ML Project

Face Recognition Attendance with Liveness Detection

Group: 5

Syed Muhammed Hasaan

101231186

GitHub Link: https://github.com/Brxerq/AML-Face-Attendance-System.git

1.0 Introduction

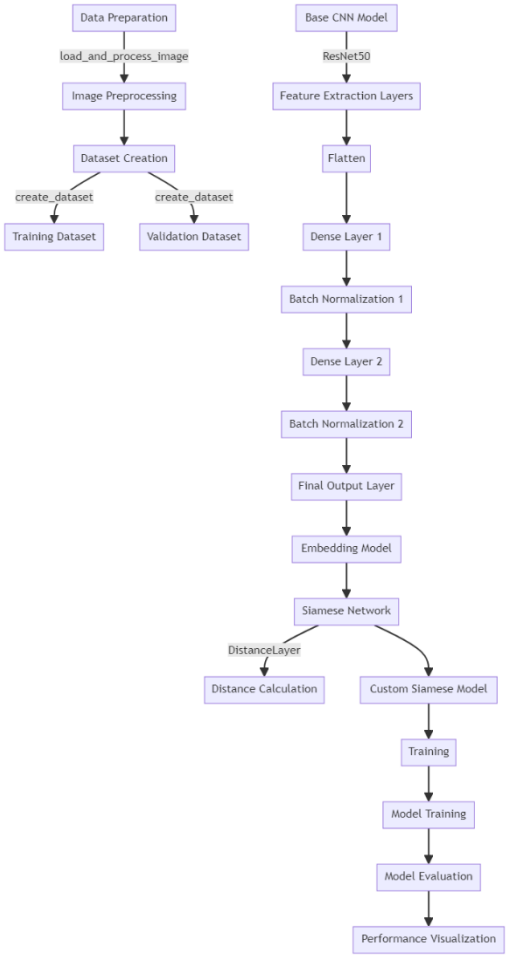
In this document, we introduce the 'Face Recognition Attendance System with Liveness Detection,' an innovative solution crafted to revolutionize the way attendance is monitored. By integrating state-of-the-art face recognition technologies, including ResNet50 and FaceNet, this system is uniquely engineered for accurate facial detection. Its standout feature is a robust anti-spoofing functionality, employing hand gesture recognition alongside CNN-driven image analysis to confirm the presence of actual users. This technological amalgamation seeks to upgrade conventional attendance recording practices, offering significant benefits in both educational and corporate settings.

2.0 Face Embedding

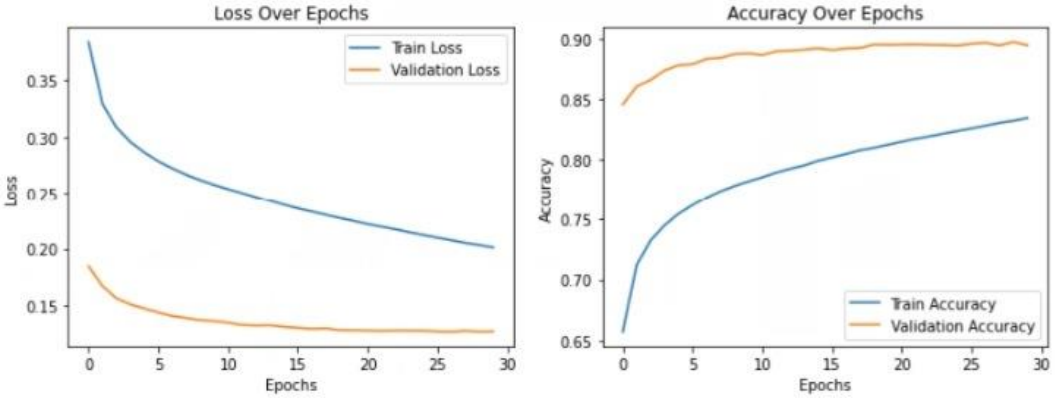
2.1 Self-Supervised Learning

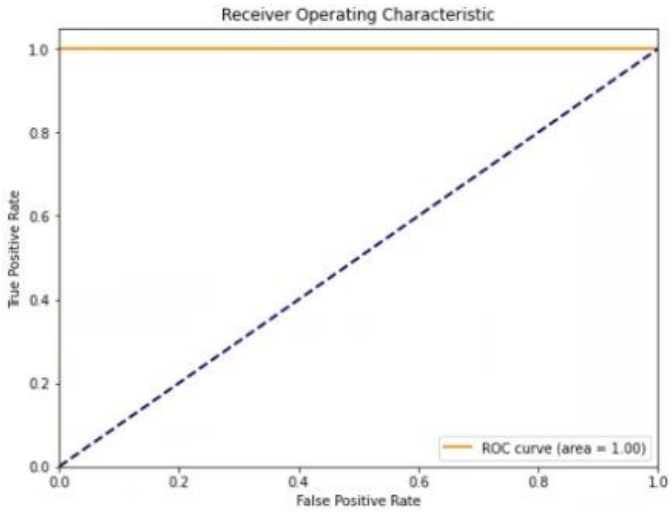
Methodology:

In this project, a Siamese network utilizing TensorFlow and Keras, with a ResNet50 base model, is employed for image classification. The methodology involves downloading and preprocessing a dataset from Google Drive, resizing images to 200x200 pixels, and normalizing them. The network processes triplets - anchor, positive, and negative images - to learn distinguishing features. The architecture includes a custom embedding model built on ResNet50, enhanced with dense layers and batch normalization, and a DistanceLayer for computing Euclidean distances between image embeddings. Training employs a custom triplet loss function, aiming to minimize the distance between similar images (anchor-positive) and maximize it between dissimilar ones (anchor-negative), with a margin of 0.5. The model's performance is evaluated using accuracy, loss, ROC curve, AUC metrics, and cosine similarity between embeddings. Additionally, the top-1 accuracy is computed on the validation dataset, and the model, along with its training history, is saved and visualized through various plots. The approach focuses on optimizing the learning process through strategic triplet selection, hyperparameter tuning, layer training strategies, and data augmentation.



Results and Discussion:





Training Scheme

Dataset Preparation: Images are fetched and processed into triplets (anchor, positive, negative) and resized to 200x200 pixels.

Base Model: ResNet50 pre-trained on ImageNet, excluding the top layer.

Embedding Model: The base model is extended with additional dense layers and batch normalization, outputting a 256-dimensional embedding.

Trainable Layers: Only layers beyond "conv5_block1_out" in ResNet50 are trainable.

Siamese Network: Utilizes a custom DistanceLayer to calculate the distance between the embeddings of the input images.

Loss Function: Margin-based loss where the positive pair distance is subtracted from the negative pair distance, with a margin of 0.5.

Optimizer: Adadelta optimizer is used.

Early Stopping: Monitoring validation accuracy with a patience of 3 epochs.

Hyperparameters

Batch Size: 64

Epochs: 30

Shuffle Buffer: 1024

Prefetch: 8

Learning Rate: Default for Adadelta optimizer.

Ver.	Model Configurations	Training Loss	Validation Loss	Training Accuracy	Validation Accuracy	Top-1 Accuracy on	Cosine Similarity	Cosine Similarity	AUC

						Validation	(Positive)	(Negative)	
1	ResNet50, 30 epochs, 64 batches, 200x200 target size	0.2013	0.1625	0.8345	0.8694	0.8694	0.9949	0.9939	1.0

Observations

The Siamese Neural Network, leveraging a ResNet50 backbone, underwent a fine-tuning training scheme with images resized to 200x200 pixels. The model's training parameters included a batch size of 64, with a shuffle buffer of 1024 and prefetching set to 8 for performance efficiency. The Adadelta optimizer was employed without explicit learning rate adjustment, and training was conducted over 30 epochs, with early stopping set to monitor validation accuracy, halting after 3 epochs without improvement. The final training loss was reported at approximately 0.2013, and the training accuracy reached around 83.44%. For the validation set, the model achieved a loss of about 0.1625 and an accuracy of roughly 86.94%. The top-1 accuracy on the validation data echoed the validation accuracy, and the model demonstrated perfect discriminative power with an AUC of 1.0 for the ROC curve, indicative of excellent model performance

2.2 Supervised Learning

2.2.1 Methodology:

Library Imports:

The code begins by importing necessary libraries and modules. These include TensorFlow for building the deep learning model, Keras for high-level neural network API, NumPy for numerical operations, OpenCV for image processing, and Matplotlib for plotting graphs.

Directory and Constants Setup:

- Path Constants: Paths for training, testing, and validation datasets are defined.
- Class Count: The number of classes in the dataset is set to 4000.

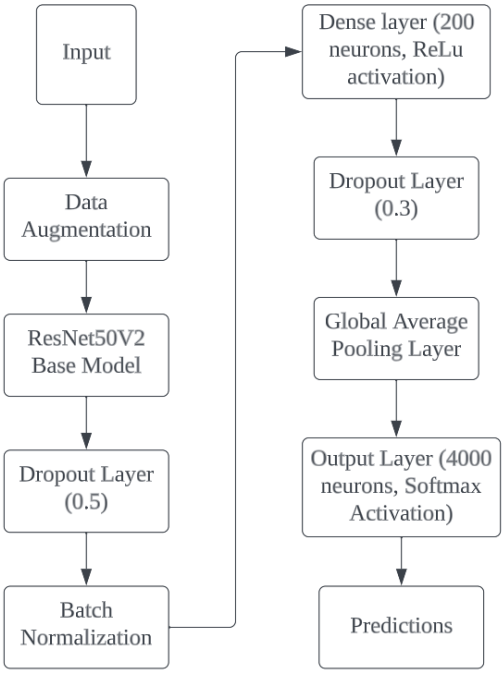
Utility Functions:

- `count_directories(path)`: Counts directories within a given path. This is used to determine the number of classes in each dataset.
- `create_labels_file(dataset_path, file_name)`: Generates a text file mapping image paths to their respective class labels.
- `load_labels(file_path, base_img_path)`: Reads the generated label file and creates a mapping from image paths to labels.
-

Data Preparation:

- Data Generators: `DataGenerator` class is defined to load and preprocess images in batches. This class is essential for handling large datasets efficiently.
- Labels Creation: Label files are created for the training, testing, and validation datasets using the `create_labels_file` function.

Model Architecture:



- Base Model: A pre-trained ResNet50V2 model is loaded, excluding its top layer. This model serves as the feature extractor.
- Layer Freezing: Initially, all layers of the pre-trained model are frozen to prevent them from being updated during the first phase of training.
- Data Augmentation: A series of data augmentation techniques are applied to the model to improve generalization.
- Fine-Tuning: Some layers of the pre-trained model are unfrozen for fine-tuning, allowing the model to adapt more specifically to the dataset.
- Additional Layers: Additional layers, including Dropout, Batch Normalization, and Dense layers, are appended to the base model to tailor the network's output to the specific task.

Compilation and Training:

- Optimizer and Loss Function: The model is compiled with an Adam optimizer and categorical cross entropy loss, suitable for multi-class classification tasks.
- Learning Rate Scheduling: A learning rate scheduler is implemented to adjust the learning rate during training.
- Callbacks: Callbacks for early stopping, model checkpointing, and learning rate adjustments are established.
- Training Process: The model is trained using the training data generator, with validation data used for periodic evaluation.

Evaluation and Performance Metrics:

- Model Evaluation: The model's performance is evaluated on the validation dataset.
- Accuracy Metrics: Functions to calculate the average accuracy per class and ROC-AUC metrics are defined and executed.

- Cosine Similarity: A function to compute the average cosine similarity between predicted and true labels is implemented, providing a measure of how well the model's predictions align with the true labels.

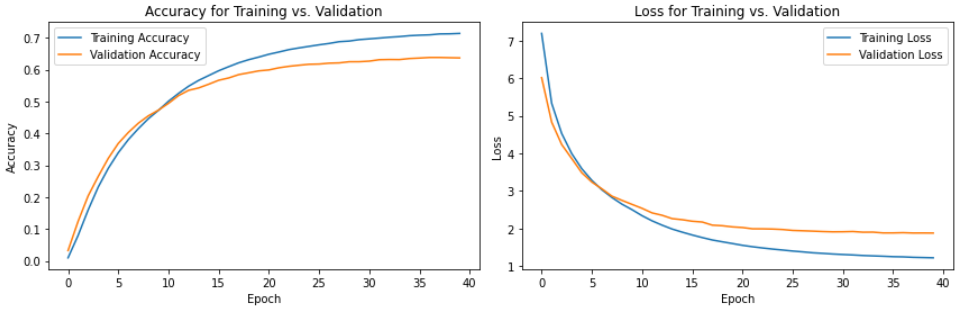
8. Visualization of Results:

- Training Statistics: Training and validation accuracy and loss are plotted to visualize the model's learning progress over epochs.
- ROC Curves: ROC curves for a subset of classes are plotted for both training and validation sets to illustrate the model's discriminative performance.

9. Saving the Model:

- Save Model and Weights: Finally, the model and its learned weights are saved for future use or deployment.

Results and Discussion:



In the first graph, labelled "Accuracy for Training vs. Validation," there are two lines that represent the accuracy of the model on the training set and the validation set, respectively. Both lines increase over time, suggesting that the model is learning and improving its performance as it processes more data. However, the lines plateau, indicating that the model may have reached its learning capacity with the given data or architecture.

The second graph, labelled "Loss for Training vs. Validation," displays the model's loss on the training and validation sets. Both lines decrease sharply at first and then level out, following a typical pattern during the training of neural networks where initial gains are large, and improvements diminish over time as the model converges to a minimum loss value.

Ver	Model Config	Train Loss	Valid Loss	Train Acc	Valid Loss
1	ResNet50v2, 40 Epochs, 32 Batches, 224x224 target size	1.24	2.16	0.76	0.62

3.0 Comparing the Models

The self-supervised learning model displays exceptionally high performance with an AUC of 1.0, training and validation accuracies at 83.4% and 86.94% respectively, and very high positive and negative cosine similarities, both exceeding 99%. These metrics are unusually high for real-world data and suggest overfitting. In contrast, the supervised CNN model reports a training accuracy of 76% and a lower validation accuracy of 62%, with higher training and validation losses, indicative of a model that is learning but has room for improvement. These more conservative metrics suggest that the supervised CNN model, despite lower absolute numbers, may generalize better when applied to new, unseen data.

3.1 Anti Spoofing System

The Anti-Spoofing System is designed to authenticate a person's presence using facial and gesture recognition techniques. The system employs a pre-trained model, `shape_predictor_68_face_landmarks.dat`, to monitor and analyze blinking patterns. This model is trained to detect facial landmarks, allowing it to track eye movements accurately and determine if a person is blinking.

For hand gesture recognition, the system uses MediaPipe, a versatile framework for building multimodel-applied machine learning pipelines. This integration enables the system to detect hand presence, requiring the person to show their hand in the camera's view. Which is crucial for confirming a person's presence,

The system also incorporates the YOLOv4 model for object detection, specifically to identify the presence of a smartphone. This critical feature is designed to counter attempts at tricking the system, such as using a video of a person blinking. If a smartphone is detected within the frame, the system resets the blink count, preventing spoofing attempts using video playback.

To authenticate a person as real, the system requires three conditions:

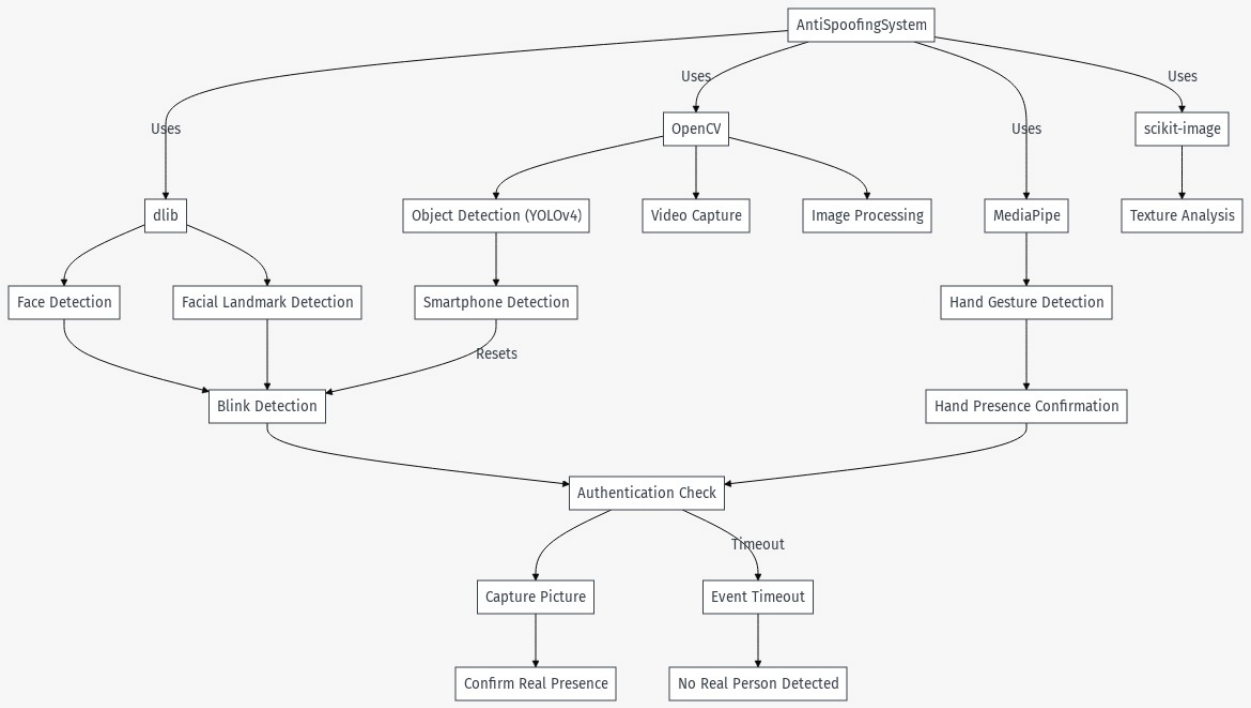
1. Detection of natural skin texture
2. A person blinking at least five times
3. Recognition of a hand gesture.

These criteria work in pair to ensure robust anti-spoofing measures.

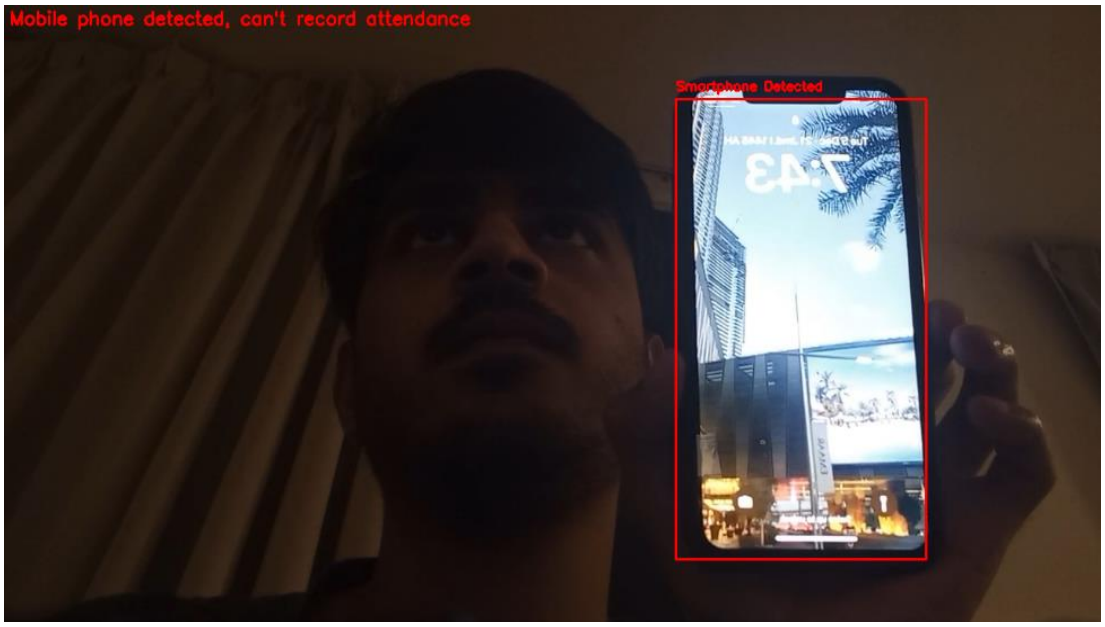
Additionally, the system has an 'event_timeout' parameter set to 60 seconds. This means that all the necessary conditions must be fulfilled within this timeframe, adding another layer of security and ensuring prompt authentication.

Once all these requirements are met the system captures a picture, thereby confirming the person's real presence. This multi-tiered approach, combining facial recognition, hand presence detection, and object detection, ensures a robust and secure anti-spoofing system.

3.1.1 Architecture Diagram of Anti-Spoofing



3.1.3 Performance of Anti-Spoofing:



3.2 Face Recognition

Image Input and Facial Recognition:

Central to the system is its ability to process the image of a person's face as input. This functionality is realized through a camera feed integrated into the user interface. The system constantly captures live images from this feed, ensuring a continuous flow of input data. For facial recognition, the system utilizes TensorFlow and a pre-trained model to generate unique facial embeddings. These embeddings are vital for comparing and identifying individuals against the database, ensuring that registered individuals are accurately recognized and verified.

Registration and Identity Verification:

The system adeptly manages the registration of new individuals and the verification of existing identities. When a face is presented, it first checks for a match in the database. A successful match leads to identity verification, while the absence of a match triggers the registration process. This registration involves capturing the individual's image and assigning a unique identifier, thus adding them to the database which is our “Person” folder. This dual functionality of recognizing existing users and registering new ones is essential for an efficient attendance system.

Graphical User Interface (GUI):

A significant aspect of the system is its Graphical User Interface (GUI), developed using customtkinter. The GUI is designed to be user-friendly and intuitive, allowing easy navigation and interaction with the system's functionalities. It includes elements like buttons for registration, check-in, and a video frame that displays the live feed. This interface plays a crucial role in simplifying user interaction with the system, making it accessible to users with varying levels of technical proficiency.

Anti-Spoofing Integration:

Security is a paramount concern in attendance systems, and this script addresses it through the integration of an Anti-Spoofing System. This system is critical in distinguishing real faces from non-real inputs such as photographs or videos. Its presence ensures the integrity and reliability of the attendance data, safeguarding against potential fraudulent activities.

Continuous Feedback and System Operation:

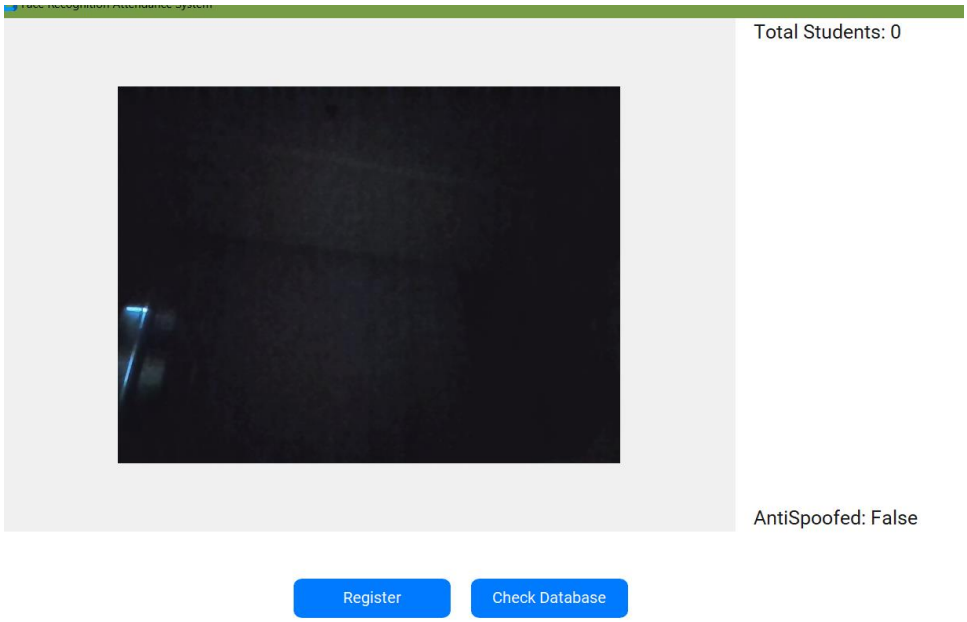
The system is designed to operate in real-time, providing continuous feedback through the GUI. This feedback includes the status of the anti-spoofing checks, identification of recognized individuals, and confirmation of registrations. Such real-time operation and feedback are crucial for the dynamic environment in which attendance systems typically function.

Summary:

In summary, the Python script for the Face Recognition Attendance System effectively fulfills the set requirements. It capably handles facial image inputs, performs accurate identity verification and registration, integrates essential security measures, and provides an accessible and user-friendly interface. These features collectively contribute to a robust, secure, and efficient attendance system.

The use of images captured by the anti-spoofing model significantly enhances the system's security and reliability. By utilizing the same image for both anti-spoofing checks and facial recognition, the system streamlines its process and fortifies its defence against potential security breaches. This approach ensures that the image being analysed for attendance is the same one vetted for authenticity, thereby reducing the likelihood of fraudulent activities such as using photographs or videos to mimic a real person. Such a mechanism not only consolidates the verification process but also has higher level of trust and reliability in the system. By integrating anti-spoofing and facial recognition in this

manner, the system stands out as more efficient, secure, and user-friendly, addressing key concerns in the domain of digital identity verification.



4.0 Conclusion

The self-supervised learning model shows suspiciously perfect accuracy, suggesting overfitting or validation errors. The supervised CNN model presents more realistic and expected learning progress, with training accuracy higher than validation accuracy, indicative of genuine learning and better generalization. Therefore, for tasks like comparing facial similarities, the supervised CNN is preferable, as it's likely to be more robust and reliable in real-world applications.