AI-Based Face Detection and Recognition Web Application Using DeepFace

Abstract

This project implements a deep learning–based face detection and recognition system deployed as a web application using Streamlit. The system allows users to upload an image containing one or more human faces and automatically detects faces, extracts facial features using a pre-trained deep learning model, and identifies known individuals by comparing facial embeddings. The application leverages the DeepFace framework with the ArcFace model for feature extraction and RetinaFace for accurate face detection. The project demonstrates an end-to-end AI pipeline, from image preprocessing to face recognition and result visualization, making it suitable for real-world applications such as access control, identity verification, and attendance systems.

Problem Statement

Face recognition is a critical component in many modern applications such as security systems, surveillance, employee attendance, and identity verification. Traditional face recognition systems require complex pipelines, large datasets, and extensive model training, which can be difficult to implement and deploy, especially for small teams or rapid prototyping.

The problem addressed by this project is to **build a simple, interactive, and reliable face recognition system** that:

- Requires minimal setup
- Uses pre-trained deep learning models
- Can be easily deployed as a web application
- Handles real-world image variations such as lighting, pose, and multiple faces

Technology Stack

Technology	Purpose	Why Chosen
Python	Core programming language	Strong ecosystem for AI and ML

Streamlit	Web application framework	Rapid deployment and minimal frontend code
DeepFace	Face recognition framework	Simplifies use of state-of-the-art models
ArcFace	Face embedding model	High accuracy and robustness to variations
RetinaFace	Face detection model	Reliable detection of multiple faces
NumPy	Numerical operations	Efficient vector and matrix computation
PIL (Pillow)	Image processing	Easy image handling and drawing

Implementation Details

Face Detection

- RetinaFace is used via DeepFace.extract_faces.
- It detects faces and returns precise bounding box coordinates.
- Works well for images with multiple faces and different orientations.

Embedding Extraction

- Each detected face is passed to DeepFace.represent.
- The ArcFace model converts the face into a fixed-length numerical vector (embedding).
- These embeddings represent unique facial features.

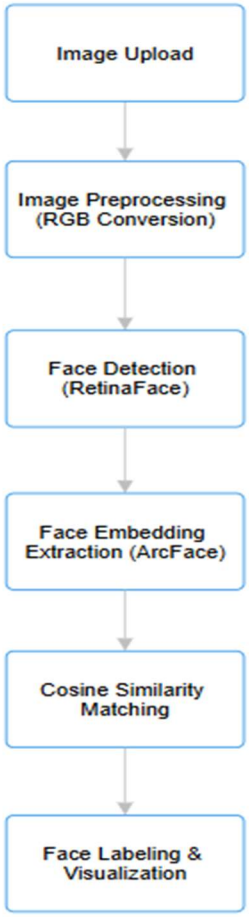
Matching Logic

- Known face embeddings are preloaded and cached.
- For each detected face, cosine distance is computed between the detected embedding and known embeddings.
- The smallest distance indicates the closest match.

Threshold Selection

- A cosine distance threshold of **0.4** is used.
- If the distance is below the threshold, the face is considered a match.
- This balances accuracy and reduces false positives.

System Architecture



Explanation

- The user uploads an image via the Streamlit UI.
- The image is converted to RGB format for model compatibility.
- RetinaFace detects all faces and their bounding boxes.
- Each detected face is passed to ArcFace to generate a numerical embedding.
- The embedding is compared with known face embeddings using cosine distance.

- The closest match below a defined threshold is labeled; otherwise, the face is marked as “Unknown”

Results

Output

- The application successfully detects one or more faces in an uploaded image.
- Bounding boxes are drawn around detected faces.
- Known faces are labeled with names; unknown faces are labeled as “Unknown”.

Examples

- Single face image → Correct identification
- Group image → Multiple faces detected and labeled
- Unknown person → Marked as “Unknown”

Observations

- ArcFace provides consistent embeddings even with slight pose or lighting variations.
- RetinaFace performs well on real-world images.

Limitations

- The system depends on preloaded known faces and does not support dynamic face registration.
- Performance may degrade for very low-resolution or blurry images.
- No liveness detection, so printed photos can be falsely recognized.
- Not optimized for real-time video or webcam input.

Future Scope

- Add webcam-based real-time face recognition.
- Implement face registration and database storage.
- Introduce liveness detection to prevent spoofing.
- Deploy the application on cloud platforms (AWS/GCP).
- Extend the system to support video input and tracking.