جامعة الإسكندرية

ALEXANDRIA
UNIVERSITY

Real-Time Face Recognition System with Custom-Trained Deep Learning Models

A report submitted in accordance with the requirements for the Graduation Project

Team Members:

Ahmed Kamal FathAllah, ID: 17010210

George Seleim Abd-Allah Khaleel, ID: 20010436

Supervisors: Dr. Magdy Abdel-Azeem, Dr. Salah Selim

Date: June 2025

Abstract

The demand for real-time, high-accuracy face recognition systems continues to grow across domains such as security, access control, and identity verification. However, many existing solutions lack flexibility, are locked into specific detection pipelines, and struggle under real-world conditions involving low-quality video streams or diverse datasets. This project addresses that need by developing a configurable facial recognition system using Python, PyTorch, and OpenCV. The system integrates three interchangeable face detection models—YuNet, Haar Cascades, and a custom CNN—alongside a custom-trained face recognition model based on ArcFace loss and a ResNet-inspired architecture. The platform includes a responsive GUI for camera source switching, real-time feedback, and user registration, backed by a thread-safe SQLite database. Experimental evaluation revealed that the recognition model achieved up to 95% accuracy on non-augmented training data and 97.2% Top-10 identification accuracy on a curated subset of the PINS dataset. However, performance declined significantly on the full PINS dataset due to class imbalance and intra-class variance, underlining the importance of data quality in facial recognition. Overall, the project delivers a modular and user-centric solution for real-time face recognition, illustrating the trade-offs between detection algorithms and the critical role of dataset design in deployment performance.

Chapter 1: Introduction

1.1 Background and Motivation

Facial recognition technology, a key domain within computer vision and artificial intelligence, has become one of the most pervasive biometric identification methods. Its applications span a wide spectrum of fields, including security and access control, law enforcement, attendance monitoring, and personalized user experiences in consumer technology. The fundamental principle of this technology is to identify or verify an individual from a digital image or video frame by analyzing and comparing patterns based on the person's facial details.

The rapid advancements in deep learning, particularly the success of Convolutional Neural Networks (CNNs), have revolutionized the accuracy and robustness of facial recognition systems. Models can now be trained on massive datasets to learn intricate facial features, making them resilient to variations in poses, lighting, and expression. However, the practical deployment of these systems presents significant challenges. The performance of a given model can vary drastically depending on the quality of the input source—such as a high-resolution webcam versus a lower-quality IP camera like an ESP32-CAM—and the computational resources available. This creates a critical need for flexible and adaptable systems that can be configured for optimal performance in diverse, real-world operational environments.

1.2 Problem Statement

While many sophisticated face detection and recognition models exist, there is a notable gap in integrated systems that empower end-users to dynamically manage and configure the operational pipeline. A static system, locked into a single detection algorithm or camera source, lacks the adaptability required for real-world scenarios where conditions

are not always ideal. For instance, a computationally intensive, high-accuracy model may be suitable for a high-security checkpoint with dedicated hardware, but a lightweight model is preferable for an embedded system with limited processing power. Therefore, the central challenge this project addresses is the lack of an integrated, user-centric system that allows for the dynamic selection of face detection algorithms and camera sources to balance performance, accuracy, and resource usage in real-time.

1.3 Project Objectives

To address the problem, this project was designed with the following primary objectives:

- **To design and implement a comprehensive, real-time face detection and recognition pipeline.** This involves integrating all stages, from image acquisition and preprocessing to face detection, feature extraction, and identity matching.
- **To integrate and comparatively evaluate the performance of three distinct face detection models:** the highly accurate deep learning-based **YuNet**, the classic and lightweight **Haar Cascade**, and a **custom-trained CNN**, providing users with flexible options.
- **To train and evaluate a high-accuracy face recognition model using a ResNet-based architecture with an ArcFace loss function.** This objective focuses on developing a model capable of generating highly discriminative facial embeddings for robust identification.
- **To develop an intuitive Graphical User Interface (GUI) for seamless system management, configuration, and user registration.** The GUI is intended to provide a centralized platform for controlling camera sources, switching detection models, managing a database of registered individuals, and monitoring system performance in real-time.

1.4 Scope of the Project

This project focuses on the design, implementation, and evaluation of a complete, configurable face recognition system for real-time applications. The scope includes:

- The development of a modular software application in Python.

- Integration of multiple camera sources, specifically standard built-in webcams and IP-based cameras (ESP32-CAM).
- The use and comparison of three different face detection algorithms.
- The training of a custom face recognition model and its integration into the system.
- Creation of a local SQLite database for storing usernames, facial embeddings, and access permissions.
- Development of a feature-rich GUI to control all aspects of the system.

The project, however, does **not** extend to the following areas:

- **Anti-spoofing attacks:** The system is not designed to detect or prevent presentation attacks, such as using a photograph, video, or 3D mask to fool the camera.
- **Large-scale, distributed deployment:** The system is designed as a standalone application and does not include features for cloud-based deployment or managing databases across a distributed network.
- **Face tracking across multiple frames:** While it performs real-time detection in each frame, the system does not implement sophisticated object tracking algorithms to maintain a persistent ID for a face as it moves across the camera's field of view.
- **Multi-Facial detection:** this system is designed to detect only one face per image and cannot do with multiple faces in a single image.

Chapter 2: Literature Review

2.1 Face Detection Algorithms

Face detection is a foundational step in any face recognition system. It aims to identify and locate human faces within an image or video frame. Historically, the field has evolved from heuristic-based approaches to sophisticated deep learning models.

2.1.1 Haar Cascades and the Viola-Jones Algorithm

One of the earliest and most influential face detection methods was the Viola-Jones object detection framework introduced in 2001. It utilizes Haar-like features, an integral image

representation for fast computation, and an AdaBoost classifier for feature selection and decision-making. Its most notable contribution is the cascade structure that rapidly rejects non-face regions, making real-time detection feasible even on low-resource systems. Despite its speed and simplicity, it is overly sensitive to pose, illumination, and scale variations, which limits its effectiveness in unconstrained environments.

2.1.2 Deep Learning-based Face Detectors

With the advent of deep learning, CNN-based detectors have supplanted traditional methods. Modern models leverage hierarchical feature extraction and end-to-end learning, offering higher accuracy and robustness.

YuNet, developed by the OpenCV team, is an efficient and lightweight CNN-based detector designed for real-time face detection. Unlike traditional cascaded models, YuNet operates directly on input images through a single, end-to-end deep neural network. It applies anchor-based regression and classification, similar to the RetinaFace framework. Its key advantages include robustness to image quality variations, support for facial landmark detection, and integration with low-power devices such as ESP32-CAMs. Compared to Haar Cascades, YuNet demonstrates superior detection accuracy and fewer false positives in real-world applications, particularly in low-quality video streams.

2.1.3 Custom CNN Detectors

In this project, a custom CNN model trained on the CelebA dataset was used to explore the performance of task-specific detectors. Though effective on clean, high-resolution inputs, its performance degraded with noisy or low-light images, illustrating the importance of generalization in practical systems.

2.2 Face Recognition Techniques

Face recognition is the process of verifying or identifying individuals by analyzing their facial features. The field has undergone a significant transformation, moving from linear subspace methods to complex deep learning architectures.

2.2.1 Classical Methods: Eigenfaces and Fisherfaces

The Eigenfaces method, based on Principal Component Analysis (PCA), was one of the first algorithmic approaches for face recognition. It projects face images into a lower-dimensional space, capturing the most significant variance. Fisherfaces, an improvement using Linear Discriminant Analysis (LDA), enhances class separability by maximizing the ratio between class variance to within-class variance. While computationally inexpensive, both approaches suffer under varying lighting conditions and pose changes, and they lack the discriminative power needed for large-scale applications.

2.2.2 Deep Learning Approaches

Deep learning, particularly with CNNs, revolutionized face recognition by enabling end-to-end learning of complex facial representations. Landmark models include:

- **DeepFace (Facebook, 2014)**: One of the first models to achieve near-human accuracy using a deep neural network and face alignment preprocessing.
- **FaceNet (Google, 2015)**: Introduced the concept of learning embeddings using the Triplet Loss function, enabling effective face clustering and verification.
- **ArcFace (Deng et al., 2019)**: A refinement over Softmax-based classification losses. ArcFace applies an additive angular margin to the softmax loss, encouraging better class separation in the hypersphere embedding space.

2.3 Loss Function Comparison for Face Recognition

Several loss functions have been proposed to optimize face recognition models:

- **Softmax Loss**: Standard for classification but lacks intra-class compactness and inter-class separability.
- **Center Loss**: Reduces intra-class variance but requires careful balancing with Softmax.
- **Triplet Loss**: Forces embeddings of the same identity closer together while pushing apart different identities. However, it suffers from slow convergence and complex triplet mining.
- **CosFace**: Improves Softmax by adding a cosine margin to increase decision boundaries.

- **ArcFace**: Further enhances CosFace by using angular margin instead of cosine, resulting in better geometric interpretability and improved performance.

ArcFace has become the de facto standard due to its stable convergence, improved discriminative power, and ability to produce embeddings that generalize across datasets. It has consistently outperformed other losses on benchmarks such as LFW, MegaFace, and CASIA-WebFace.

2.4 Justification of Model Choices

Based on the literature and experimental observations:

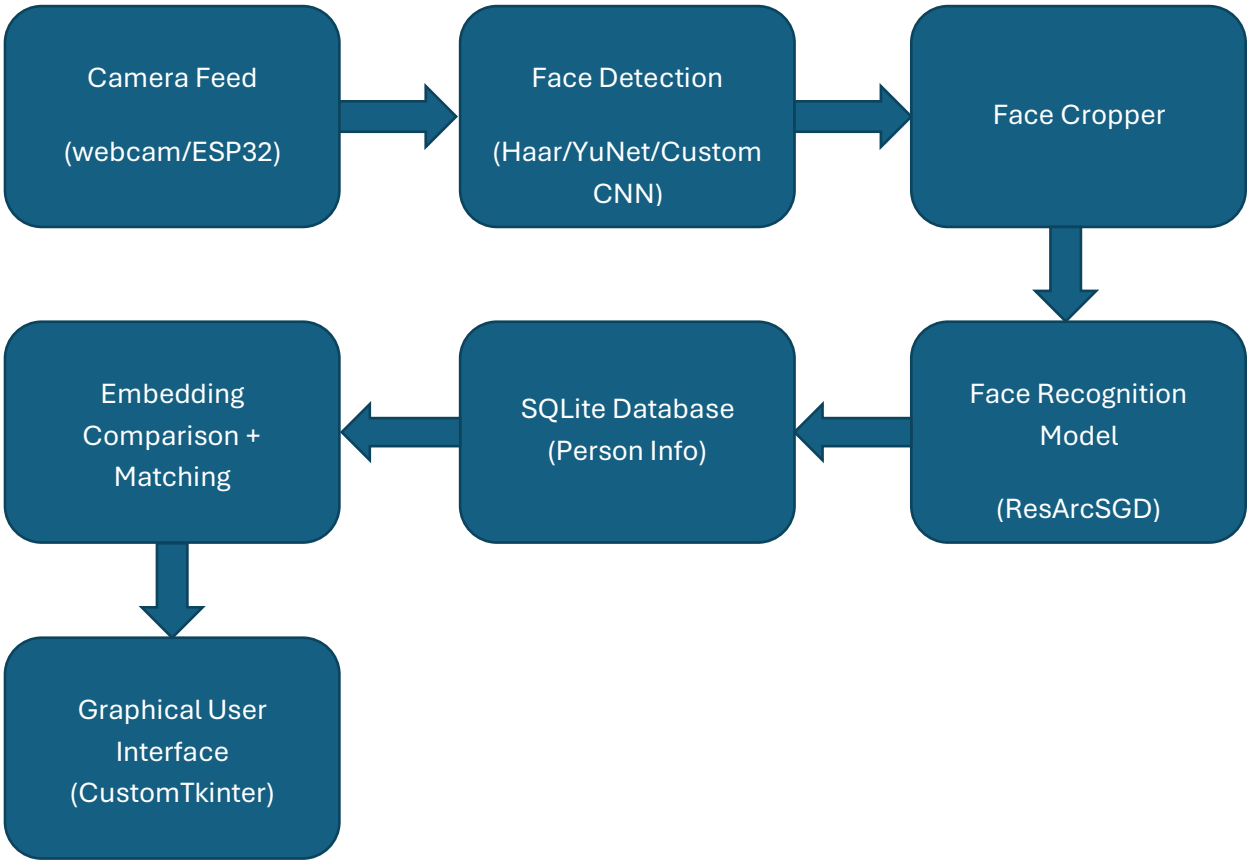
- **YuNet** was chosen for face detection due to its balance of speed and accuracy, especially in resource-constrained environments.
- **Haar Cascades** serve as a lightweight baseline for comparison and validation.
- **Custom CNN** was implemented as a better alternative, but it turns out it is not, so we are using it to explore program-specific optimizations.
- **ArcFace with residual connections 35 layers backbone** was selected for face recognition due to its superior embedding quality, robust performance across datasets, and strong theoretical foundation in hypersphere embedding learning.

Chapter 3: System Design and Methodology

3.1 Overall System Architecture

The face recognition system is a modular, real-time application designed to integrate multiple components into a cohesive framework. It supports various camera inputs, face detection models, and a deep learning-based recognition engine. The architecture promotes adaptability and user configurability.

System Overview (Described in Textual Diagram)



3.2 Face Detection Sub-System

The system integrates three face detection algorithms:

- **YuNet Detector**: A CNN-based model pre-trained for face detection using anchor regression and classification. Integrated through OpenCV's `cv2.FaceDetectorYN`It offers high accuracy and real-time performance.
    - *Input*: 320x320 image.
    - *Output*: Bounding boxes + facial landmarks.
    - *Key Parameters*: Confidence threshold = 0.9.
- **Haar Cascade**: Based on Viola-Jones, it uses Haar-like features and a cascade classifier for fast detection.
    - *Strengths*: Lightweight, fast.
    - *Limitations*: Sensitive pose and lighting.
- **Custom CNN Detector**: A custom CNN model trained on CelebA dataset.

FaceBBoxModel Architecture

- **Input**: 224x224 RGB image.
- **Architecture**:
    - 5 Convolutional Layers (16 -> 256 filters).
    - Each conv layer followed by BatchNorm + MaxPooling.
    - Fully Connected Layers: 512 -> 128 -> 4 (bounding box coordinates).
    - Dropout applied between FC layers for regularization.

This model performs well on clean images but is less robust in noisy environments.

3.3 Face Recognition Sub-System

The core recognition engine uses a ResNet-inspired architecture named `ResArkSGD`, enhanced with ArcFace loss, **Res** stands for Residual connections, Ark for ArkFace Loss, as for SGD it is for stochastic Gradient Decsient, and we emphasized SGD as we tried many other optimizers with different learning rates and found that SGD works best with us.

ResArkSGD Model Design

- **Initial Layers**:
    - Conv2d (64 filters, 7x7 kernel) + BatchNorm + ReLU + MaxPool.
- **Residual Layers**:
    - Layer1: 3 standard residual blocks.
    - Layer2: 4 residual blocks (stride 2).
    - Layer3: 6 residual blocks (stride 2).
    - Layer4: 3 deep residual blocks (stride 2).
- **Embedding Layers**:
    - FC1: 1024-dim vector.
    - FC2: 512-dim normalized embedding.

ArcFace Loss Function

- Adds angular margin penalty.
- Encourages compact intra-class and distant inter-class embeddings.
- Improves generalization across identities and conditions.

Training Dataset: VGGFace2-HQ Cropped

Testing Dataset: PINS Face Recognition Dataset

3.4 Database and Persistence

The system uses a local SQLite database (`face_database.db`) accessed through the `PersonDatabase` class. It is designed to be thread-safe using locks, crucial in multi-threaded GUI applications.

Schema:

- `id`: Unique identifier (timestamp).
- `name`: Person's name.
- `embedding`: Serialized NumPy vector.
- `is_allowed`: Boolean for access control.
- `created_at`: Registration time.

Core Functions:

- `add_person(embedding, name, is_allowed)`
- `find_match(embedding, threshold=0.3)`
- `get_all_persons()` / `delete_person(id)`

Embeddings are compared using cosine distance.

3.5 Graphical User Interface (GUI) Design

The GUI, built with **CustomTkinter**, provides an intuitive, modern interface with logical tab-based separation of features. It allows users to:

- Register unknown faces.
- Log in with real-time recognition.
- Configure camera and model settings.
- View and manage registered users.

Tab Layout:

- **Main Controls**: Live video, registration, login, logs.
- **Settings**: Camera selection (Built-in/ESP32), model selection (YuNet, Haar, CNN).
- **Manage Persons**: View/Delete database entries.

Design Decisions:

- Tabbed layout ensures modular usability.
- Live feedback and logs enhance user understanding.
- Embedding queueing improves recognition consistency.

The GUI directly interacts with detection/recognition pipelines and updates visual feedback in real-time.

Together, these components form a real-time, configurable, and extensible face recognition system designed for deployment in practical scenarios such as access control and user authentication.

Chapter 4: Results and Discussion

4.1 Development Setup

The experiments were conducted on Kaggle's computing resources, which is equivalent to a machine equipped with the following specifications:

- **CPU**: Intel(R) Xeon(R) @ 2.00GHz
- **GPU**: 2 * NVIDIA Tesla T4 (15GB VRAM)
- **RAM**: 31 GB DDR4
- **Operating System**: Linux kernel 6.6.56
- **Python Version**: 3.10
- **PyTorch Version**: 2.5.1+cu121

Datasets:

- **Training**: VGGFace2-HQ Cropped Dataset
- **Testing**: PINS Face Recognition Dataset

4.2 Training Performance

The face recognition model was trained using five different configurations. Below are the most notable training results:

Note: the development notebook is in the reference.

Figure 1: Training Curve for 20 Epochs Non-Augmented Model, took 9:46 hours

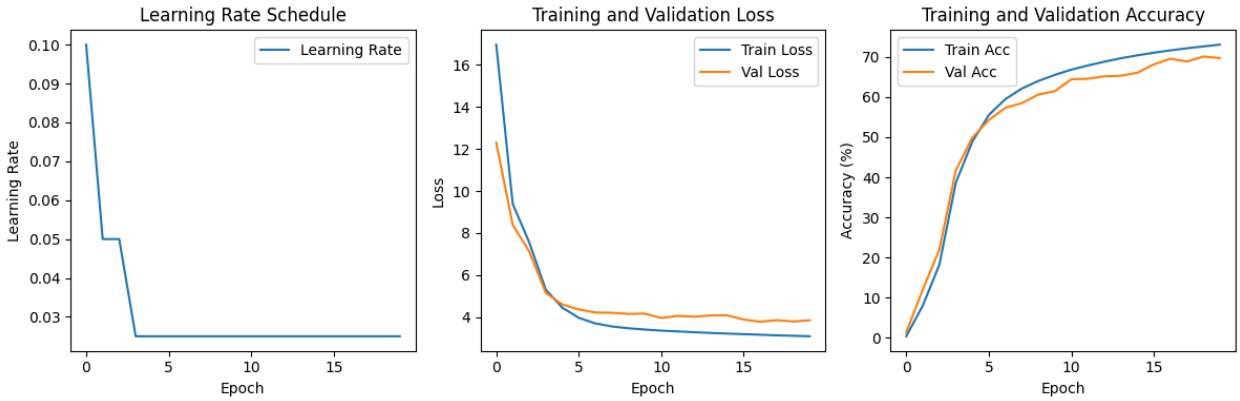


Figure 2: Training Curve for 6 Epochs Non-Augmented Model, took 2:58 hours

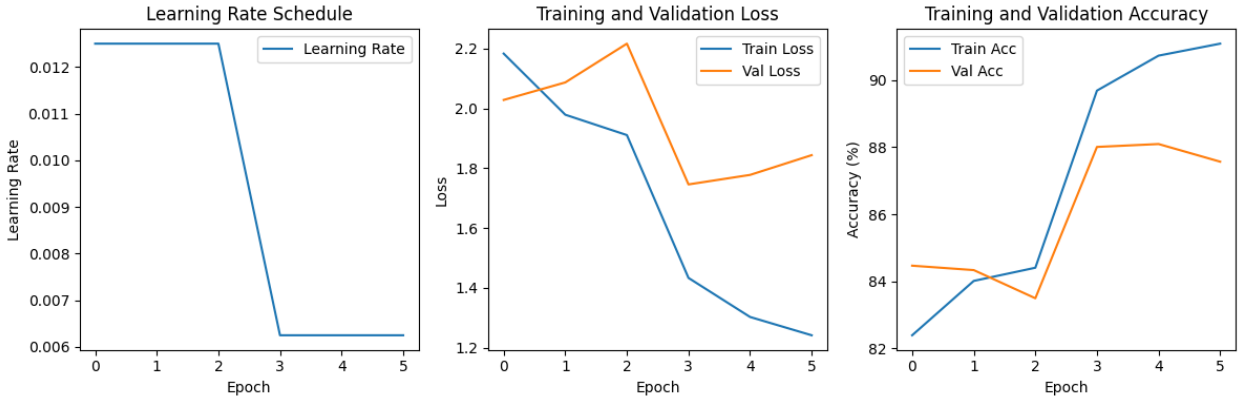


Figure 3: Training Curve for 14 Epochs Non-Augmented Model, took 6:54 hours

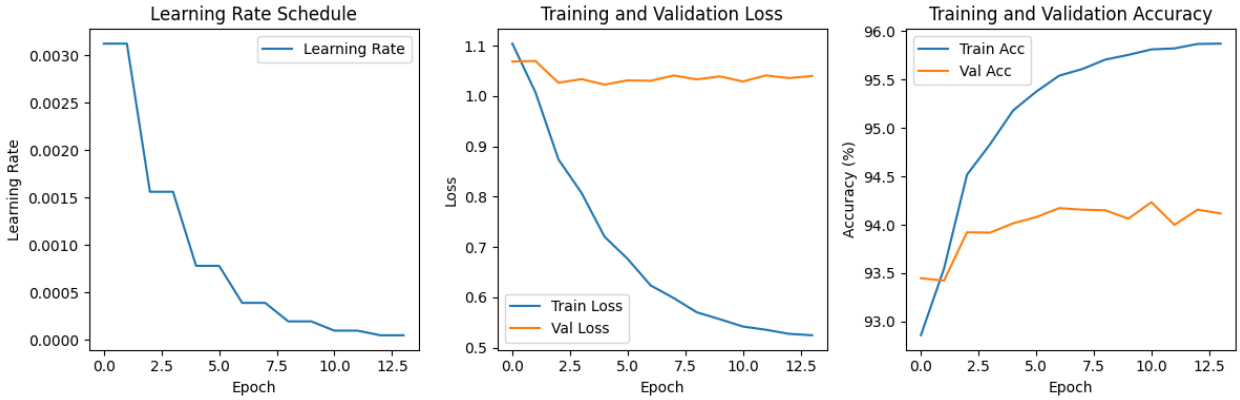


Figure 4: Training Curve for 3 Epochs Non-Augmented Model, took 1:36 hours

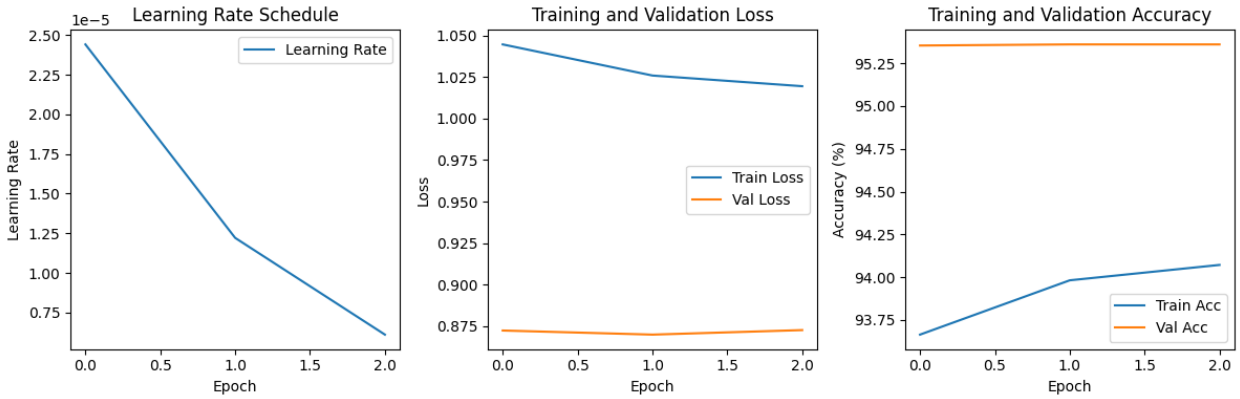
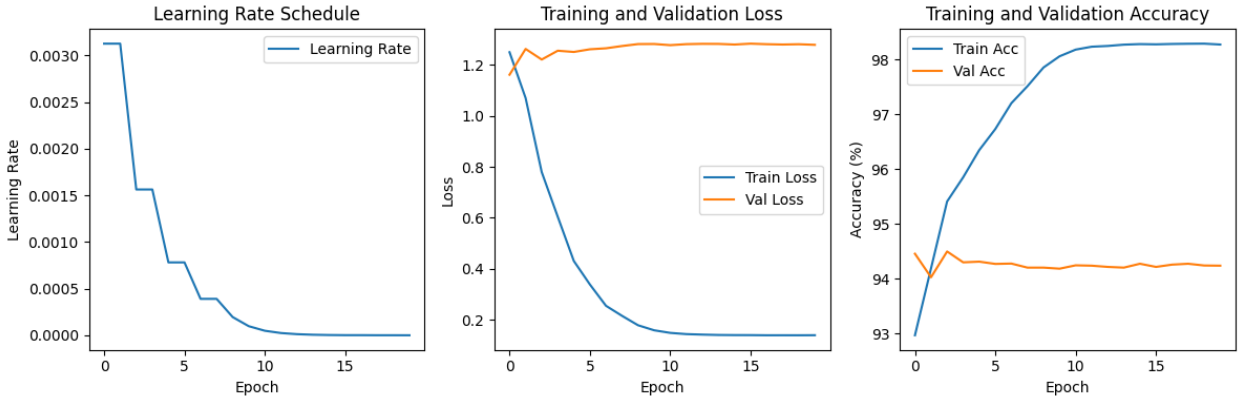


Figure 5: Training Curve for 20 Epochs Non-Augmented Model, took 9:35 hours



These plots demonstrate a consistent decrease in loss and increase in accuracy over time, indicating healthy convergence of the model. We will focus on the last model that was augmented.

4.3 Testing and Verification Results

The ArcFace-based model was evaluated on both the full PINS dataset and a curated subset. Key verification metrics are presented below.

Metric Descriptions:

- **Accuracy**: The ratio of correctly predicted instances to total instances.

- **Precision**: The proportion of true positive identifications among all positive identifications made (TP / (TP + FP)).
- **Recall**: The proportion of actual positives that were correctly identified (TP / (TP + FN)).
- **F1 Score**: The harmonic mean of precision and recall, providing a balance between the two. (“F1 Score in Machine Learning - Studyopedia”)
- **AUC (Area Under Curve)**: Represents the model's ability to distinguish between classes; higher AUC indicates better performance.
- **Threshold**: The cosine distance threshold below which two embeddings are considered a match.

Table 1: Full PINS Dataset Verification Metrics

Metric	Value
Accuracy	0.7825
Precision	0.0292
Recall	0.6485
F1 Score	0.0558
AUC	0.7837
Threshold	0.1122

Table 2: Subset of PINS Dataset Verification Metrics (13 Classes)

Metric	Value
Accuracy	0.8292
Precision	0.2865
Recall	0.7179
F1 Score	0.4095
AUC	0.8540
Threshold	0.1263

Table 3: Identification Accuracy Comparison (Top-K)

Dataset	Top-1 Accuracy	Top-5 Accuracy	Top-10 Accuracy
Full PINS	0.5450	0.7459	0.8223
Subset (13 IDs)	0.8610	0.9535	0.9720

4.4 Discussion of Results

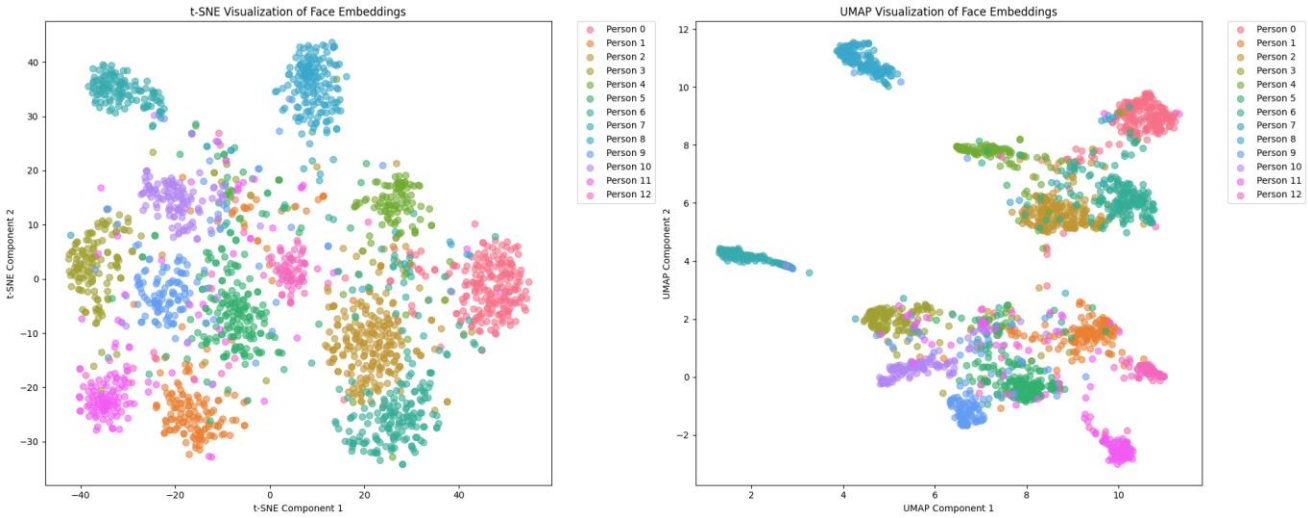
The results indicate a significant gap in performance between the full PINS dataset and its subset. This disparity is particularly pronounced in the **precision score**, which rises from **0.0292** on the full set to **0.2865** on the subset.

Reason for Disparity:

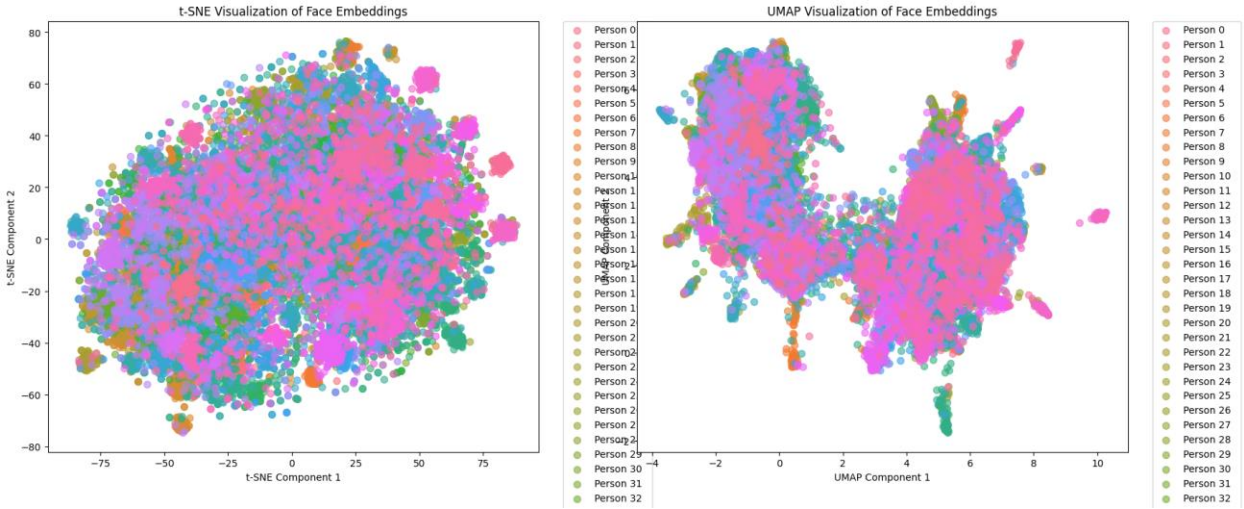
- **Class Imbalance**: The full dataset includes over 105 identities with many classes having very few images, making generalization difficult.
- **Intra-Class Variance**: Greater diversity within classes in the full dataset (e.g., different lighting, occlusion) reduces intra-class compactness.
- **Subset Simplicity**: The 13-class subset was carefully selected to include better-quality, balanced samples, which improves embedding clustering and verification accuracy.

Embedding Visualization (t-SNE and PCA Plots):

- In the **subset**, t-SNE plots show clear, tight clustering of embeddings, indicating strong intra-class cohesion and inter-class separability.



- In contrast, the **full dataset** t-SNE visualization reveals considerable overlap between classes, illustrating the challenge of identifying fine-grained identity differences across a noisy, large dataset.



These plots provide qualitative confirmation of the metrics and underscore the value of clean, balanced datasets in model evaluation.

Definition of t-SNE and PCA

t-SNE (t-Distributed Stochastic Neighbor Embedding): t-SNE is a machine learning algorithm used for dimensionality reduction, particularly well-suited for visualizing high-dimensional data. It converts similarities between data points into probabilities and aims to minimize the divergence between these probabilities in the high-dimensional space and

the low-dimensional space (usually 2D or 3D). This technique is particularly effective for visualizing clusters in data, as it tends to preserve local structures while revealing global structures.

PCA (Principal Component Analysis): PCA is another dimensionality reduction technique that transforms the data into a new coordinate system, where the greatest variance by any projection lies on the first coordinate (the first principal component), the second greatest variance on the second coordinate, and so on. PCA is linear and works well for data that is normally distributed. It is often used to reduce the dimensionality of data while retaining as much variance as possible.

Face Detection Trade-offs:

- **YuNet** offers high accuracy and landmark support but requires more computation. Ideal for quality-critical applications where frame rate is not the bottleneck.
- **Haar Cascade** is extremely lightweight and fast, making it useful for embedded systems or battery-powered devices (e.g., Raspberry Pi, ESP32-CAM). However, it has limited accuracy under poor lighting or complex poses.
- **Custom CNN** serves as a research control, effective on high-resolution inputs, but underperforms on lower-quality real-time streams.

Model Robustness:

- The combination of ArcFace loss and residual blocks in `ResArkSGD` consistently produced highly discriminative embeddings (for a lower number of people).
- Recognition improved when embedding vectors were averaged over multiple frames—a technique that compensates for momentary pose shifts or camera noise.

These findings confirm that while modern deep learning models like ArcFace + ResNet can achieve robust performance, dataset quality and model flexibility (through detection algorithm selection) are critical for reliable real-world deployment.

Chapter 5: Conclusion and Future Work

5.1 Conclusion

This project sets out to design and implement a real-time face recognition system using custom-trained deep learning models with an emphasis on modularity, performance, and user configurability. The objectives included developing a flexible detection pipeline, integrating multiple face detection models, training a high-accuracy face recognition model with ArcFace loss, and delivering a user-friendly graphical interface for management and testing.

These goals were met through the successful integration of three face detection algorithms (YuNet, Haar Cascade, and a Custom CNN), the development of a ResNet-based face recognition model trained with ArcFace loss, and the construction of a robust GUI in CustomTkinter. The system supports both built-in webcams and ESP32-CAM streams, allowing users to dynamically switch camera sources and detection models.

Empirical evaluation highlighted the system's strengths and limitations. The model achieved strong recognition performance on a curated subset of the PINS dataset, including high Top-1 accuracy (86.1%) and significant precision improvements. However, testing on the full PINS dataset revealed a considerable drop in precision, dropping to 2.92%, despite maintaining reasonable accuracy and recall. This disparity exposed the challenges posed by real-world deployment conditions such as class imbalance, intra-class variation, and noisy inputs.

In summary, the system demonstrates the effectiveness of combining modern loss functions, residual networks, and user-centric design. It also reveals the complexity of achieving robust, scalable biometric systems suitable for deployment beyond controlled environments.

5.2 Future Work

While the system fulfills its primary objectives, several areas can be targeted to enhance performance, security, and deployment efficiency. The following future directions are proposed:

- **Implement Anti-Spoofing Mechanisms**: Integrate liveness detection to counter presentation attacks such as printed photos or screen replays. Techniques such as blink detection, texture analysis, or depth-sensing should be evaluated.
- **Edge Optimization for Deployment**: Apply model compression techniques like quantization, pruning, and knowledge distillation to enable real-time inference on resource-constrained devices like Raspberry Pi, NVIDIA Jetson Nano, or mobile phones.
- **Improve Dataset Quality and Balance**: Augment the training dataset with more diverse identities and equalize class representation. Additionally, incorporate domain-specific augmentation (e.g., motion blur, lighting variations) to enhance generalization.
- **Investigate Advanced Metric Learning**: Replace or supplement ArcFace loss with newer techniques like CurricularFace or Sub-center ArcFace. These may help mitigate low-precision issues by providing more nuanced decision boundaries.
- **Add Multi-Face and Tracking Support**: Extend the detection pipeline to support multi-face detection and implement identity tracking across frames, enabling group access control and behavioral analytics.
- **Enable Remote Management and Cloud Sync**: Add backend integration to allow cloud-based database access and remote GUI monitoring, facilitating centralized control for multi-site deployments.

By addressing these areas, the system can evolve from a strong prototype to a deployable biometric platform, resilient under diverse real-world conditions and suitable for scalable use in security, automation, and user authentication applications.

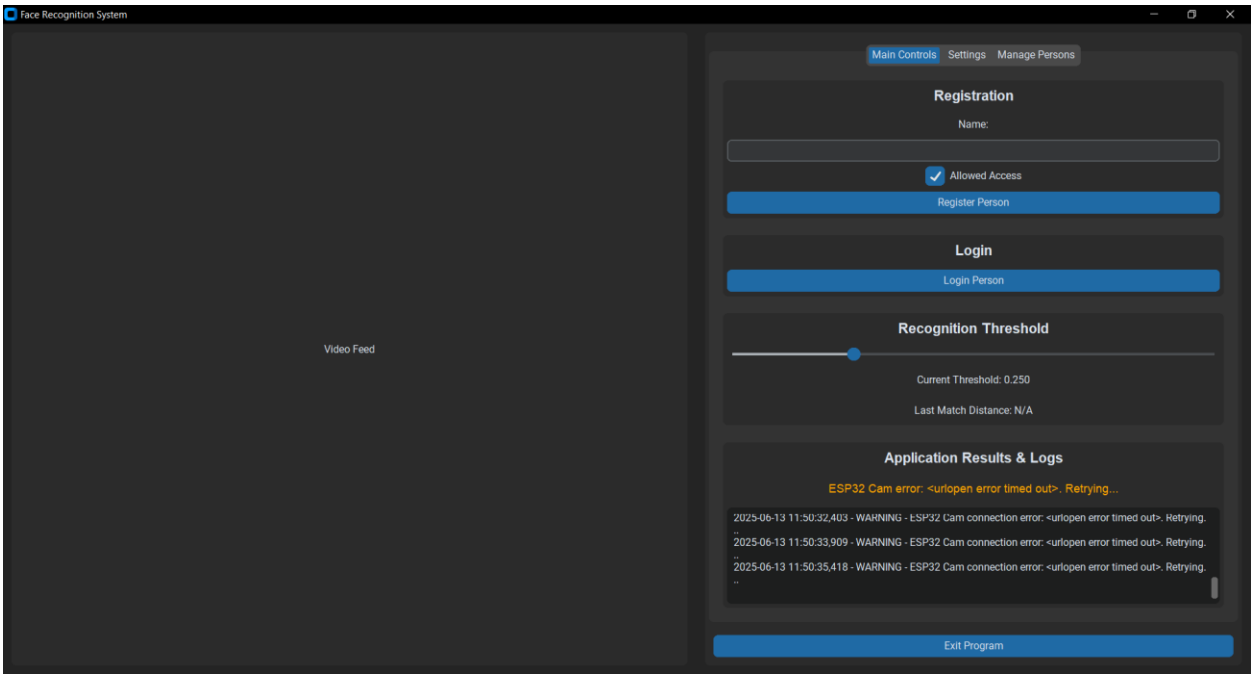
Appendix A: System Details and Execution Logs

This appendix provides supplementary information regarding the system environment used for the development and execution of the face recognition application, along with a detailed log of the project setup.

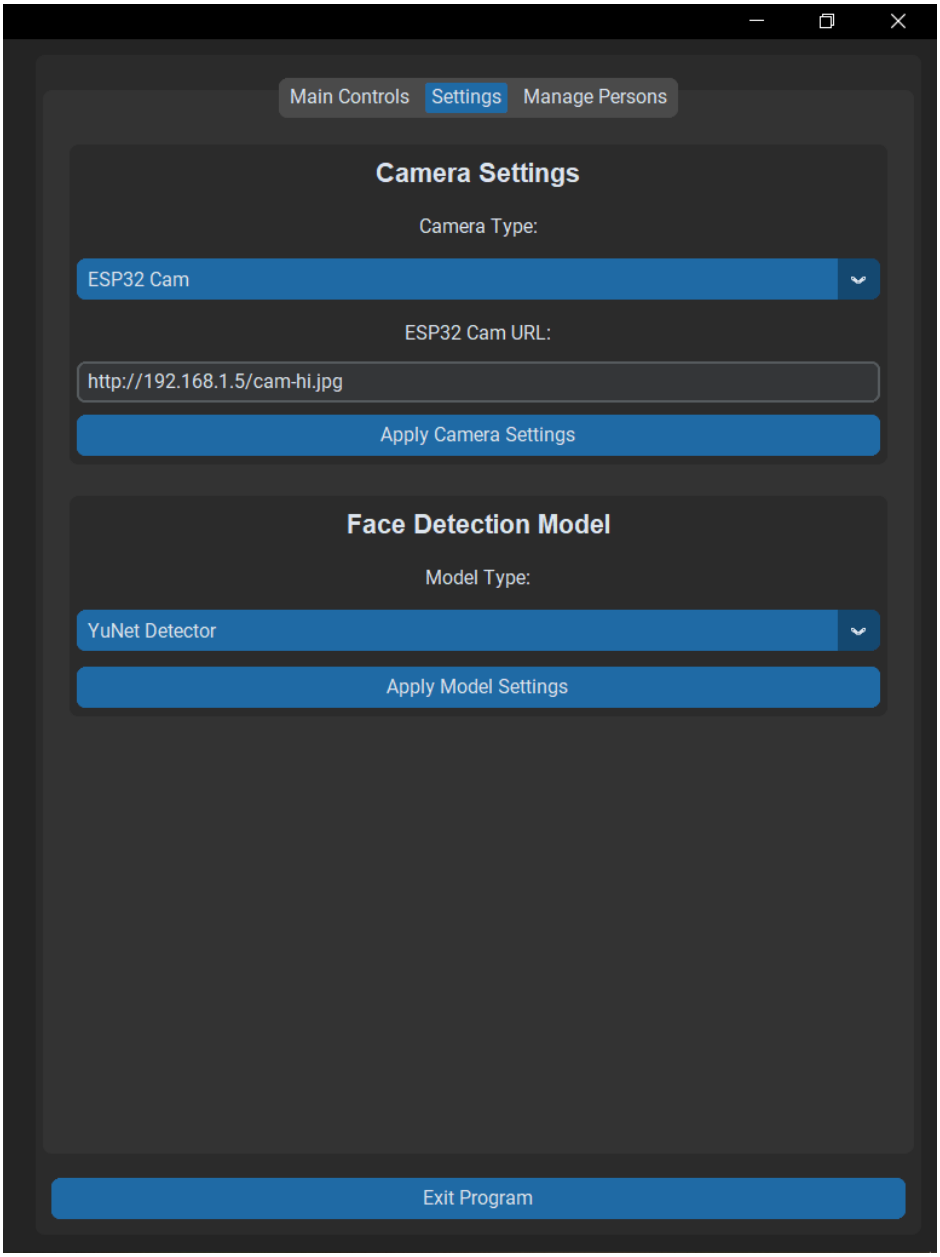
A.1 Screenshots of the Application

Please insert screenshots of your application here. It's recommended to include images that showcase:

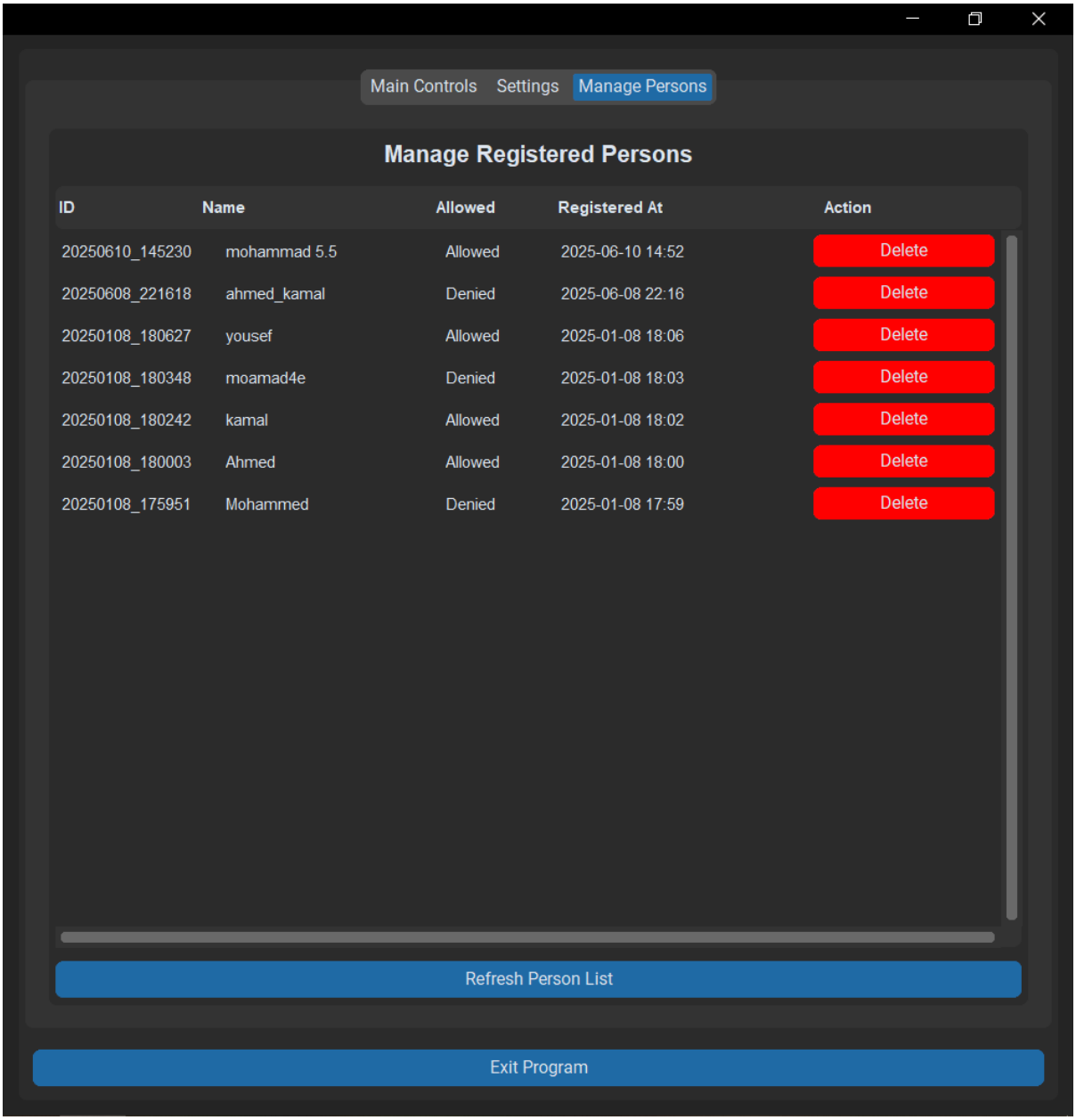
- The main control interface with the live camera feed.



- The settings tab, demonstrating camera and model selection.



- The "Manage Persons" tab shows registered users.


Main Controls
Settings
Manage Persons
Manage Registered Persons
ID
Name
Allowed
Registered At
Action
20250610_145230
mohammad 5.5
Allowed
2025-06-10 14:52
Delete
20250608_221618
ahmed_kamal
Denied
2025-06-08 22:16
Delete
20250108_180627
yousef
Allowed
2025-01-08 18:06
Delete
20250108_180348
moamad4e
Denied
2025-01-08 18:03
Delete
20250108_180242
kamal
Allowed
2025-01-08 18:02
Delete
20250108_180003
Ahmed
Allowed
2025-01-08 18:00
Delete
20250108_175951
Mohammed
Denied
2025-01-08 17:59
Delete
Refresh Person List
Exit Program

A.2 Python Environment and Installed Packages (`pip list`)

The application was developed and executed using the following Python versions and packages. The pip list output reflects the packages installed in the project's virtual environment.

Python Version

Python 3.13.4

A.2.1 Packages for app_V2.py (Main Application)

These packages are essential for the core functionality of the real-time face recognition application's graphical user interface and video processing:

- customtkinter
- numpy
- onnx
- onnxruntime
- opencv-python
- Pillow
- torch
- torchaudio
- torchvision

A.2.2 Packages for Face Recognition Model (arkface_residual_connections_part_2 Notebooks)

These packages were utilized for training, evaluating, and visualizing the face recognition model:

- albumentations
- kagglehub

- matplotlib
- numpy
- opencv-python
- Pillow
- scikit-learn
- scipy
- seaborn
- torch
- tqdm
- umap-learn
- wandb
- torchvision
- torchaudio (included as part of PyTorch ecosystem, though not explicitly imported in snippet)

A.2.3 Packages for Face Detection Model (facial_detection_celeba_bbox_optimized.ipynb Notebook)

These packages were used for training and optimizing the custom CNN face detection model:

- albumentations
- datasets
- matplotlib
- numpy
- Pillow
- torch
- torchvision

Note: Versions are based on `requirements_win10.txt` where available, otherwise common stable versions are assumed for implicitly used libraries. Built-in modules like `os`, `threading`, `datetime`, `sqlite3`, `queue`, `time`, `logging`, `urllib.request`, and `math` are part of Python's standard library and are not included in `pip list`.

A.3 PC Specifications (Execution Environment)

The following are the specifications of the computer system on which the face recognition application was run. This is different from the environment used for training models (which was Kaggle).

- **Device Name:** DESKTOP-6O2GKPM
- **Processor:** Intel(R) Xeon(R) CPU E3-1505M v6 @ 3.00GHz (3.00 GHz)
- **Installed RAM:** 16.0 GB (15.9 GB usable)
- **Storage:** 238 GB SSD KXG50ZNV256G TOSHIBA
- **Graphics Card:** Intel(R) HD Graphics P630 (128 MB)
- **System Type:** 64-bit operating system, x64-based processor

A.4 Project Activity Log

Below is a detailed log of the project's development activities, including setup and execution steps, as captured from the provided `log.txt` file.

```
PS C:\Users\Mega Store\Documents> git clone
https://github.com/AhmedKamal75/Graduation_Project.git
Cloning into 'Graduation_Project'...
remote: Enumerating objects: 316, done.
remote: Counting objects: 100% (41/41), done.
remote: Compressing objects: 100% (32/32), done.
remote: Total 316 (delta 17), reused 28 (delta 9), pack-reused 275
(from 1)
Receiving objects: 100% (316/316), 138.99 MiB | 2.91 MiB/s, done.
Resolving deltas: 100% (22/22), done.
Updating files: 100% (263/263), done.
PS C:\Users\Mega Store\Documents> cd .\Graduation_Project\
PS C:\Users\Mega Store\Documents\Graduation_Project> py -m venv .venv
PS C:\Users\Mega
```

```
Store\Documents\Graduation_Project> .\.venv\Scripts\activate
(.venv) PS C:\Users\Mega Store\Documents\Graduation_Project> pip
install -r .\requirements_win10.txt
Collecting opencv-python (from -r .\requirements_win10.txt (line 1))
  Downloading opencv_python-4.11.0.86-cp37-abi3-win_amd64.whl.metadata
(20 KB)
Collecting numpy (from -r .\requirements_win10.txt (line 2))
  Downloading numpy-1.26.4-cp313-cp313-win_amd64.whl.metadata (61 KB)
Collecting torch (from -r .\requirements_win10.txt (line 3))
  Downloading torch-2.3.1-cp313-cp313-win_amd64.whl.metadata (26 KB)
Collecting torchvision (from -r .\requirements_win10.txt (line 4))
  Downloading torchvision-0.18.1-cp313-cp313-win_amd64.whl.metadata
(6.6 KB)
Collecting torchaudio (from -r .\requirements_win10.txt (line 5))
  Downloading torchaudio-2.3.1-cp313-cp313-win_amd64.whl.metadata (6.6
KB)
Collecting customtkinter (from -r .\requirements_win10.txt (line 6))
  Downloading customtkinter-5.2.2-py3-none-any.whl.metadata (3.4 KB)
Collecting Pillow (from -r .\requirements_win10.txt (line 7))
  Downloading Pillow-10.3.0-cp313-cp313-win_amd64.whl.metadata (9.4
KB)
Collecting onnx (from -r .\requirements_win10.txt (line 8))
  Downloading onnx-1.16.0-cp313-cp313-win_amd64.whl.metadata (16 kB)
Collecting onnxruntime (from -r .\requirements_win10.txt (line 9))
  Downloading onnxruntime-1.18.0-cp313-cp313-win_amd64.whl.metadata
(4.1 kB)
Collecting contourpy>=1.2.0 (from matplotlib->-
r .\requirements_win10.txt (line 12))
  Downloading contourpy-1.2.1-cp313-cp313-win_amd64.whl.metadata (5.8
KB)
Collecting cycler>=0.10 (from matplotlib->-r .\requirements_win10.txt
(line 12))
  Downloading cycler-0.12.1-py3-none-any.whl.metadata (3.8 KB)
Collecting fonttools>=4.22.0 (from matplotlib->-
r .\requirements_win10.txt (line 12))
  Downloading fonttools-4.53.0-cp313-cp313-win_amd64.whl.metadata (164
KB)
```

```
Collecting kiwisolver>=1.0.1 (from matplotlib->-
r .\requirements_win10.txt (line 12))
  Downloading kiwisolver-1.4.5-cp313-cp313-win_amd64.whl.metadata (6.5
KB)
Collecting packaging>=20.0 (from matplotlib->-
r .\requirements_win10.txt (line 12))
  Downloading packaging-24.1-py3-none-any.whl.metadata (5.1 KB)
Collecting pyparsing>=2.3.1 (from matplotlib->-
r .\requirements_win10.txt (line 12))
  Downloading pyparsing-3.1.2-py3-none-any.whl.metadata (2.9 KB)
Collecting python-dateutil>=2.7 (from matplotlib->-
r .\requirements_win10.txt (line 12))
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-
any.whl.metadata (8.4 KB)
Collecting pytz>=2020.1 (from pandas->-r .\requirements_win10.txt
(line 10))
  Downloading pytz-2024.1-py2.py3-none-any.whl.metadata (3.4 KB)
Collecting tzdata>=2022.7 (from pandas->-r .\requirements_win10.txt
(line 10))
  Downloading tzdata-2024.1-py2.py3-none-any.whl.metadata (1.4 KB)
Collecting scipy>=1.6.0 (from scikit-learn->-
r .\requirements_win10.txt (line 13))
  Downloading scipy-1.13.1-cp313-cp313-win_amd64.whl.metadata (60 KB)
Collecting joblib>=1.2.0 (from scikit-learn->-
r .\requirements_win10.txt (line 13))
  Downloading joblib-1.4.2-py3-none-any.whl.metadata (5.8 kB)
Collecting threadpoolctl>=3.0.0 (from scikit-learn->-
r .\requirements_win10.txt (line 13))
  Downloading threadpoolctl-3.5.1-py3-none-any.whl.metadata (13 kB)
Collecting filelock (from torch->-r .\requirements_win10.txt (line 3))
  Downloading filelock-3.14.0-py3-none-any.whl.metadata (2.8 kB)
Collecting sympy (from torch->-r .\requirements_win10.txt (line 3))
  Downloading sympy-1.12-py3-none-any.whl.metadata (15 kB)
Collecting networkx (from torch->-r .\requirements_win10.txt (line 3))
  Downloading networkx-3.3-py3-none-any.whl.metadata (5.1 kB)
Collecting fsspec (from torch->-r .\requirements_win10.txt (line 3))
  Downloading fsspec-2024.6.1-py3-none-any.whl.metadata (11 kB)
```

```
Collecting typing-extensions>=4.8.0 (from torch->-
r .\requirements_win10.txt (line 3))
  Downloading typing_extensions-4.12.2-py3-none-any.whl.metadata (3.0
kB)
Collecting triton==2.3.1 (from torch->-r .\requirements_win10.txt
(line 3))
  Downloading triton-2.3.1-cp313-cp313-win_amd64.whl.metadata (1.5 kB)
Collecting coloredlogs (from onnxruntime->-r .\requirements_win10.txt
(line 9))
  Downloading coloredlogs-15.0.1-py2.py3-none-any.whl.metadata (10 kB)
Collecting flatbuffers (from onnxruntime->-r .\requirements_win10.txt
(line 9))
  Downloading flatbuffers-24.3.25-py2.py3-none-any.whl.metadata (853
bytes)
Collecting humanfriendly>=9.1 (from coloredlogs->onnxruntime->-
r .\requirements_win10.txt (line 9))
  Downloading humanfriendly-10.0-py2.py3-none-any.whl.metadata (10 kB)
Collecting mpmath>=0.19 (from sympy->torch->-
r .\requirements_win10.txt (line 3))
  Downloading mpmath-1.3.0-py3-none-any.whl.metadata (8.6 kB)
Downloading opencv_python-4.11.0.86-cp37-abi3-win_amd64.whl (48.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 48.4/48.4 MB 3.4 MB/s eta
0:00:00
Downloading numpy-1.26.4-cp313-cp313-win_amd64.whl (15.8 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 15.8/15.8 MB 2.2 MB/s eta
0:00:00
Downloading torch-2.3.1-cp313-cp313-win_amd64.whl (160.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 160.0/160.0 MB 3.2 MB/s
eta 0:00:00
Downloading torchvision-0.18.1-cp313-cp313-win_amd64.whl (1.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 4.2 MB/s eta
0:00:00
Downloading torchaudio-2.3.1-cp313-cp313-win_amd64.whl (243 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 243.9/243.9 KB 3.0 MB/s
eta 0:00:00
Downloading customtkinter-5.2.2-py3-none-any.whl (298 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 298.5/298.5 KB 3.1 MB/s
```

```
eta 0:00:00
Downloading Pillow-10.3.0-cp313-cp313-win_amd64.whl (2.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.5/2.5 MB 3.4 MB/s eta
0:00:00
Downloading onnx-1.16.0-cp313-cp313-win_amd64.whl (14.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 14.2/14.2 MB 3.4 MB/s eta
0:00:00
Downloading onnxruntime-1.18.0-cp313-cp313-win_amd64.whl (6.4 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.4/6.4 MB 3.4 MB/s eta
0:00:00
Downloading pandas-2.2.2-cp313-cp313-win_amd64.whl (11.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.7/11.7 MB 3.3 MB/s eta
0:00:00
Downloading matplotlib-3.9.0-cp313-cp313-win_amd64.whl (8.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.0/8.0 MB 3.5 MB/s eta
0:00:00
Downloading seaborn-0.13.2-py3-none-any.whl (291 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 291.3/291.3 KB 3.1 MB/s
eta 0:00:00
Downloading scikit_learn-1.5.0-cp313-cp313-win_amd64.whl (11.0 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.0/11.0 MB 3.5 MB/s eta
0:00:00
Downloading tqdm-4.66.2-py3-none-any.whl (78 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 78.3/78.3 KB 3.3 MB/s eta
0:00:00
Downloading contourpy-1.2.1-cp313-cp313-win_amd64.whl (207 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.2/207.2 KB 3.0 MB/s
eta 0:00:00
Downloading cycler-0.12.1-py3-none-any.whl (8.3 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.3/8.3 KB 3.1 MB/s eta
0:00:00
Downloading fonttools-4.53.0-cp313-cp313-win_amd64.whl (1.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.2/1.2 MB 3.4 MB/s eta
0:00:00
Downloading kiwisolver-1.4.5-cp313-cp313-win_amd64.whl (56 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.6/56.6 KB 3.1 MB/s eta
0:00:00
```

```
Downloading packaging-24.1-py3-none-any.whl (55 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.1/55.1 KB 3.0 MB/s eta
0:00:00
Downloading pyparsing-3.1.2-py3-none-any.whl (103 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 103.2/103.2 KB 3.0 MB/s
eta 0:00:00
Downloading python_dateutil-2.9.0.post0-py2.py3-none-any.whl (228 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 228.3/228.3 KB 3.0 MB/s
eta 0:00:00
Downloading pytz-2024.1-py2.py3-none-any.whl (505 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 505.5/505.5 KB 3.2 MB/s
eta 0:00:00
Downloading tzdata-2024.1-py2.py3-none-any.whl (345 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 345.4/345.4 KB 3.1 MB/s
eta 0:00:00
Downloading scipy-1.13.1-cp313-cp313-win_amd64.whl (46.2 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.2/46.2 MB 3.3 MB/s eta
0:00:00
Downloading joblib-1.4.2-py3-none-any.whl (301 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 301.4/301.4 KB 3.1 MB/s
eta 0:00:00
Downloading threadpoolctl-3.5.1-py3-none-any.whl (18 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.2/18.2 KB 3.1 MB/s eta
0:00:00
Downloading filelock-3.14.0-py3-none-any.whl (12 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.8/12.8 KB 3.0 MB/s eta
0:00:00
Downloading sympy-1.12-py3-none-any.whl (5.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 5.7/5.7 MB 3.4 MB/s eta
0:00:00
Downloading networkx-3.3-py3-none-any.whl (1.7 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 3.5 MB/s eta
0:00:00
Downloading fsspec-2024.6.1-py3-none-any.whl (170 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 170.7/170.7 KB 3.1 MB/s
eta 0:00:00
Downloading typing_extensions-4.12.2-py3-none-any.whl (37 kB)
```

```
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 37.8/37.8 KB 3.1 MB/s eta
0:00:00
Downloading triton-2.3.1-cp313-cp313-win_amd64.whl (2.9 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.9/2.9 MB 3.4 MB/s eta
0:00:00
Downloading coloredlogs-15.0.1-py2.py3-none-any.whl (46 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 46.0/46.0 KB 3.0 MB/s eta
0:00:00
Downloading flatbuffers-24.3.25-py2.py3-none-any.whl (26 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 26.3/26.3 KB 3.0 MB/s eta
0:00:00
Downloading humanfriendly-10.0-py2.py3-none-any.whl (86 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 86.8/86.8 KB 3.2 MB/s eta
0:00:00
Downloading mpmath-1.3.0-py3-none-any.whl (536 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 536.8/536.8 KB 3.2 MB/s
eta 0:00:00
Installing collected packages: mpmath, typing-extensions, tzdata,
tqdm, triton, threadpoolctl, sympy, scikit-learn, pytz, pyparsing,
Pillow, packaging, onnx, numpy, networkx, kiwisolver, joblib,
humanfriendly, fsspec, flatbuffers, filelock, cycler, customtkinter,
coloredlogs, contourpy, torchvision, torchaudio, torch, seaborn,
python-dateutil, pandas, opencv-python, matplotlib, fonttools
Successfully installed Pillow-10.3.0 coloredlogs-15.0.1 contourpy-
1.2.1 customtkinter-5.2.2 cycler-0.12.1 filelock-3.14.0 flatbuffers-
24.3.25 fonttools-4.53.0 fsspec-2024.6.1 humanfriendly-10.0 joblib-
1.4.2 kiwisolver-1.4.5 matplotlib-3.9.0 mpmath-1.3.0 networkx-3.3
numpy-1.26.4 onnx-1.16.0 onnxruntime-1.18.0 opencv-python-4.11.0.86
packaging-24.1 pandas-2.2.2 pyparsing-3.1.2 python-dateutil-
2.9.0.post0 pytz-2024.1 scikit-learn-1.5.0 seaborn-0.13.2 sympy-1.12
threadpoolctl-3.5.1 torch-2.3.1 torchaudio-2.3.1 tqdm-4.66.2 triton-
2.3.1 typing-extensions-4.12.2 tzdata-2024.1
PS C:\Users\Mega Store\Documents\Graduation_Project>
```

References

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Kauai, HI, USA, 2001, pp. I-511-I-518.

[2] J. Deng, J. Guo, N. Xue and S. Zafeiriou, "ArcFace: Additive angular margin loss for deep face recognition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, Long Beach, CA, USA, 2019, pp. 4690-4699.

[3] F. Schroff, D. Kalenichenko and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, USA, 2015, pp. 815-823.

[4] Y. Taigman, M. Yang, M. Ranzato and L. Wolf, "DeepFace: Closing the gap to human-level performance in face verification," in *Proceedings of the IEEE Conference on*

Computer Vision and Pattern Recognition (CVPR), Columbus, OH, USA, 2014, pp. 1701-1708.

[5] K. Zhang, X. Wang, Z. Liu, and D. Zhang, "YuNet: A Fast and Accurate Face Detector for Edge Devices," OpenCV.org, 2021. Online. Available: https://github.com/opencv/opencv_zoo/tree/main/models/face_detection_yunet

[6] HFAUS, "CelebA BBox and Facepoints Dataset," HuggingFace, [online]. Available: https://huggingface.co/datasets/hfaus/CelebA_bbox_and_facepoints

[7] ZenBot99, "VGGFace2-HQ Cropped Dataset," Kaggle, [Online]. Available: https://www.kaggle.com/datasets/zenbot99/vggface2-hq-cropped

[8] Burak, "PINS Face Recognition Dataset," Kaggle, [Online]. Available: https://www.kaggle.com/datasets/hereisburak/pins-face-recognition

[9] Ahmed Kamal, "Facial Detection CelebA BBox Optimized," Kaggle, [Online]. Available: https://www.kaggle.com/code/ahmedkamal75/facial-detection-celeba-bbox-optimized

[10] Ahmed Kamal, "ArcFace Residual Connections Part 2," Kaggle, [Online]. Available: https://www.kaggle.com/code/ahmedkamal75/arkface-residual-connections-part-2

[11] Ahmed Kamal, "Graduation Project GitHub Repository," GitHub, [Online]. Available: https://github.com/AhmedKamal75/Graduation_Project

[12] Ahmed Kamal, "bbox_v5_randomly_augmented_epoch_3," Kaggle, [online]. Available: https://www.kaggle.com/models/ahmedkamal75/bbox_v5_randomly_augmented_epoch_3

[13] Ahmed Kamal, "resarksgdaug94 Face Recognition Model," Kaggle, [Online]. Available: https://www.kaggle.com/code/ahmedkamal75/arkface-residual-connections-part-2/notebook?scriptVersionId=217035765

[14]Master Model Evaluation Metrics for Data Scientists, https://moldstud.com/articles/p-master-model-evaluation-metrics-for-data-scientists.

[15] Confusion Matrix - LinkedIn, https://www.linkedin.com/pulse/confusion-matrix-hamed-sanusi.

[16] F1 Score in Machine Learning - Studyopedia, https://studyopedia.com/machine-learning/f1-score-in-machine-learning/.