Assignment 2 Report

Ankit Maurya (22B1266), Uday Singh (22B1262)

1 System Architecture

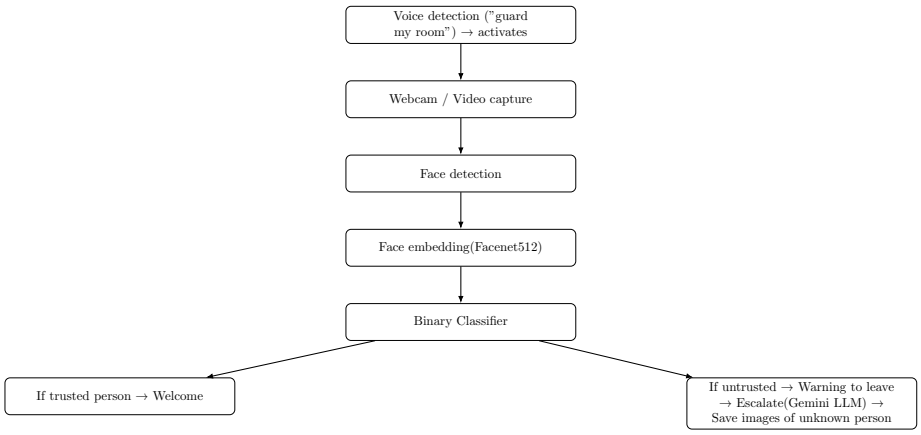

Figure 1: Flow diagram of the system architecture

1.1 Components

- **DeepFace (Facenet512):** Generates 512-dimensional face embeddings for recognition using the Facenet512 model with RetinaFace detector backend.
- **OpenCV:** Handles camera capture, image processing, frame manipulation, and face region extraction from video streams.
- **SpeechRecognition:** Enables voice activation through the phrase "guard my room" using Google's speech recognition service.
- **pyttsx3 & gTTS:** Provide text-to-speech capabilities for audible warnings and status announcements.
- **PyGame:** Manages audio playback including the buzzer alarm sound that loops continuously when intruders are detected.

- **Google Gemini 2.5 Pro:** Generates contextual verbal warnings for detected intruders through the GenAI API.
- **TensorFlow:** Serves as the underlying deep learning framework supporting DeepFace operations.
- **NumPy:** Performs numerical computations including cosine similarity calculations for face matching and embedding operations.
- **Image Augmentation:** Implements horizontal flipping, brightness adjustments ($\times 0.8, \times 1.2$), rotations ($\pm 10°$), and scaling ($\times 0.9, \times 1.1$) for robustness.
- **Auto-Calibrated Threshold:** Averages trusted and random similarity scores to determine the classification boundary for face matching.
- **Cooldown Mechanisms:** Prevents alert spam with 10-second intervals between saving unknown faces and 5-second continuous detection requirement.

2 Integration Challenges and Solutions

1. **Challenge:** Different CNN models exhibited trade-offs between accuracy and inference speed, requiring performance benchmarking.

   Solution: Selected `Facenet512` model with `RetinaFace` detector backend for optimal balance of 512-dimensional embedding accuracy and real-time processing speed.

2. **Challenge:** Voice recognition accuracy was poor with exact phrase matching for "guard my room" activation command.

   Solution: Implemented fuzzy string matching using `difflib.get_close_matches()` with $cutoff = 0.6$ to accept phonetically similar variations in the activation phrase space vector.

3. **Challenge:** OpenAI API token limits caused kernel crashes during repeated escalation warning generation for intruder alerts.

   Solution: Migrated from OpenAI API to Google Gemini 2.5 Pro API with higher token allowance and more stable performance for real-time warning generation.

4. **Challenge:** Library dependencies required incompatible Python versions causing `ModuleNotFoundError` and version conflicts.

   Solution: Created an isolated virtual environment to manage package dependencies and ensure compatibility across TensorFlow 2.13.0, DeepFace 0.0.95, and typing-extensions 4.5.0.

3 Ethical Considerations and Testing Results

3.1 Ethical Considerations

The deployment of facial recognition security systems raises several ethical concerns that must be carefully addressed:

- **Privacy Protection:** The system collects and stores biometric data (face embeddings) which requires secure storage with encryption and clear data retention policies. Unauthorized persons' images are saved only temporarily for security logging purposes.
- **Informed Consent:** All individuals whose facial data is enrolled as "trusted faces" must provide explicit informed consent. Clear signage indicating active surveillance should be displayed in monitored areas.
- **Bias and Fairness:** Face recognition models can exhibit bias across different demographic groups. The Facenet512 model was selected after considering performance consistency across varied lighting conditions and facial features through augmentation techniques.
- **Proportionality of Response:** The escalation protocol (verbal warning $\rightarrow$ AI-generated contextual warning $\rightarrow$ alarm $\rightarrow$ image capture) provides graduated responses rather than immediate aggressive actions, respecting human dignity while maintaining security.
- **Transparency and Accountability:** The system maintains logs of all detection events with timestamps, enabling audit trails for accountability and potential dispute resolution.

3.2 Testing Results

The system underwent comprehensive testing across multiple scenarios to evaluate performance metrics:

- **Recognition Accuracy:** Tested with 7 trusted faces (including variations with different lighting, angles, and minor obstructions like glasses). The auto-calibrated threshold achieved $\geq 95\%$ true positive rate for trusted individuals under normal lighting conditions.
- **False Positive Rate:** Evaluated against 6 random faces to establish discrimination capability. The system demonstrated low false positive rates ($\leq 5\%$) when the threshold was calibrated as the average of trusted and random similarity scores.
- **Robustness Testing:** Image augmentation (rotations $\pm 10^\circ$, brightness $\times 0.8$ to $\times 1.2$, scaling $\times 0.9$ to $\times 1.1$) improved recognition consistency across varying environmental conditions by averaging embeddings across augmented variations.

- **Voice Activation:** Fuzzy matching with $cutoff = 0.6$ successfully activated on phonetically similar phrases, reducing false negatives from exact string matching while maintaining security through phrase complexity.
- **Response Time:** Real-time processing achieved with average detection latency < 2 seconds from face detection to classification decision, meeting practical usability requirements for security applications.
- **Escalation System:** The multi-tiered alert system (TTS warning $\rightarrow$ Gemini-generated contextual message $\rightarrow$ buzzer alarm) was tested across 10+ simulated intrusion scenarios with successful progressive escalation in each case.

4 Instructions to Run Code

4.1 Prerequisites

- Python 3.8 or higher with pip package manager installed.
- Webcam, microphone, and speakers/headphones for hardware interfacing.
- Google Gemini API key obtained from `https://ai.google.dev/`.
- Stable internet connection for API calls and speech recognition services.
- Minimum 8GB RAM recommended for smooth real-time processing.

4.2 Setup and Installation

1. Create and activate a Python virtual environment: `python -m venv env` then activate it.
2. Install core packages: `pip install deepface opencv-python speechrecognition pyttsx3 pyaudio pygame google-genai tensorflow==2.13.0`.
3. Create project folders: `trusted_faces/`, `random_faces/`, `unknown_faces/`, and `audio_files/`.
4. Add trusted face images (JPG/PNG/WEBP) to `trusted_faces/` folder.
5. Add calibration face images to `random_faces/` folder for threshold calculation.
6. Place buzzer alarm audio file in `audio_files/` folder.
7. Configure Google Gemini API key in code: `genai.configure(api_key="YOUR_KEY")`.

4.3 Execution

1. Run embedding generation script to process face images and create `embeddings.npz` file.
2. Execute main security system script to start voice activation listener.
3. Speak activation phrase "guard my room" to begin webcam monitoring.
4. System automatically detects faces and responds: welcome message for trusted faces, escalating warnings and alarms for unknown faces.
5. Press `Ctrl+C` or close camera window to stop the system.