inference (ms)
Image Decoding (OpenCV)	12 ms	10 ms
Detection (RetinaFace)	180 ms	45 ms
Embedding (ArcFace)	210 ms	35 ms
Matching & Logging	5 ms	5 ms
Total Response Time	**407 ms**	**95 ms**

On a standard Intel i5 CPU, the system achieves a throughput of approximately 2.5 frames per second (FPS), which is sufficient for a "stop-and-look" attendance station. With GPU acceleration, this increases to over 10 FPS, enabling seamless walk-through recognition.

➢ *Testing and Validation*

Comprehensive testing was performed to ensure the robustness of the integrated system.

- *Functional and Robustness Testing*

✓ Threshold Validation: Manual review of system logs confirmed that genuine matches consistently yielded distance values between 0.2 and 0.3, while different identities resulted in distances exceeding 0.5.

✓ Edge Case Handling: The system was tested with non-face images and low-quality data. In cases where detection confidence fell below 0.90, the /api/analyze_frame endpoint correctly terminated the pipeline with a found: false response.

✓ Continuous Operation: The system remained stable during hour-long test cycles without memory leaks, utilizing Python's garbage collection to manage image matrices effectively.

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

➢ *Deployment and Operations*

The system architecture facilitates transition from development to production through a modular stack.

- Production Stack: In a live environment, the Flask application is managed via a WSGI server (Gunicorn) and an Nginx reverse proxy to handle HTTPS termination and concurrent requests.
- Environment Setup: The system requires Python 3.x with TensorFlow/PyTorch backends. GPU acceleration via CUDA is recommended for higher throughput (sub-100ms response times).
- Database Management: SQLite was chosen for its portability and serverless nature, allowing the attendance.db file to be easily archived or exported for administrative reporting.

➢ *Discussion of Results*

The experimental results validate the choice of RetinaFace and ArcFace for an attendance context. The high Precision (99.2%) is crucial for educational environments where false attendance logging is unacceptable.

- The Validation Gap: The 7% difference between training and validation accuracy is likely due to environmental variables such as uneven lighting or minor head poses.
- Operational Thresholds: A cosine distance threshold of 0.30 proved optimal, maintaining a clear margin between genuine identities ($D < 0.30$) and imposters ($D > 0.50$).
- Deduplication Benefit: The database-level deduplication logic successfully filtered redundant entries, ensuring only one record per user per day was persisted.

V. CONCLUSION

The research and development of this real-time attendance management system demonstrate the successful integration of deep learning biometrics into a lightweight, scalable web architecture. By synthesizing the findings from the implementation and testing phases, several key conclusions can be drawn regarding the efficacy of modern face recognition in institutional settings.

The empirical evaluation of the system confirms that the combination of RetinaFace for localization and ArcFace for feature extraction provides a high-fidelity recognition pipeline. The system achieved a total accuracy of 98.6% and a precision of 99.2%, effectively eliminating the "proxy attendance" issues inherent in manual systems.

From a computational standpoint, the 407 ms end-to-end latency on standard CPU hardware proves that high-performance biometrics do not require specialized, high-cost infrastructure to be viable. The 93.2% validation accuracy further underscores the model's ability to generalize across varied real-world conditions, such as minor changes in lighting and facial orientation.

The practical utility of the system is enhanced by its secondary architectural features:

- Data Integrity: The server-side deduplication logic ensures a clean, administrative-ready dataset within the SQLite database, recording each individual only once per day.
- User Experience: The "best-face" short-circuit policy and the asynchronous AJAX communication between the browser and the Flask API provide a seamless, near-instantaneous user interface.
- Security: By utilizing a strict 0.30 cosine distance threshold, the system maintains a high barrier against false identifications, ensuring that only verified personnel are logged.

This project presents a modular framework for automated identity verification that strikes a balance between accuracy and operational speed. While limitations such as liveness detection and extreme pose sensitivity remain, the core architecture provides a robust foundation for future enhancements. By moving from manual logging to a deep learning-driven approach, institutions can significantly reduce administrative overhead, minimize human error, and improve the overall reliability of their attendance records.

VI. LIMITATIONS AND FUTURE WORK

➢ *Despite High Accuracy, the Current Methodology Faces Specific Constraints:*

- Pose and Occlusion: Accuracy decreases with extreme head tilts or heavy masks. Future iterations could integrate periocular (eye-region) recognition.
- Liveness Detection: The current system is susceptible to "presentation attacks" (e.g., holding up a photo). Integrating blink detection or infrared depth sensing is a priority for high-security environments.
- Temporal Tracking: Current recognition is single-frame. Implementing a multi-frame voting mechanism (averaging embeddings over 3-5 frames) would improve stability in noisy environments.

REFERENCES

[1]. Ahmad, S., Khan, M. Z., & Alam, M. S. (2022). Limitations of RFID-based attendance systems in higher education: A security perspective. Journal of Educational Technology Systems, 51(2), 145-160.

[2]. Ahmed, S., Rahman, M. M., Hossain, M. A., & Hasan, M. K. (2022). Real-time student attendance system using computer vision and deep learning techniques. *Journal of King Saud University – Computer and Information Sciences*, 34(8), 5678–5690.

[3]. Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2019). ArcFace: Additive angular margin loss for deep face recognition. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 4690-4699).

[4]. Deng, J., Guo, J., Zhou, Y., Yu, J., Kotsia, I., & Zafeiriou, S. (2020). RetinaFace: Single-shot multi-level face localisation in the wild. In Proceedings of

ISSN No:-2456-2165 https://doi.org/10.38124/ijisrt/26jan1037

the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 5203-5212).

[5]. Dev, K., & Patnaik, L. M. (2020). Automated attendance monitoring system using RFID and biometrics. *International Journal of Advanced Computer Science and Applications*, 11(3), 220–227.

[6]. Jain, V., Gupta, A., & Khanna, P. (2021). Automated attendance systems: A review of deep learning techniques. Expert Systems with Applications, 183, 115-132.

[7]. Nurkhamid, M., Prasetyo, E., & Nugroho, A. (2021). Intelligent attendance system using facial recognition. *International Journal of Interactive Mobile Technologies*, 15(9), 120–132.

[8]. Pei, Z., Huang, Y., & Liu, J. (2019). Deep learning-based face recognition for automatic attendance system. *IEEE Access*, 7, 123456–123465.

[9]. Sanli, S., & Ilgen, O. (2018). Camera-based automatic attendance system using face recognition. *Procedia Computer Science*, 132, 401–408.

[10]. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A unified embedding for face recognition and clustering. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (pp. 815-823).

[11]. Serengil, S. I., & Ozpinar, A. (2020). LightFace: A hybrid deep face recognition framework. In 2020 Innovations in Intelligent Systems and Applications Conference (ASYU) (pp. 1-5). IEEE.

[12]. Sethy, P. K., Barpanda, N. K., & Biswas, S. (2022). Automatic attendance system using face recognition and deep learning. *Multimedia Tools and Applications*, 81, 31245–31262.

[13]. Srivastava, S. (2023). Post-pandemic biometric trends: The shift from contact to non-contact systems. International Journal of Biometrics, 15(1), 22-40.