Assignment 2 Report

Prayash Kumar Sahu (22B1261), Aditya Singh Bhadoria (22B1247)

1 System Architecture

1.1 Components

- **DeepFace (Facenet):** Generates face embeddings for recognition using the `Facenet` model.
- **OpenCV:** Handles camera capture, image processing (resizing, augmentation), frame manipulation, and real-time face detection (using its built-in detector).
- **NumPy:** Performs all numerical computations, including creating a trusted "centroid" embedding, calculating cosine similarity, and managing embedding arrays.
- **TensorFlow:** Serves as the underlying deep learning framework supporting DeepFace operations (v2.13.0).
- **Image Augmentation:** A pre-processing step creates robust embeddings by generating variations of source images, including horizontal flipping, brightness adjustments ($\times 0.8, \times 1.2$), rotations ($\pm 10°$), and scaling ($\times 0.9, \times 1.1$).
- **Trusted Centroid:** The system does not store individual trusted embeddings for real-time comparison. Instead, it computes a single "average" embedding (centroid) from all augmented trusted faces.
- **Auto-Calibrated Threshold:** The classification boundary is not hardcoded. It is dynamically calculated as the midpoint between the average similarity of trusted faces to the centroid and the average similarity of random faces to the centroid.
- **Unknown Face Logging:** When an "UNKNOWN" face is detected, the system saves a cropped image of the face to the `unknown_faces/` directory with a timestamp and a 10-second cooldown period to prevent spam.

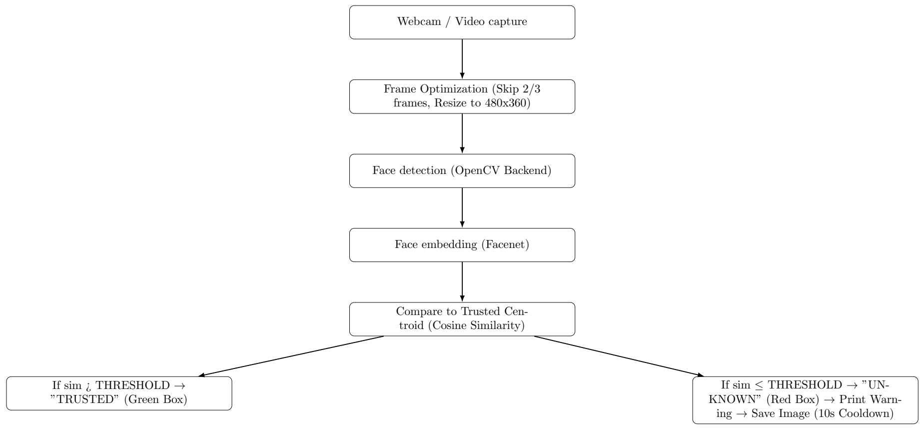

Figure 1: Flow diagram of the system architecture

2 Integration Challenges and Solutions

1. **Challenge:** Facial recognition is highly sensitive to variations in lighting, head pose, and distance, which can lead to false negatives for trusted users.

   Solution: Implemented a robust pre-processing pipeline. Instead of a single embedding, the system generates multiple augmentations (rotation, scale, brightness, flip) for each trusted image, computes embeddings for all, and averages them into a single, highly robust vector. This "mean" embedding is much more resilient to real-world conditions.

2. **Challenge:** High-accuracy face detectors like `RetinaFace` are too computationally expensive for smooth, real-time webcam processing.

   Solution: A hybrid detector strategy was used. The slow, high-accuracy `RetinaFace` detector is used *only* for the one-time, offline pre-processing of trusted and random faces. For the real-time application, the system switches to the much faster `opencv` backend for face detection.

3. **Challenge:** Real-time video processing (detection + embedding) can still cause significant lag, even with a faster detector.

   Solution: Two performance optimizations were added to the main loop:

   - **Frame Resizing:** The input frame is immediately resized to a small 480x360 resolution, drastically reducing the pixels the detector needs to scan.
   - **Frame Skipping:** The system processes only one out of every three frames (`frame_count % 3`), effectively tripling the potential FPS by reducing the processing load.

4. **Challenge:** Manually selecting a fixed similarity threshold (e.g., 0.6) is unreliable and may not be optimal for the specific faces enrolled in the system.

   Solution: The system uses an auto-calibrated threshold. It calculates the similarity of known trusted faces and known random faces (from the `random_faces` folder) against the trusted centroid, then sets the threshold as the midpoint between these two groups: $(mean(trusted\_sims) + mean(random\_sims))/2$.

3 Ethical Considerations and Testing Results

3.1 Ethical Considerations

The deployment of this system involves handling sensitive biometric data, raising key ethical points:

- **Privacy and Data Storage:** The system stores face embeddings (numerical representations) in `embeddings.npz`. More critically, it actively captures and saves images of "UNKNOWN" individuals to the `unknown_faces` directory. This requires a clear policy for data retention, access control, and deletion.
- **Informed Consent:** Individuals whose images are placed in the `trusted_faces` folder must provide explicit consent for their biometric data to be used for authentication.
- **Bias and Fairness:** The `Facenet` model, like all facial recognition models, may have performance biases across different demographics. The use of image augmentation (brightness, scaling) in the pre-processing step is a technique that can help mitigate some of these biases by creating a more generalized embedding.
- **Proportionality of Response:** The system's response to an "UNKNOWN" detection is relatively passive: a console warning (`"You are not authorized!! Please Leave!"`) and logging the face. This is a less intrusive security measure than a loud alarm or immediate authority notification.

3.2 Testing Results

The notebook provides execution output from the pre-processing and initialization stages, which serve as the system's baseline configuration and calibration:

- **Database Configuration:** The pre-processing script (Cell 7) was executed with a database of 1 trusted face (which was augmented to generate multiple embeddings) and 5 random faces.

- **Auto-Calibrated Threshold:** Based on the 1 trusted and 5 random face embeddings, the real-time script (Cell 8) successfully auto-calibrated its classification boundary. The calculated cosine similarity threshold was **0.619**.
- **System Status:** The real-time detection script successfully initialized the webcam and began monitoring, confirming that all components (OpenCV, DeepFace, NumPy) were loaded correctly.
- **Performance Features:** The code demonstrates the implementation of robustness features (augmentation, mean embeddings) and real-time performance optimizations (frame skipping, resizing) in its design, though no quantitative latency or accuracy metrics (e.g., FPS, FRR/FAR) are provided in the notebook.

4 Instructions to Run Code

4.1 Prerequisites

- Python 3.8 or higher.
- A functional webcam connected to the system.
- Required Python packages: `deepface`, `opencv-python`, `speechrecognition`, `pyttsx3`, `pyaudio`, `tensorflow`.

4.2 Setup and Installation

1. Create and activate a Python virtual environment (recommended).
2. Install all required packages using pip:
   `pip install deepface opencv-python speechrecognition pyttsx3 pyaudio tensorflow`
3. Create three folders in the same directory as the notebook: `trusted_faces/`, `random_faces/`, and `unknown_faces/`.
4. Place at least one high-quality image (JPG, PNG, WEBP) of a trusted person in the `trusted_faces/` folder.
5. Place several images of different, non-trusted people in the `random_faces/` folder. These are used to calculate the threshold.

4.3 Execution

The notebook is divided into two main parts that must be run in order.

1. **Generate Embeddings:** Run the code cell containing the `augment_image`, `get_embedding`, and `compute_all` functions (Cell 7). This script will process all images in `trusted_faces/` and `random_faces/` and save the results into a new file named `embeddings.npz`.

2. **Run Real-Time Verification:** After `embeddings.npz` has been created, run the next code cell (Cell 8). This script will:

   - Load the embeddings.
   - Calculate the auto-calibrated threshold (e.g., 0.619).
   - Open the webcam and display the "Face Verification" window.

3. **Operation:** The system will draw a green "TRUSTED" box around recognized faces and a red "UNKNOWN" box around others. Unknown faces will be saved to the `unknown_faces/` folder.

4. **To Stop:** Press the `'q'` key while the "Face Verification" window is active.