Face Detection Models

1. RetinaFace

RetinaFace is a high-performance face detection model that works well in many different scenarios. It offers both high accuracy and good speed, which makes it suitable for real-world use cases such as face recognition systems, biometric authentication, and even entertainment tech.

- **High Accuracy:** Can detect faces under difficult conditions such as poor lighting, occlusion, and various head angles (based on the WIDER FACE benchmark).
- **Facial Landmarks:** Detects important key points like eyes, nose, and mouth. This helps with face alignment and also supports 3D face-related tasks.
- **Efficiency:** Despite its high accuracy, the model is fast enough to be used in real-time applications.
- **Robustness:** Handles a wide range of image qualities, face sizes, and poses. It's reliable in real-world situations.

Limitation in my testing:
During implementation, I found that RetinaFace failed to detect a child's face in a side profile. Even though the face was clearly visible to the human eye, RetinaFace didn't recognize it. This could be due to the face not being frontal enough for its landmark detection module. It seems the model is more sensitive to pose variations in younger faces or those not similar to its training dataset.

2. YOLOv11n

To compare, I ran the same image using **YOLOv11n**, a lightweight version of the YOLO, this model detected the face successfully.

This may be because YOLO is trained on more diverse face types, especially in real-world environments. Its architecture is flexible and works well in edge cases, particularly after fine-tuning with a dedicated face dataset.

Face Embedding Models

ArcFace

ArcFace is one of the most widely used deep learning models for face recognition. Its main advantage is producing discriminative embeddings, which helps clearly separate identities in the feature space.

Older models typically used softmax loss, which doesn’t always ensure that embeddings from the same person are close together or that different people are far apart. ArcFace improves this using Additive Angular Margin Loss, which forces better separation between identities.

Face Recognition System: Pipeline Design

Stage 1: Face Detection

Class: HybridFaceDetector

The main objective of this stage is to detect and locate faces in an input image. A hybrid detection strategy is used to improve robustness and accuracy.

- **Primary Detector (RetinaFace):**
  The system first tries to detect faces using the RetinaFace model (buffalo_l) from the insightface library.

- **Fallback Detector (YOLOv11):**
  If RetinaFace cannot detect any faces, the system automatically switches to a YOLOv11 face detection model. YOLOv11 is more flexible in real-world conditions and increases the overall reliability of detection.

Output:
Returns a list of bounding boxes in [x1, y1, x2, y2] format, which indicates the pixel coordinates of each detected face. If no face is detected by either model, the process for that image ends.

Stage 2: Feature Extraction

Class: DeepFaceRecognitionSystem

After detecting a face, the system extracts unique facial features and converts them into a numeric representation called an embedding. This is the key part of the recognition process.

- **Model Priority (ArcFace):**
  The system is set to use the ArcFace model from the DeepFace library by default. ArcFace is known for producing highly discriminative embeddings that improve recognition performance.

- **Preprocessing:**
  The DeepFace.represent function automatically crops, aligns, and resizes the face before generating the embedding.

- **Embedding Generation:**
  The face is converted into a fixed-size vector which represents its unique features.

Output:
Returns the embedding vector along with the model name.

Stage 3: Database Construction

Method: build_database

- **Image Iteration** The system loads and processes all images from a specified directory
- **Pipeline Execution** Each image goes through Stage 1 (Detection) and Stage 2 (Feature Extraction).
- **Embedding Normalization:** Each embedding is normalized
- **Storage Format:**
    - person_id: Extracted from the image filename
    - image_path: The file path to the original image
    - model: The embedding model used
    - embedding : embedding vector

Output:
The final output is an in-memory list of these embeddings and metadata.

Stage 4: Face Recognition

Method: recognize_face

Compared with known faces from the database.

- **Query Embedding** The query image is passed through the same detection and embedding pipeline (Stages 1 and 2), and the result is normalized.
- **Similarity Calculation:** The query embedding is compared to each database entry using *cosine similarity*. This metric measures how close two vectors are:
    - Value close to 1 means high similarity
    - Value close to 0 means low similarity
- **Ranking and Thresholding:**
    - Results are sorted in descending order of similarity score
    - A threshold is used to filter out low-confidence matches
    - The top k matches that exceed the threshold are returned

Output:
A ranked list of individuals in the database that are most similar to the input face.