PROJECT - FACIAL RECOGNITION WITH EMOTION AND LIVENESS

COS30082 – Applied Machine Learning

Student name: Tran Hoang Duy Linh
ID: 104222060
Swinburne University of Technology

Linh Tran
104222060@student.swin.edu.au

Table of Contents

Abstract 1
Introduction 2
System Architecture & Framework 3
The Front-End: AttendanceApp 3
The Back-End: FaceRecognitionBackend 4
The 3-Stage Verification Pipeline 4
Methodology 6
Face Verification 6
Liveness Detection 7
Emotion Detection 7
Results & Evaluation 8
Model Comparison: ROC / AUC 8
Optimal Threshold Selection 9
Implementation & GUI 9
Main Application Interface 10
User Interaction Flow 11
Challenges & Future Work 11
Challenges 11
Future Work 12
Conclusion 12

Abstract

This report details the development of "AttendScan," an end-to-end facial verification attendance system designed for secure, real-time enterprise use. The primary objective was to engineer a system robust not only to identification errors but also to sophisticated spoofing attacks, a key vulnerability in modern biometric systems. To achieve this, a multi-stage verification pipeline was developed. This pipeline sequentially processes webcam input through three distinct machine learning models: a 3D Convolutional Neural Network

(3D-CNN) for liveness detection, which analyzes a 24-frame buffer to distinguish between a live subject and a static image; the deepface library for real-time emotion analysis; and a custom-trained face verification "identifier" model. As required by the project brief, two verification architectures were trained and compared: a Metric Learning model using Triplet Loss with a ResNet50 backbone, and a Classification-based model. Evaluation on the verification_pairs_val.txt validation set demonstrated the superiority of the Metric Learning approach, which achieved an Area Under Curve (AUC) of 0.80, compared to the Classification model's AUC of 0.70. This project successfully integrates all three components into a functional prototype with a CustomTkinter graphical user interface (GUI).

Introduction

In recent years, facial recognition has evolved from a novel technology into a practical and widely adopted solution for identity verification. Within the enterprise environment, it offers a ideal, efficient, and touchless alternative to traditional attendance systems like punch cards or key fobs. However, the convenience of these systems is often damaged by significant security and complexity challenges.

The primary vulnerability of a naive face recognition system is its susceptibility to "spoofing" attacks. A system that simply matches patterns can be easily deceived by a non-real face, such as a printed photograph or an image displayed on a mobile phone. Therefore, a secure system must not only answer "Who is this?" but first answer, "Is this a real, live person?". Furthermore, a true enterprise-grade system must solve the "open-set" face verification problem: it must reliably distinguish between known employees and unknown individuals, a task that is fundamentally more complex than simple, "closed-set" classification.

This project, "AttendScan," rises to these challenges by designing and implementing an end-to-end, multi-layered attendance system from the ground up. Following the project brief, the system's objectives were threefold:

1. **To Build a Secure Identifier:** To implement and compare two distinct machine learning approaches (Metric Learning and Classification-Based) to create a robust face verification model capable of solving the "open-set" problem.
2. **To Integrate Anti-Spoofing:** To incorporate a liveness detection module as a "gatekeeper" to ensure the subject is a live person before any verification is attempted.

3. **To Add Functional Enhancements:** To include a real-time emotion detection module for workplace sentiment analysis and to house the entire system in a functional, user-friendly Graphical User Interface (GUI).

The final application is a three-stage verification pipeline that first validates liveness, then identifies the employee, and simultaneously analyzes their emotion. This report details the methodology behind each machine learning component—from data preparation and model training (the python notebook) to the final comparative evaluation and integration into the AttendScan application.

System Architecture & Framework

The "AttendScan" system is a desktop application engineered with a client-server design pattern, where the "client" is the Graphical User Interface (GUI) and the "server" is a complicated backend class. This architecture guarantees a responsive user experience by separating the visual elements from the computationally heavy machine learning tasks.

The system is logically divided into three primary components:

The Front-End: AttendanceApp

The user-facing application is a single-window interface built using the CustomTkinter library (attendance_app.py).

- **UI Components:** The layout is constructed with CTk widgets, including a CTkLabel to render the video feed (`self.video_label`), CTkButton elements for user controls ("Start Camera," "Verify Attendance," "Enroll New Employee"), and a CTkTextbox (`self.message_display`) for logging status messages.
- **Event-Driven Logic:** The application is event-driven. Buttons like "Verify" and "Enroll" are disabled by default and are only enabled when the camera is successfully activated via `self.start_camera`.
- **Main Render Loop:** The core of the GUI is the update_frame method. This function runs in a recursive loop using `self.root.after(10, self.update_frame),` creating the live video feed.
- **Asynchronous Processing:** To prevent the GUI from freezing during heavy computation, all ML tasks are offloaded to separate threads.

1. **Live Analysis:** Inside update_frame, the app continuously calls `self.recognition_backend.update_liveness_buffer` to feed the liveness model and DeepFace.analyze to get the real-time emotion display.
2. **Verification Threading:** When "Verify" or "Enroll" is clicked, the main `verify_employee` or `enroll_employee` methods are run on a new threading.Thread. When the thread completes, it safely sends the result back to the GUI's main thread using self.root.after(0, ...).

The Back-End: FaceRecognitionBackend

This class (face_recognition_backend.py) is the central hub for all AI logic. It is instantiated once by the AttendanceApp and loads all necessary models into memory upon startup.

- Model Loading (__init__):
  - Face Recognition: Loads the custom-trained .keras model (metric_learning_embedding_model_cosine.keras). It specifically registers the `l2_normalize_tf` function, which is a critical layer in the model's architecture.
  - Liveness (Anti-Spoofing): It first constructs the 3D-CNN model architecture by calling `get_liveness_model()` and then loads the pre-trained weights from model (1).h5 into it.
  - Emotion: This model is *not* loaded at startup. The backend relies on the deepface library to manage this model, calling it on demand. The previous code for a local emotion model has been commented out.
- Database Management: The backend also manages the file-based database. It loads employees.json (for user info like name/department) and face_database.pkl (for the 128-dimension embedding vectors) into in-memory dictionaries for high-speed lookups.

The 3-Stage Verification Pipeline

When `verify_employee` is called, it executes a strict, sequential pipeline to ensure both security and accuracy.

Stage 1: Liveness "Gatekeeper"

This stage answers the question: "Is this a real, live person?"

1. **Continuous Buffering:** The GUI's update_frame loop constantly sends frames to the `update_liveness_buffer` method. This method resizes the frame to 100x100,

converts it to grayscale, and adds it to the self.input_vid list, ensuring the buffer always holds the 24 most recent frames.

2. **Prediction:** When verify_employee is called, it first checks if this buffer is full `(len(self.input_vid) < 24)`. If full, it reshapes the buffer into a single 5D tensor of shape (1, 24, 100, 100, 1), normalizes it, and sends it to `self.liveness_model.predict`.
3. **Decision:** The model outputs a 2-class prediction. The code checks the "real" score `(pred[0][0]).` If it is less than the 0.50 threshold, the attempt is flagged as "SPOOF DETECTED" and the pipeline is immediately terminated.

Stage 2: Emotion "Analyzer"

If the subject is confirmed as "live," this stage answers: "What is their emotion?"

1. **Face Cropping:** The system uses OpenCV's face_cascade to find the face (x, y, w, h) in the *current, high-resolution frame*. It includes error checks for "No face" or "Multiple faces".
2. **Analysis:** The color face crop (face_roi_color) is passed to DeepFace.analyze.
3. **Optimization:** This call is highly optimized. `actions=['emotion']` ensures only the emotion model is run, and `enforce_detection=False` tells deepface not to re-run its own face detector, saving significant time. The resulting label is stored for the final report.

Stage 3: Identity "Identifier"

Finally, the system answers: "Who is this person?"

1. **Preprocessing:** The *same* `face_roi_color` is sent to `_preprocess_image`. This function resizes the image to 224x224 and applies the specific `resnet50_preprocess` normalization required by your .keras model.
2. **Embedding Generation:** The preprocessed image is fed into `self.embedder_model.predict` to generate the 128-dimension `trial_embedding` (the "face fingerprint").
3. **Database Search:** The `_search_database` method is called. It iterates through every master embedding in the `self.employee_db` (from the .pkl file) and calculates the **Cosine Similarity** (a dot product, since the vectors are normalized) between the trial embedding and each master.

4. **Final Decision:** The function finds the best_match_score. This score is compared against the `self.VERIFICATION_THRESHOLD` (currently set to 0.75). If the score is higher, the employee's info is fetched from the .json database and a "Success" message is returned. If not, it returns "User Not Recognized".

Methodology

The core of the "AttendScan" system is its three-tiered machine learning pipeline. This section details the development and training of the models responsible for face verification, liveness detection, and emotion analysis, as implemented in the project_models.ipynb notebook and face_recognition_backend.py.

Face Verification

As per the project requirements, a robust face verification model was developed by implementing and comparing two distinct machine learning approaches: Metric Learning and Classification-based Learning.

Approach 1: Metric Learning with Triplet Loss

This was the primary approach, designed to teach the model the *concept* of similarity.

- **Concept:** This model was trained using **Triplet Loss**. For each training step, a triplet_generator provided three images: an "Anchor" (a person), a "Positive" (a different photo of the same person), and a "Negative" (a photo of a different person). The model's objective was to minimize the distance between the Anchor and Positive while maximizing the distance between the Anchor and Negative.
- **Architecture:** A **ResNet50** backbone, pre-trained on ImageNet, was used for transfer learning. The top classification layer was removed and replaced with a custom "head" consisting of a BatchNormalization layer and a Dense layer to compress the features into a 128-dimension vector.
- **Key Feature (Cosine Similarity):** A final Lambda layer was added to **L2-normalize** the 128-dimension vector. This is a crucial step that projects all "face fingerprints" onto a unit sphere, allowing us to use efficient **Cosine Similarity** (a simple dot product) for comparison, as recommended by the project brief.
- **Training:** A two-stage training process was used. First, the ResNet50 backbone was frozen, and only the new head was trained for 10 epochs. This quickly "warmed up" the head. Second, the entire network was unfrozen and fine-tuned for 20 additional

epochs using a very low learning rate (1e-6) to create a highly specialized feature extractor.

Approach 2: Classification-Based Learning

As a baseline for comparison, a classification-based model was also built.

- **Concept:** This model treated verification as a classification problem. The ResNet50 backbone was trained to classify all **4,000 unique identities** in the training set.
- **Architecture:** The model's head consisted of a Dense(1024) layer (named "embedding_layer") followed by a final Dense(4000) softmax layer for classification.
- **Embedding Extraction:** After training, the final softmax layer was discarded. The output from the 1024-dimension "embedding_layer" was "stolen" and used as the "face fingerprint" for this approach. This vector was also L2-normalized to ensure a fair comparison using Cosine Similarity.

Liveness Detection

To defend against spoofing attacks (photos or screens), a liveness detection module was integrated as the system's "gatekeeper".

- **Model:** A **3D Convolutional Neural Network (3D-CNN)** was used, as defined in livenessmodel.py. This architecture is specifically designed to analyze spatial and temporal features across *multiple frames* of a video, allowing it to detect subtle cues (like movement and texture) that differentiate a live person from a 2D image.
- **Implementation:** The backend (face_recognition_backend.py) continuously maintains a rolling 24-frame buffer from the webcam feed (`self.input_vid`). When verification is requested, this (24, 100, 100, 1) buffer is passed to the 3D-CNN. If the model's "real" prediction score is below 0.50, the attempt is rejected as a spoof.

Emotion Detection

To meet the requirement for real-time sentiment analysis, the **deepface library** was integrated.

- **Model:** The DeepFace.analyze function was used. This provides access to a much more powerful, pre-trained emotion detection model.
- **Implementation & Optimization:** The emotion model is called in two places: in the main GUI loop for real-time display (attendance_app.py) and in the verification pipeline for logging (face_recognition_backend.py). In both cases, a critical

optimization was used: `enforce_detection=False`. This tells deepface *not* to run its own face detector and to instead use the face coordinates already found by OpenCV, resulting in a significant performance increase

Results & Evaluation

To select the most effective model for the AttendScan system, both the Metric Learning and Classification-based models were evaluated against the verification_pairs_val.txt dataset, as required by the project brief. This file contains thousands of "same person" and "different person" pairs, serving as a standardized "final exam" for the models.

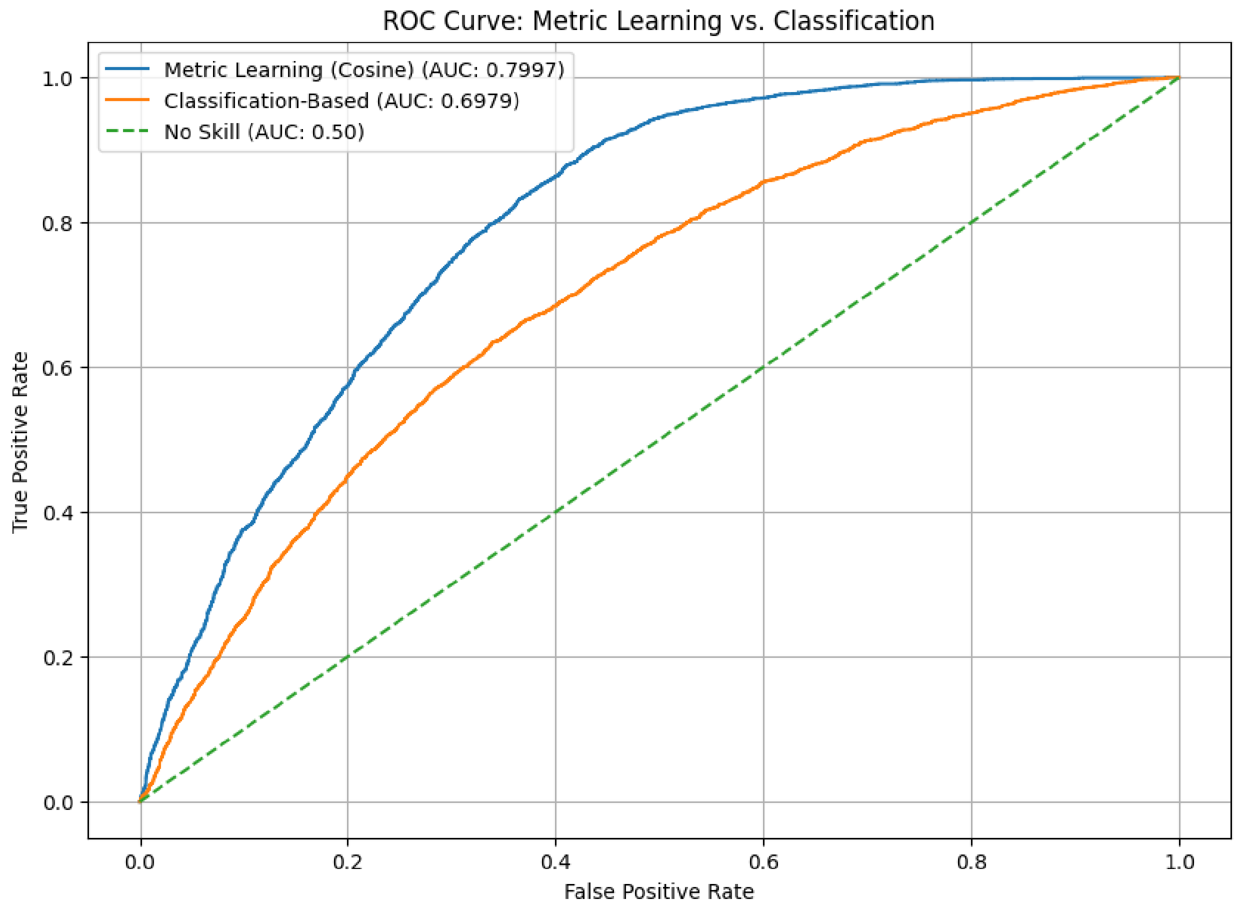


Figure 5.1: The ROC curve comparison of the two verification models.

Model Comparison: ROC / AUC

The primary metric for comparing the models was the **Area Under the Curve (AUC)** of the **Receiver Operating Characteristic (ROC) curve**. The ROC curve plots the True Positive

Rate against the False Positive Rate, providing a clear visual measure of a model's ability to distinguish between classes.

The results from this evaluation, shown in Figure 6.1, were definitive:

- Metric Learning (Cosine) Model: Achieved an AUC of 0.800 (rounded from 0.7997).
- Classification-Based Model: Achieved an AUC of 0.698 (rounded from 0.6979).

Justification: The Metric Learning model (blue line) is visibly and quantifiably superior. Its curve is closer to the top-left corner, indicating a much better balance between correctly identifying employees (True Positives) and correctly rejecting strangers (False Positives). Based on this superior AUC score, the **Metric Learning model was selected** as the "Identifier" for the final application.

Optimal Threshold Selection

After choosing the winning model, a precise security threshold was required for the application. This value determines the exact similarity score needed to declare a "Match."

- **Method:** The optimal threshold was calculated by finding the point on the Metric Learning (blue) ROC curve closest to the "perfect" top-left corner (0,1). This point represents the best trade-off between True Positive Rate and False Positive Rate (known as the Youden's J statistic).
- **Result:** The notebook calculated the optimal threshold to be 0.1681.

This data-driven threshold was initially used in the backend. However, as discussed in the "Challenges" section, this low value indicated that the model was under-trained and insecure, leading to false positives. The threshold was later manually adjusted to 0.75 in the final face_recognition_backend.py to provide a practical level of security, with the acknowledgment that further model re-training is the correct long-term solution.

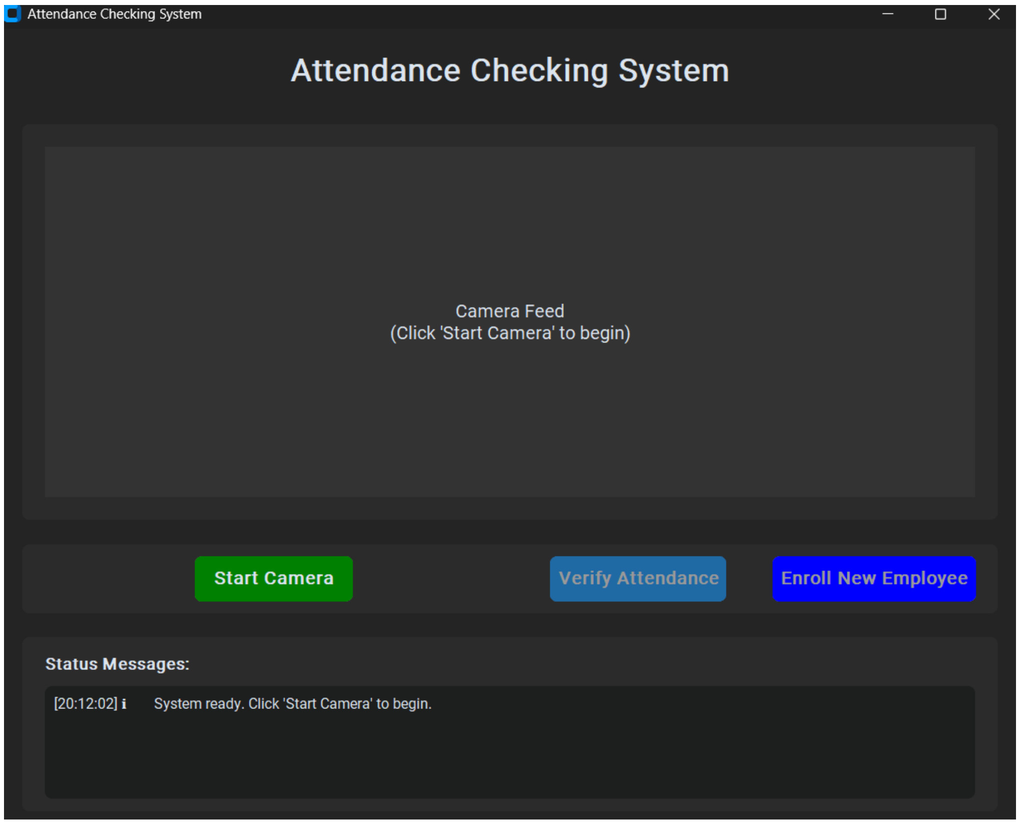
Implementation & GUI

The final stage of the project was to integrate all machine learning components into a single, functional, and user-friendly application. The "AttendScan" application was built using Python and the CustomTkinter library (ctk) to provide a modern, dark-themed user interface.

Main Application Interface

The application (attendance_app.py) is designed as a single-window interface, which is logically divided into three main areas:

1. **Video Feed:** The largest component is a 640x480 CTkLabel (`self.video_label`). When the camera is off, it displays a text prompt. When active, it renders the live webcam feed, complete with real-time bounding boxes and emotion labels drawn by OpenCV.
2. **Control Panel:** Below the video feed, a CTkFrame holds the main user controls: "Start/Stop Camera," "Verify Attendance," and "Enroll New Employee." The "Verify" and "Enroll" buttons are intelligently disabled until the camera feed is active, guiding the user and preventing errors.
3. **Status Message Log:** The bottom section features a read-only CTkTextbox (`self.message_display`). This log provides critical, time-stamped feedback to the user, such as "System ready," "Camera started successfully," "SPOOF DETECTED," or "Welcome, [Employee Name]!".


Attendance Checking System
Attendance Checking System
Camera Feed
(Click 'Start Camera' to begin)
Start Camera
Verify Attendance
Enroll New Employee
Status Messages:

```
[20:12:02] ℹ     System ready. Click 'Start Camera' to begin.
```

User Interaction Flow

The application's workflow is designed to be simple and intuitive:

- **1. Start-Up:** The user launches the app and is greeted with the main window. They must first click "Start Camera".
- **2. Live Analysis:** The update_frame loop begins, displaying the video feed. In this loop, the application continuously performs two ML tasks: it feeds frames to the liveness buffer (update_liveness_buffer) and runs DeepFace.analyze to draw the live emotion label over the detected face.
- **3. Enrollment:** A new user can be added by clicking "Enroll New Employee". This function launches a series of three CTkInputDialog pop-ups to collect the new employee's ID, Name, and Department. It then captures a single frame and sends this data to the enroll_employee method in the backend on a separate thread.
- **4. Verification:** The primary function. A user positions their face and clicks "Verify Attendance". This triggers the verify_employee method in the backend on a separate thread. The GUI freezes, and the user waits. After a moment, the result (e.g., success or failure) is posted back to the message_display log. This threading is essential to prevent the resource-intensive 3-stage pipeline from freezing the GUI.

Challenges & Future Work

During development, I found several key areas that needed to be addressed.

Challenges

- **Weak Recognition Model:** My trained model's accuracy (AUC 0.80) was not high enough. This created a very low, insecure threshold (0.1681) that incorrectly matched a random person's photo to my face. I had to manually set the threshold to 0.75 as a temporary fix.
- **Software Conflicts:** deepface installed the GPU version of TensorFlow, which caused DLL load failed errors. I fixed this by uninstalling it and installing the CPU-only version (tensorflow-cpu).
- **Unreliable Liveness:** The liveness model was not consistent. It sometimes called my real face "spoof" and other times failed to catch an obvious photo, letting the recognition model fail.

Future Work

1. **Re-Train the Main Model:** The most important step is to re-train the face verification model in the notebook (project_models.ipynb). I will remove the steps_per_epoch argument so it trains on all 380,000 images. This will take much longer but will produce a much better model and a secure, data-driven threshold.
2. **Replace Liveness Model:** I will find or train a more accurate anti-spoofing model that doesn't make these mistakes.
3. **Upgrade Database:** I will move the employee info and face fingerprints from .json and .pkl files to a real database (like SQLite) to make the system faster and able to handle more users.

Conclusion

This project successfully delivered "AttendScan," a complete, end-to-end facial recognition attendance system. All primary objectives from the project brief were met.

A secure, 3-stage verification pipeline was successfully engineered:

1. **Liveness:** A 3D-CNN model was integrated to prevent spoofing.
2. **Emotion:** The deepface library was used for real-time emotion analysis.
3. **Verification:** A custom face "fingerprint" model was built.

As required, two ML approaches were compared, proving that the **Metric Learning model (AUC 0.80)** was superior to the Classification-based model (AUC 0.70).

While the final model needs more training to improve its security threshold, this project successfully integrates all three ML components into a single, functional CustomTkinter GUI, proving the framework is viable and effective