Face Recognition with ArcFace ONNX and 5-Point Alignment

Gabriel Baziramwabo

Benax Technologies Ltd · Rwanda Coding Academy

Table of Content

Acknowledgements

I would like to express my sincere gratitude to my family for their unwavering support throughout the development of this book. My wife's constant encouragement, thoughtful care, and countless cups of spice tea—along with her excellent cooking—created the calm and energy needed to complete this work. She also generously contributed her face identity during testing, helping validate the system in real-world conditions. My three children participated with curiosity and enthusiasm, contributing their own identities and reminding me that technology can also be a source of joy and fun.

I am equally grateful to my students at Rwanda Coding Academy, whose insatiable curiosity and persistent questioning continuously pushed me to seek a face recognition pipeline that is not only correct in theory, but stable, explainable, and practical in real-world use. Their challenges, experiments, and feedback played a significant role in shaping the design decisions and clarity of this work.

Chapter 1 — Architecture, Setup, and First Validation

1.1 Background

Face recognition is often introduced as a black box: a pre-trained model is loaded, a function is called, and an identity is returned, with little understanding of how or why the system works. While that approach may produce quick results, it hides failure modes and makes debugging, improvement, and trust nearly impossible. This book takes a different path. You will build a face recognition system step by step, where every stage is explicit, runnable, replaceable, and understandable. Rather than relying on an end-to-end framework, the system is decomposed into independent modules—face detection, landmark extraction, alignment, embedding, enrollment, and recognition—each performing a single task and testable in isolation.

In earlier work, presented in my book *Face Recognition without Deep Learning*, face recognition was implemented using *LBPH (Local Binary Patterns Histogram)* to illustrate the fundamentals of feature extraction and similarity measurement without relying on trained neural networks. While effective as a learning tool, LBPH fundamentally operates by *memorizing local texture patterns* from enrolled images rather than learning a compact, identity-discriminative embedding. As a result, even when face images are carefully cropped or aligned, LBPH struggles to generalize across pose changes, illumination variations, and natural appearance differences, and its performance degrades rapidly as the number of enrolled identities grows. These limitations stem from the representation itself, not merely from image quality. This book therefore represents a natural progression from that foundation by introducing *deep learning–based face embeddings using the ArcFace ONNX model, where ONNX (pronounced "on-eks") stands for Open Neural Network Exchang*e, an open standard that enables trained neural network models to be executed efficiently across different platforms and frameworks. Using ONNX allows ArcFace to run with CPU-only inference while remaining practical and reproducible. Because ArcFace requires well-normalized inputs, *5-point facial landmarks* are used to align faces by correcting tilt and scale before embedding extraction. In this way, you do not merely run face recognition—you build it, understand it, and control it.

1.2 What Face Recognition Really Solves

At its core, face recognition answers a single question: are these two faces likely to belong to the same person? This is not a traditional classification problem. Instead of assigning a face to one of a fixed set of classes, each face is mapped to a numerical vector, known as an embedding. Faces of the same person produce embeddings that lie close to one another in this vector space, while different faces are separated by larger distances. Recognition is therefore performed by *measuring similarity* using a distance metric rather than by *predicting a class label.* This design enables open-set recognition, allowing the system to reject identities it has never seen before without retraining.

1.3 CPU-Oriented Execution Model

This project adopts a CPU-oriented execution model. The system is designed to run efficiently on laptops, desktops, and shared laboratory machines without relying on GPU acceleration. Treating the CPU as the primary execution target enforces architectural discipline, exposes inefficiencies early, and ensures consistent behavior across macOS, Linux, and Windows. When GPU acceleration is available, it can be introduced as an optimization without changing the system's structure.

1.4 Face Alignment as a Fundamental Preprocessing Step

Raw face images vary significantly due to *head pose*, *camera angle*, *distance from the camera*, and *facial expression*. Extracting embeddings directly from such unaligned faces forces the model to compensate for irrelevant variation, which degrades recognition reliability. In this project, faces are aligned using five facial landmarks—the left eye, right eye, nose tip, left mouth corner, and right mouth corner—and are warped (reshaped) into a canonical pose before embedding. This alignment step reduces intra-class variance, stabilizes similarity scores, and improves recognition performance. In professional face recognition systems, alignment is fundamental rather than optional.

1.5 The Pipelines You Will Build

Throughout this book, you will construct **two complete pipelines** by running and observing individual stages, then combining them into a real-time system.

Pipeline 1: Enrollment Pipeline (Reference Creation/storing)

- Face Detection
- 5-Point Landmark Detection
- Face Alignment (Warping)
- Face Embedding Extraction (ArcFace ONNX)
- Store Embedding in Database

Pipeline 2: Recognition Pipeline (Live Identification/querying)

- Face Detection
- 5-Point Landmark Detection
- Face Alignment (Warping)
- Query Face Embedding Extraction (ArcFace ONNX)
- Compare Query Face Embedding with Stored Reference Embeddings (Threshold Decision)

Note: Validation & Debugging (Not a Pipeline)

Camera testing, face detection checks, 5-point landmark visualization, and embedding inspection are used before and during development for validation and debugging.

These activities do not form a pipeline and do not perform enrollment or recognition.

1.6 Project Structure and Design Philosophy

The project is organized so that each stage can be validated independently. Failures remain localized, and debugging is straightforward.

Top-level structure:

```
face-recognition-5pt/
├── data/
├── models/
├── src/
├── init_project.py
├── README.md
└── book/
```

Design guarantees:

- models can be replaced without code changes
- data, models, and logic are cleanly separated
- results are reproducible across machines

1.7 The data/ Directory

The *data/* directory stores persistent artifacts generated by the system. Nothing here is written by hand.

```
data/
├── db/
└── enroll/
```

Enrollment Images

```
data/enroll

├── Fani

|    ├── 1767874944225.jpg

|    ├── 1767874946090.jpg

|    ├── ...

└── Gabriel

     ├── 1767874858183.jpg

     ├── 1767874859974.jpg

     ├── ...
```

- images are already aligned to *112×112*
- stored for inspection, evaluation, and debugging
- not required at runtime

Recognition Database

```
data/db/
├── face_db.json
└── face_db.npz
```

- **face_db.json:** metadata (names, dimensions, timestamps)
- **face_db.npz:** one L2-normalized embedding per identity

Numerical data and metadata are separated to keep formats inspectable.

1.8 The models/ Directory

```
models
└── embedder_arcface.onnx
```

- ONNX keeps models framework-independent
- models are treated as plug-ins

1.9 The src/ Directory

Each file has exactly one responsibility:

```
src/
├── align.py
├── camera.py
├── detect.py
├── embed.py
├── enroll.py
├── evaluate.py
├── haar_5pt.py
├── landmarks.py
└── recognize.py
```

Table 1: Involved files and their responsibilities.

File	Responsibility
camera.py	Testing the video capture + FPS
detect.py	Testing face bounding boxes
landmarks.py	Testing 5-point landmarks
align.py	Testing geometric alignment
embed.py	Testing embedding extraction
enroll.py	database construction
evaluate.py	threshold tuning
haar_5pt.py	Haar serves for face detection together while MediaPipe FaceMesh extract 5 keypoints used for alignment
recognize.py	real-time integration

1.10 Automated Project Creation

The entire project file system is created programmatically using *init_project.py*. This approach eliminates manual setup errors, ensures the documentation and code never diverge, and guarantees that every reader starts from the exact same baseline. The script can be run from an empty project directory using ***`python init_project.py`***, and it is safe to re-run at any time because existing files are never overwritten. The full source code for this script is provided in Appendix A1.

1.11 Environment Setup

Requirements

- Python 3.9+
- webcam
- macOS, Linux, or Windows

Check: ***`python --version`***

Virtual Environment

Step 1: Create a new isolated Python virtual environment named *.venv* for the project, so dependencies can be installed without affecting the default Python used by the OS and shared by all projects.

```
python -m venv .venv
```

Step 2: Activate:

macOS / Linux: `source .venv/bin/activate`
Windows (PowerShell): `.venv\Scripts\Activate.ps1`

1.12 Dependencies

```
python -m pip install --upgrade pip
```

```
pip install opencv-python numpy onnxruntime scipy tqdm mediapipe
```

Table 2: Packages and their purposes

Package	Purpose
OpenCV (opencv-python)	Camera access, Haar face detection, image processing
NumPy	Numerical and vector operations
ONNX Runtime	CPU inference for ArcFace embedding model
SciPy	Distance computation and evaluation utilities
MediaPipe	Facial landmark detection (used to extract 5-point landmarks for alignment)
tqdm	Progress bars for enrollment and evaluation

1.13 Camera Permissions and First Validation

Before proceeding, verify that the system can reliably access a webcam.

If this step fails, **do not continue**.

Camera Permissions
macOS: System Settings → Privacy & Security → Camera; Allow access for Terminal or VS Code, then restart the terminal.
Windows / Linux: Ensure no other application is currently using the camera.

Mandatory Camera Validation
The camera module is intentionally simple and serves as the first sanity check for the entire project. It must:

- open the webcam
- read frames continuously
- display live video
- report an approximate FPS

File: scr/camera.py

Code: Refer to Appendix A2

How to run: From an empty project directory type **`python -m src.camera`** then hit ENTER.

Expected result
- a live camera window opens
- motion appears smooth
- FPS is displayed
- pressing q exits cleanly

If this does not work
- recheck camera permissions
- try another camera index
- reduce resolution
- resolve this issue before moving on

Once camera access is validated, the face recognition pipeline can safely proceed.

1.14 Next Steps: From Setup to Face Detection
Once this section is complete, you can:
- design a face recognition pipeline from scratch
- explain why alignment improves accuracy
- debug camera and environment issues
- run CPU-only real-time vision reliably

With architecture, structure, and environment locked, the system is ready to move from input validation to face detection.

1.15 Face Detection Testing
This section performs a minimal validation of the face detection stage to confirm that a face can be detected and localized with a bounding box using CPU-only execution. The goal is not accuracy or robustness, but to ensure that the vision pipeline is operational before progressing further. The implementation is provided in src/detect.py (see Appendix A3). When executed, the script should open a webcam window and draw a bounding box around the detected face; failure at this stage indicates unresolved camera or environment issues that must be fixed before continuing.

1.16 5-Point Landmarks Testing
This section validates the generation of five facial landmarks from a detected face to ensure that the system consistently produces points in a fixed order: left eye, right eye, nose tip, left mouth corner, and right mouth corner. These landmarks are required for all downstream stages, particularly face alignment, and their ordering is treated as a strict contract throughout the project. The implementation is provided in src/landmarks.py (see Appendix A4). When executed, the script should display a face bounding box with five labeled points; inaccuracies in point placement are expected and acceptable at this stage, as the purpose is to verify consistent landmark format rather than precision.

1.17 Face Alignment Testing
This section validates face alignment using the five landmarks produced in Section 1.16, confirming that detected faces can be rotated, scaled, and repositioned into a fixed, canonical 112×112 image. Alignment ensures that all faces entering later stages share the same pose and scale, regardless of camera angle or distance. The process uses a similarity transform (rotation, scale, and translation only) and does not model perspective distortion. The implementation is provided in src/align.py (see Appendix A5). When executed, the script should display both the original face with landmarks and a second window showing an upright, centered aligned face; failure at this stage typically indicates incorrect landmarks or inadequate face bounding boxes and must be resolved before proceeding.

1.18 Face Embeddings Testing
This section validates the transition from images to numerical representations by confirming that an aligned face can be converted into a fixed-length, L2-normalized embedding vector suitable for similarity comparison. After detection, landmark estimation, and alignment, the aligned face crop is processed to produce a compact embedding whose norm should be approximately 1.0, indicating correct normalization. The implementation is provided in src/embed.py (see Appendix A6). When executed, the script should display the aligned face and print the embedding shape and norm; failures at this stage typically indicate upstream alignment errors or incorrect normalization and must be fixed before proceeding.

Chapter 2 — Face Enrollment

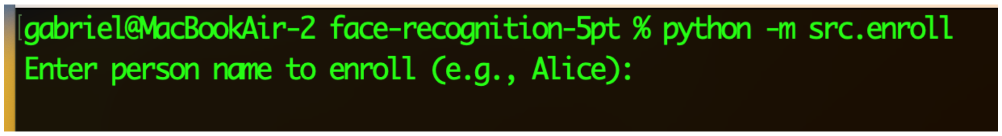
Enrollment is the phase where the system creates a *durable identity record* for a person. Instead of saving raw photos as "the identity," the system captures multiple face samples, converts each sample into an *embedding vector*, and then compresses those samples (*converted into embeddings*) into a single template embedding (*the mean embedding*, L2-normalized). After enrollment, recognition becomes possible because the system now has known identities stored in a persistent database it can compare against.

The enrollment pipeline is implemented in src/enroll.py (see Appendix A7) and reuses the same core stages you already validated in Chapter 1:

camera capture ⟶ face detection → 5-point landmark localization ⟶ face alignment → embedding extraction.

When you run ***`python -m src.enroll`***,

1) the terminal prompts for a person's name,



```
gabriel@MacBookAir-2 face-recognition-5pt % python -m src.enroll
Enter person name to enroll (e.g., Alice):
```

Figure 1: Screenshot of the terminal after initiating the enroll process

2) then opens a live camera window showing the face box (*face detection*), landmarks (*5pt-face landmarks*), and a running sample counter; moreover, a second window displays the aligned 112×112 crop (*face alignment*) that is used for embedding.

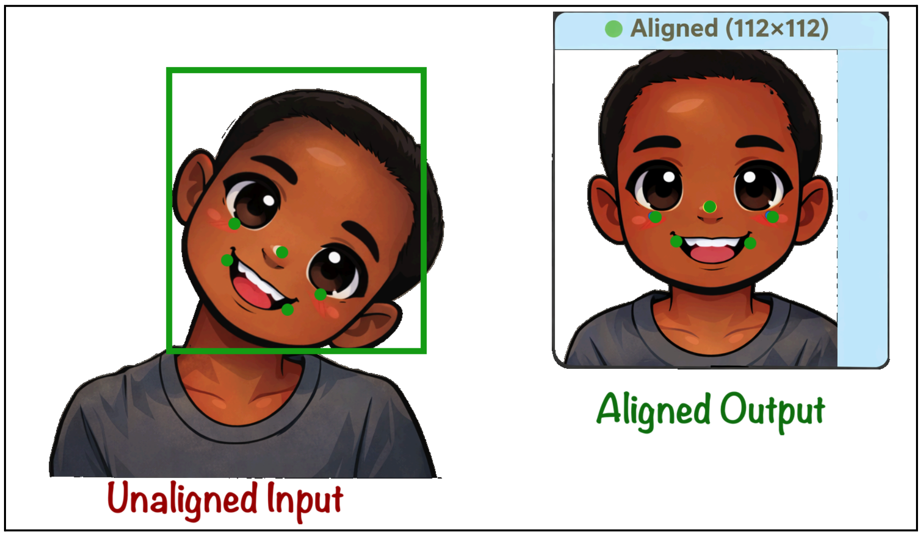


Figure 2: Despite head tilt in the input image (left), 5-point landmark alignment produces a normalized, upright face (right) suitable for embedding extraction.

3) Each time you capture a sample (manually with SPACE or automatically in auto mode by pressing "A" key), the script takes the current aligned face crop and immediately computes an embedding vector from it; that embedding is appended to the current sample set in memory.

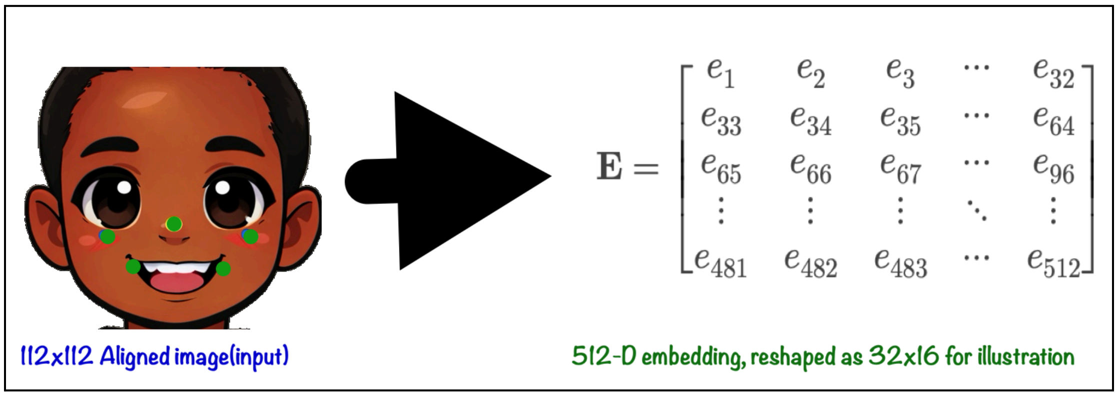

$$\mathbf{E} = \begin{bmatrix} e_1 & e_2 & e_3 & \cdots & e_{32} \\ e_{33} & e_{34} & e_{35} & \cdots & e_{64} \\ e_{65} & e_{66} & e_{67} & \cdots & e_{96} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ e_{481} & e_{482} & e_{483} & \cdots & e_{512} \end{bmatrix}$$
112x112 Aligned image(input)
512-D embedding, reshaped as 32x16 for illustration

Figure 3: An aligned face image is converted by ArcFace into a 512-dimensional embedding. For visualization only, the embedding can be reshaped into a 16×32 matrix with no spatial meaning.

4) When you press "s" to save the crop, the aligned face image is written to *data/enroll/<name>/* as a timestamped file for inspection and evaluation. These images are not the recognition database; the actual database consists only of the stored face embeddings. The saved images are debugging artifacts and can be deleted at any time without affecting enrollment or recognition results. After a successful save, the person is now officially enrolled: subsequent recognition runs can embed a live face and *compare* it to this stored template to decide whether the person matches or should be rejected as unknown.

Chapter 3 — Threshold Evaluation

This chapter evaluates how well the enrolled identities are separated and helps determine an appropriate recognition threshold. It does this by computing cosine distances for two types of comparisons: *genuine pairs*, where both samples belong to the same person, and *impostor pairs*, where the samples come from different people. A good threshold is one that keeps *false accepts (FAR)* low while avoiding excessive *false rejects (FRR)*. The evaluation uses the aligned enrollment images saved earlier in data/enroll/<name>/*.jpg, prints statistics comparing genuine and impostor distance distributions, and suggests a threshold for a target false-accept rate (default 1%).

File: src/evaluate.py

Code: Refer to Appendix A8

How to run: From an empty project directory type `python -m src.evaluate` then hit ENTER.

Note: Make sure you saved aligned crops during enrollment (Refer to Chapter 2)

Expected output

- You should see two distribution summaries:
    - Genuine distances are usually smaller
    - Impostor distances are usually larger
- Then a sweep table (10 rows sampled)
- Then one suggested threshold

Notes

- If you only enrolled one person, impostor distances will be empty. Enroll at least two people for a meaningful FAR.

- If you get meaningless results, please check that a real ArcFace model is plugged in first. If not, plug it in and then rerun the evaluation.

Chapter 4 — Live Recognition

This chapter brings all previous components together into a real-time face recognition system. In a continuous loop, the system detects a face, extracts 5-point landmarks, aligns the face to a 112×112 (HxW in pixels) canonical crop, computes an embedding, matches it against the enrolled database, and displays the predicted identity or Unknown. The chapter also introduces practical CPU-focused optimizations, including processing every N frames, ROI-based detection to reduce computation, and temporal smoothing to stabilize predictions over time. The recognition process uses the enrolled database stored in data/db/face_db.npz and is implemented in src/recognize.py, which orchestrates detection, alignment, embedding, matching, and camera input using the supporting modules in the project.

File: src/recognize.py

Code: Refer to Appendix A9

How to run: From an empty project directory type `python -m src.recognize` then hit ENTER.

Note: Ensure you enrolled at least one person (Refer to Chapter 2)

Expected output

- A webcam window opens.
- A face box appears.
- A label appears above the box:
    - your enrolled name, or
    - Unknown
- A confidence bar appears near the top of the box.
- Press q to quit.
- Press + to loosen threshold (more accepts), - to tighten threshold (fewer accepts).

Notes

This implementation uses OpenCV Haar cascades for face detection and MediaPipe FaceMesh for 5-point landmarks; only the ArcFace embedder requires an ONNX model file.

Common issues

Common issues include a missing database file (which means enrollment has not been run yet), failure to detect faces due to an invalid or empty detector model, and unstable or flickering identity labels, which can usually be improved by increasing the smoothing_window or accept_hold_frames parameters.

Chapter 5 — Plugging in ONNX Model

In this chapter, you replace the placeholder embedding model with a real ArcFace ONNX model. After this step, embeddings and thresholds become meaningful.

Your models/ directory must end up exactly like this:

```
models/
└── embedder_arcface.onnx
```

Follow the steps below in order.

Step 1 — Install the ArcFace embedding model (CPU)

a) From the project root download the files:

```
curl -L -o buffalo_l.zip \

"https://sourceforge.net/projects/insightface.mirror/files/v0.7/buffalo_l.zip/download"
```

b) Unzip the archive:

```
unzip -o buffalo_l.zip
```

c) Copy the ArcFace ONNX model into the project:

```
cp w600k_r50.onnx models/embedder_arcface.onnx
```

d) Clean up unused files (optional but recommended)

```
rm -f buffalo_l.zip w600k_r50.onnx 1k3d68.onnx 2d106det.onnx det_10g.onnx genderage.onnx
```

e) Quick validation

```
python -m src.embed
```

f) Expected output on the Window

```
embedding dim: 512
norm(before L2): 21.85
cos(prev,this): 0.988
```

What each value means

1) embedding dim: 512

ArcFace (ResNet-50, w600k) outputs 512-dimensional embeddings.
Think of the embedding as a 512-character password that uniquely describes a face — not readable by humans, but extremely effective for comparison. While 112 × 112 is the standardized aligned photo size in pixels; 512 is the digital fingerprint extracted from that photo.

This confirms real ArcFace model is loaded

2) norm(before L2): 21.85
ArcFace outputs a raw feature vector:
- magnitude depends on network activations
- usually between 15 and 30
- not yet normalized

To normalize it, you do
`emb_normed = emb / ||emb||`
which resulted into
`||emb_normed|| = 1.000`

So the pipeline is doing the correct mathematical sequence:
raw embedding → L2 normalization → unit vector

3) cos(prev,this): 0.988
This means:
- same face
- consecutive frames
- extremely stable embedding

For reference:
- same person: 0.6–0.99
- different people: < 0.4 (usually far lower)

Chapter 6 — Final Takeaways

This chapter highlights the key lessons from the entire project. It does not introduce new components or techniques; instead, it summarizes the design decisions, performance principles, and practical insights that shaped the system. These takeaways explain *why* the pipeline works, *what* truly matters in practice, and *how* to reason about real-world face recognition systems built for clarity, stability, and CPU-first execution.

10.1 Design Philosophy

- Built CPU-first to enforce clarity, discipline, and reproducibility
- Inefficiencies are visible; unstable logic cannot hide
- A system that runs well on CPU scales naturally elsewhere

10.2 Real-Time Performance

- Stability matters more than maximum FPS
- Identity does not change frame-to-frame
- Heavy stages (detection, alignment, embedding, matching) do not run every frame
- ROI-based detection reduces computation without sacrificing robustness
- Fixed input sizes keep behavior predictable

10.3 Decision Stability

- Single-frame decisions are noisy
- Temporal smoothing stabilizes recognition results
- Acceptance holds reduce flicker between names and “Unknown”

10.4 What Matters Most

- Face alignment quality
- Embedding model choice
- Enrollment data quality

10.5 What Matters Least

- Micro-optimizations
- Drawing speed
- Clever but opaque tricks

10.6 Common Failure Sources

- Camera access and permissions
- Poor detection or landmark quality
- Databases built with the wrong embedder
- Untuned or guessed thresholds

10.7 Debugging Principles

- Test each stage independently
- Re-enroll after changing models
- Tune thresholds using real data

10.8 Final Outcome

- CPU-only
- Real-time
- Explainable
- Debuggable
- Practical

You now have a research-grade face recognition pipeline you can understand, fix, extend, and deploy.

End of book.

References

1. Deng, J., Guo, J., Xue, N., & Zafeiriou, S. (2019). ArcFace: Additive Angular Margin Loss for Deep Face Recognition. *CVPR 2019*.
2. InsightFace Project. InsightFace: 2D & 3D Face Analysis Project. (Project documentation and models such as ArcFace / ResNet).
3. ONNX. Open Neural Network Exchange (ONNX): Specification and ecosystem.
4. ONNX Runtime. ONNX Runtime: Cross-platform inference engine for ONNX models.
5. Viola, P., & Jones, M. (2001). Rapid Object Detection using a Boosted Cascade of Simple Features. *CVPR 2001*. (Foundation of Haar cascade face detection.)
6. OpenCV Documentation. OpenCV: Computer Vision Library (including Haar cascades, image processing, and camera I/O).
7. Lugaresi, C., Tang, J., Nash, H., et al. (2019). MediaPipe: A Framework for Building Perception Pipelines.(MediaPipe framework overview.)
8. MediaPipe Documentation. MediaPipe Face Mesh (facial landmark detection and landmark indexing).
9. NumPy Developers. NumPy: Fundamental package for numerical computing in Python.
10. Virtanen, P., Gommers, R., Oliphant, T. E., et al. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17, 261–272.
11. Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools.*
12. Hartley, R., & Zisserman, A. (2003). Multiple View Geometry in Computer Vision (2nd ed.). Cambridge University Press. (Geometric transforms; alignment concepts.)
13. Jain, A. K., Ross, A., & Prabhakar, S. (2004). An Introduction to Biometric Recognition. *IEEE Transactions on Circuits and Systems for Video Technology*, 14(1), 4–20.
14. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. *CVPR 2015*. (Embedding-space concept; comparison baseline.)
15. CV2 / OpenCV Function Reference. cv2.estimateAffinePartial2D and cv2.warpAffine documentation(similarity transform and warping used for alignment).

Appendices

Appendix A: Code

A1: init_projects.py

A2: camera.py

A3: detect.py

A4: landmarks.py

A5: align.py

A6: embed.py

A7: enroll.py

A8: evaluate.py

A9: recognize.py

A9: haar_5pt.py

Appendix A: Code

A1: src/init_projects.py

```
from pathlib import Path

# Canonical project structure
structure = {
   "data/enroll": [],
   "data/db": [],
   "models": [
       "embedder_arcface.onnx",
   ],
   "src": [
       "camera.py",
       "detect.py",
       "landmarks.py",
       "align.py",
       "embed.py",
       "enroll.py",
       "recognize.py",
       "evaluate.py",
       "haar_5pt.py",
   ],
   "book": [],
}
for folder, files in structure.items():
   folder_path = Path(folder)
   folder_path.mkdir(parents=True, exist_ok=True)

   for file in files:
       file_path = folder_path / file
       if not file_path.exists():
           file_path.touch()
print("face-recognition-5pt project structure created successfully.")
```

A2: src/camera.py

```python
# src/camera.py
import cv2

def main():
   cap = cv2.VideoCapture(0)
   if not cap.isOpened():
       raise RuntimeError("Camera not opened. Try changing index (0/1/2).")

   print("Camera test. Press 'q' to quit.")
   while True:
       ok, frame = cap.read()
       if not ok:
           print("Failed to read frame.")
           break

       cv2.imshow("Camera Test", frame)
       if (cv2.waitKey(1) & 0xFF) == ord("q"):
           break

   cap.release()
   cv2.destroyAllWindows()

if __name__ == "__main__":
   main()
```

A3: src/detect.py

```python
# src/detect.py
import cv2

def main():
   cascade_path = cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
   face = cv2.CascadeClassifier(cascade_path)
   if face.empty():
       raise RuntimeError(f"Failed to load cascade: {cascade_path}")

   cap = cv2.VideoCapture(0)
   if not cap.isOpened():
       raise RuntimeError("Camera not opened. Try camera index 0/1/2.")

   print("Haar face detect (minimal). Press 'q' to quit.")
   while True:
       ok, frame = cap.read()
       if not ok:
           break

       gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

       # minimal but practical defaults
       faces = face.detectMultiScale(
           gray,
           scaleFactor=1.1,
           minNeighbors=5,
           minSize=(60, 60),
       )

       for (x, y, w, h) in faces:
           cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

       cv2.imshow("Face Detection", frame)
       if (cv2.waitKey(1) & 0xFF) == ord("q"):
           break

   cap.release()
   cv2.destroyAllWindows()

if __name__ == "__main__":
   main()
```

A4: src/landmarks.py

```python
# src/landmarks.py
"""
Minimal pipeline:
camera -> Haar face box -> MediaPipe FaceMesh (full-frame) -> extract 5 keypoints -> draw

Run:
 python -m src.landmarks

Keys:
 q : quit
"""

import cv2
import numpy as np
import mediapipe as mp


# 5-point indices (FaceMesh)
IDX_LEFT_EYE = 33
IDX_RIGHT_EYE = 263
IDX_NOSE_TIP = 1
IDX_MOUTH_LEFT = 61
IDX_MOUTH_RIGHT = 291


def main():
   # Haar
   cascade_path = cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
   face = cv2.CascadeClassifier(cascade_path)
   if face.empty():
       raise RuntimeError(f"Failed to load cascade: {cascade_path}")

   # FaceMesh
   fm = mp.solutions.face_mesh.FaceMesh(
       static_image_mode=False,
       max_num_faces=1,
       refine_landmarks=True,
       min_detection_confidence=0.5,
       min_tracking_confidence=0.5,
   )

   cap = cv2.VideoCapture(0)
   if not cap.isOpened():
       raise RuntimeError("Camera not opened. Try camera index 0/1/2.")

   print("Haar + FaceMesh 5pt (minimal). Press 'q' to quit.")
   while True:
       ok, frame = cap.read()
       if not ok:
           break

       H, W = frame.shape[:2]
       gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

       faces = face.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(60, 60))

       # draw ALL haar faces (no ranking)
       for (x, y, w, h) in faces:
           cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

       # FaceMesh on full frame (simple)
       rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
       res = fm.process(rgb)

       if res.multi_face_landmarks:
           lm = res.multi_face_landmarks[0].landmark
           idxs = [IDX_LEFT_EYE, IDX_RIGHT_EYE, IDX_NOSE_TIP, IDX_MOUTH_LEFT, IDX_MOUTH_RIGHT]

           pts = []
           for i in idxs:
               p = lm[i]
               pts.append([p.x * W, p.y * H])
           kps = np.array(pts, dtype=np.float32)   # (5,2)

           # enforce left/right ordering
           if kps[0, 0] > kps[1, 0]:
               kps[[0, 1]] = kps[[1, 0]]
           if kps[3, 0] > kps[4, 0]:
               kps[[3, 4]] = kps[[4, 3]]

           # draw 5 points
           for (px, py) in kps.astype(int):
               cv2.circle(frame, (int(px), int(py)), 4, (0, 255, 0), -1)

           cv2.putText(frame, "5pt", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

       cv2.imshow("5pt Landmarks", frame)
       if (cv2.waitKey(1) & 0xFF) == ord("q"):
           break

   cap.release()
   cv2.destroyAllWindows()


if __name__ == "__main__":
   main()
```

A5: src/align.py

```python
# src/align.py
"""
Alignment demo using your WORKING pipeline:
- Haar face detection (fast)
- MediaPipe FaceMesh -> 5 keypoints (stable)
- ArcFace-style 5pt alignment -> 112x112 (or any size you set)

This avoids the bug in haar_5pt.py where the aligned window was shown
only after the loop and using stale variables.

Run:
 python -m src.align

Keys:
 q  quit
 s  save current aligned face to data/debug_aligned/<timestamp>.jpg
"""

from __future__ import annotations

import os
import time
from pathlib import Path
from typing import Tuple

import cv2
import numpy as np

# Import from your existing script
from .haar_5pt import Haar5ptDetector, align_face_5pt


def _put_text(img, text: str, xy=(10, 30), scale=0.8, thickness=2):
   cv2.putText(img, text, xy, cv2.FONT_HERSHEY_SIMPLEX, scale, (255, 255, 255), thickness, cv2.LINE_AA)


def _safe_imshow(win: str, img: np.ndarray):
   if img is None:
       return
   cv2.imshow(win, img)


def main(
   cam_index: int = 0,
   out_size: Tuple[int, int] = (112, 112),
   mirror: bool = True,
):
   cap = cv2.VideoCapture(cam_index)

   det = Haar5ptDetector(
       min_size=(70, 70),
       smooth_alpha=0.80,
       debug=True,
   )

   out_w, out_h = int(out_size[0]), int(out_size[1])
   blank = np.zeros((out_h, out_w, 3), dtype=np.uint8)

   # Where to save aligned snapshots
   save_dir = Path("data/debug_aligned")
   save_dir.mkdir(parents=True, exist_ok=True)

   last_aligned = blank.copy()
   fps_t0 = time.time()
   fps_n = 0
   fps = 0.0

   print("align running. Press 'q' to quit, 's' to save aligned face.")

   while True:
       ok, frame = cap.read()
       if not ok:
           break

       if mirror:
           frame = cv2.flip(frame, 1)

       faces = det.detect(frame, max_faces=1)

       vis = frame.copy()
       aligned = None

       if faces:
           f = faces[0]

           # Draw box + 5 pts
           cv2.rectangle(vis, (f.x1, f.y1), (f.x2, f.y2), (0, 255, 0), 2)
           for (x, y) in f.kps.astype(int):
               cv2.circle(vis, (int(x), int(y)), 3, (0, 255, 0), -1)

           # Align (this is the whole point)
           aligned, _M = align_face_5pt(frame, f.kps, out_size=out_size)

           # Keep last good aligned (so window doesn't go black on brief misses)
           if aligned is not None and aligned.size:
               last_aligned = aligned

           _put_text(vis, "OK  (Haar + FaceMesh 5pt)", (10, 30), 0.75, 2)
       else:
           _put_text(vis, "no face", (10, 30), 0.9, 2)

       # FPS
       fps_n += 1
       dt = time.time() - fps_t0
       if dt >= 1.0:
           fps = fps_n / dt
           fps_n = 0
           fps_t0 = time.time()
       _put_text(vis, f"FPS: {fps:.1f}", (10, 60), 0.75, 2)
       _put_text(vis, f"warp: 5pt -> {out_w}x{out_h}", (10, 90), 0.75, 2)

       _safe_imshow("align - camera", vis)
       _safe_imshow("align - aligned", last_aligned)

       key = cv2.waitKey(1) & 0xFF
       if key == ord("q"):
           break
       if key == ord("s"):
           ts = int(time.time() * 1000)
           out_path = save_dir / f"{ts}.jpg"
           cv2.imwrite(str(out_path), last_aligned)
           print(f"[align] saved: {out_path}")

   cap.release()
   cv2.destroyAllWindows()


if __name__ == "__main__":
   main()
```

A6: src/embed.py

```python
# src/embed.py
"""
Embedding stage (ArcFace ONNX) using your working pipeline:

camera
-> Haar detection
-> FaceMesh 5pt
-> align_face_5pt (112x112)
-> ArcFace embedding
-> vector visualization (education)

Run:
 python -m src.embed

Keys:
 q : quit
 p : print embedding stats to terminal
"""

from __future__ import annotations

from dataclasses import dataclass
from typing import Tuple, Optional

import time
import cv2
import numpy as np
import onnxruntime as ort

from .haar_5pt import Haar5ptDetector, align_face_5pt


# -------------------------
# Data
# -------------------------

@dataclass
class EmbeddingResult:
   embedding: np.ndarray   # (D,) float32, L2-normalized
   norm_before: float
   dim: int


# -------------------------
# Embedder
# -------------------------

class ArcFaceEmbedderONNX:
   """
   ArcFace / InsightFace-style ONNX embedder.
   Input: aligned 112x112 BGR image.
   Output: L2-normalized embedding vector.
   """

   def __init__(
       self,
       model_path: str = "models/embedder_arcface.onnx",
       input_size: Tuple[int, int] = (112, 112),
       debug: bool = False,
   ):
       self.in_w, self.in_h = input_size
       self.debug = debug

       self.sess = ort.InferenceSession(model_path, providers=["CPUExecutionProvider"])
       self.in_name = self.sess.get_inputs()[0].name
       self.out_name = self.sess.get_outputs()[0].name

       if debug:
           print("[embed] model loaded")
           print("[embed] input:", self.sess.get_inputs()[0].shape)
```

```python
            print("[embed] output:", self.sess.get_outputs()[0].shape)

    def _preprocess(self, aligned_bgr: np.ndarray) -> np.ndarray:
        if aligned_bgr.shape[:2] != (self.in_h, self.in_w):
            aligned_bgr = cv2.resize(aligned_bgr, (self.in_w, self.in_h))

        rgb = cv2.cvtColor(aligned_bgr, cv2.COLOR_BGR2RGB).astype(np.float32)
        rgb = (rgb - 127.5) / 128.0
        x = np.transpose(rgb, (2, 0, 1))[None, ...]
        return x.astype(np.float32)

    @staticmethod
    def _l2_normalize(v: np.ndarray, eps: float = 1e-12):
        n = float(np.linalg.norm(v) + eps)
        return (v / n).astype(np.float32), n

    def embed(self, aligned_bgr: np.ndarray) -> EmbeddingResult:
        x = self._preprocess(aligned_bgr)
        y = self.sess.run([self.out_name], {self.in_name: x})[0]
        v = y.reshape(-1).astype(np.float32)
        v_norm, n0 = self._l2_normalize(v)
        return EmbeddingResult(v_norm, n0, v_norm.size)


# -------------------------
# Visualization helpers
# -------------------------

def draw_text_block(img, lines, origin=(10, 30), scale=0.7, color=(0, 255, 0)):
    x, y = origin
    for line in lines:
        cv2.putText(img, line, (x, y), cv2.FONT_HERSHEY_SIMPLEX, scale, color, 2)
        y += int(28 * scale)


def draw_embedding_matrix(
    img: np.ndarray,
    emb: np.ndarray,
    top_left=(10, 220),
    cell_scale: int = 6,
    title: str = "embedding"
):
    """
    Visualize embedding vector as a heatmap matrix.
    """
    D = emb.size
    cols = int(np.ceil(np.sqrt(D)))
    rows = int(np.ceil(D / cols))

    mat = np.zeros((rows, cols), dtype=np.float32)
    mat.flat[:D] = emb

    norm = (mat - mat.min()) / (mat.max() - mat.min() + 1e-6)
    gray = (norm * 255).astype(np.uint8)
    heat = cv2.applyColorMap(gray, cv2.COLORMAP_JET)

    heat = cv2.resize(
        heat,
        (cols * cell_scale, rows * cell_scale),
        interpolation=cv2.INTER_NEAREST,
    )

    x, y = top_left
    h, w = heat.shape[:2]
    ih, iw = img.shape[:2]

    if x + w > iw or y + h > ih:
        return 0, 0

    img[y:y+h, x:x+w] = heat
    cv2.putText(
        img,
        title,
        (x, y - 8),
        cv2.FONT_HERSHEY_SIMPLEX,
        0.6,
        (200, 200, 200),
        2,
    )
    return w, h


def emb_preview_str(emb: np.ndarray, n: int = 8) -> str:
    vals = " ".join(f"{v:+.3f}" for v in emb[:n])
    return f"vec[0:{n}]: {vals} ..."


def cosine_similarity(a: np.ndarray, b: np.ndarray) -> float:
    return float(np.dot(a, b))


# ------------------------
# Demo
# ------------------------

def main():
    cap = cv2.VideoCapture(0)

    det = Haar5ptDetector(
        min_size=(70, 70),
        smooth_alpha=0.80,
        debug=False,
    )

    emb_model = ArcFaceEmbedderONNX(
        model_path="models/embedder_arcface.onnx",
        debug=False,
    )

    prev_emb: Optional[np.ndarray] = None

    print("Embedding Demo running. Press 'q' to quit, 'p' to print embedding.")

    t0 = time.time()
    frames = 0
    fps = 0.0

    while True:
        ok, frame = cap.read()
        if not ok:
            break

        vis = frame.copy()
        faces = det.detect(frame, max_faces=1)
        info = []

        if faces:
            f = faces[0]

            # draw detection
            cv2.rectangle(vis, (f.x1, f.y1), (f.x2, f.y2), (0, 255, 0), 2)
            for (x, y) in f.kps.astype(int):
                cv2.circle(vis, (x, y), 3, (0, 255, 0), -1)

            # align + embed
            aligned, _ = align_face_5pt(frame, f.kps, out_size=(112, 112))
            res = emb_model.embed(aligned)

            info.append(f"embedding dim: {res.dim}")
            info.append(f"norm(before L2): {res.norm_before:.2f}")

            if prev_emb is not None:
                sim = cosine_similarity(prev_emb, res.embedding)
                info.append(f"cos(prev,this): {sim:.3f}")

            prev_emb = res.embedding
            # aligned preview (top-right)
            aligned_small = cv2.resize(aligned, (160, 160))
            h, w = vis.shape[:2]
            vis[10:170, w-170:w-10] = aligned_small

            # --------- VISUALIZATION LAYOUT ---------
            draw_text_block(vis, info, origin=(10, 30))

            HEAT_X, HEAT_Y = 10, 220
            CELL_SCALE = 6

            ww, hh = draw_embedding_matrix(
                vis,
                res.embedding,
                top_left=(HEAT_X, HEAT_Y),
                cell_scale=CELL_SCALE,
                title="embedding heatmap",
            )

            if ww > 0:
                cv2.putText(
                    vis,
                    emb_preview_str(res.embedding),
                    (HEAT_X, HEAT_Y + hh + 28),
                    cv2.FONT_HERSHEY_SIMPLEX,
                    0.55,
                    (200, 200, 200),
                    2,
                )

        else:
            draw_text_block(vis, ["no face"], origin=(10, 30), color=(0, 0, 255))

        # FPS
        frames += 1
        dt = time.time() - t0
        if dt >= 1.0:
            fps = frames / dt
            frames = 0
            t0 = time.time()
        cv2.putText(vis, f"fps: {fps:.1f}", (10, vis.shape[0] - 15),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)

        cv2.imshow("Face Embedding", vis)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break
        elif key == ord("p") and prev_emb is not None:
            print("[embedding]")
            print(" dim:", prev_emb.size)
            print(" min/max:", prev_emb.min(), prev_emb.max())
            print(" first10:", prev_emb[:10])

    cap.release()
    cv2.destroyAllWindows()


if __name__ == "__main__":
    main()
```

A7: src/enroll.py

```python
# src/enroll.py
"""
enroll.py
Enrollment tool using your working pipeline:

camera -> Haar detection -> FaceMesh 5pt -> align_face_5pt (112x112) -> ArcFace embedding
Stores template per identity (mean embedding, L2-normalized).

Re-enroll behavior:
- If data/enroll/<name> already contains aligned crops, those are loaded,
 embedded again, and INCLUDED in the template. New captures are appended.

Outputs:
- data/db/face_db.npz   (name -> embedding vector)
- data/db/face_db.json  (metadata)
Optional:
- data/enroll/<name>/*.jpg aligned face crops

Controls:
- SPACE: capture one sample (if face found)
- a: auto-capture toggle (captures periodically)
- s: save enrollment (after enough total samples)
- r: reset NEW samples (keeps existing crops on disk)
- q: quit
"""

from __future__ import annotations

import json
import time
from dataclasses import dataclass
from pathlib import Path
from typing import Dict, List, Optional, Tuple

import cv2
import numpy as np

from .haar_5pt import Haar5ptDetector, align_face_5pt
from .embed import ArcFaceEmbedderONNX


# -------------------------
# Config
# -------------------------

@dataclass
class EnrollConfig:
   out_db_npz: Path = Path("data/db/face_db.npz")
   out_db_json: Path = Path("data/db/face_db.json")

   save_crops: bool = True
   crops_dir: Path = Path("data/enroll")

   samples_needed: int = 15
   auto_capture_every_s: float = 0.25
   max_existing_crops: int = 300

   # UI
   window_main: str = "enroll"
   window_aligned: str = "aligned_112"


# -------------------------
# DB helpers
# -------------------------

def ensure_dirs(cfg: EnrollConfig) -> None:
   cfg.out_db_npz.parent.mkdir(parents=True, exist_ok=True)
   cfg.out_db_json.parent.mkdir(parents=True, exist_ok=True)
   if cfg.save_crops:
       cfg.crops_dir.mkdir(parents=True, exist_ok=True)


def load_db(cfg: EnrollConfig) -> Dict[str, np.ndarray]:
   if cfg.out_db_npz.exists():
       data = np.load(cfg.out_db_npz, allow_pickle=True)
       return {k: data[k].astype(np.float32) for k in data.files}
   return {}


def save_db(cfg: EnrollConfig, db: Dict[str, np.ndarray], meta: dict) -> None:
   ensure_dirs(cfg)
   np.savez(cfg.out_db_npz, **{k: v.astype(np.float32) for k, v in db.items()})
   cfg.out_db_json.write_text(json.dumps(meta, indent=2), encoding="utf-8")


def mean_embedding(embeddings: List[np.ndarray]) -> np.ndarray:
   """Mean + L2 normalize."""
   E = np.stack([e.reshape(-1) for e in embeddings], axis=0).astype(np.float32)
   m = E.mean(axis=0)
   m = m / (np.linalg.norm(m) + 1e-12)
   return m.astype(np.float32)


# -------------------------
# Crops loader
# -------------------------

def _list_existing_crops(person_dir: Path, max_count: int) -> List[Path]:
   if not person_dir.exists():
       return []
   files = sorted([p for p in person_dir.glob("*.jpg") if p.is_file()])
   if len(files) > max_count:
       files = files[-max_count:]
   return files


def load_existing_samples_from_crops(
   cfg: EnrollConfig,
   emb: ArcFaceEmbedderONNX,
   person_dir: Path,
) -> List[np.ndarray]:
   """
   Reads aligned crops from disk and re-embeds them.
   """
   if not cfg.save_crops:
       return []

   crops = _list_existing_crops(person_dir, cfg.max_existing_crops)
   base: List[np.ndarray] = []

   for p in crops:
       img = cv2.imread(str(p))
       if img is None:
           continue
       try:
           r = emb.embed(img)
           base.append(r.embedding)
       except Exception:
           continue

   return base


# -------------------------
# UI helpers
# -------------------------

def draw_status(
   frame: np.ndarray,
   name: str,
   base_count: int,
   new_count: int,
   needed: int,
   auto: bool,
   msg: str = "",
) -> None:
   total = base_count + new_count
   lines = [
       f"ENROLL: {name}",
       f"Existing: {base_count} | New: {new_count} | Total: {total} / {needed}",
       f"Auto: {'ON' if auto else 'OFF'}  (toggle: a)",
       "SPACE=capture | s=save | r=reset NEW | q=quit",
   ]
   if msg:
       lines.insert(0, msg)

   # draw with black shadow for readability
   y = 30
   for line in lines:
       cv2.putText(frame, line, (10, y), cv2.FONT_HERSHEY_SIMPLEX, 0.62, (0, 0, 0), 4, cv2.LINE_AA)
       cv2.putText(frame, line, (10, y), cv2.FONT_HERSHEY_SIMPLEX, 0.62, (255, 255, 255), 2, cv2.LINE_AA)
       y += 26


# -------------------------
# Main
# -------------------------

def main():
   cfg = EnrollConfig()
   ensure_dirs(cfg)

   name = input("Enter person name to enroll (e.g., Alice): ").strip()
   if not name:
       print("No name provided. Exiting.")
       return

   # Pipeline (your working practical stack)
   det = Haar5ptDetector(min_size=(70, 70), smooth_alpha=0.80, debug=False)
   emb = ArcFaceEmbedderONNX(model_path="models/embedder_arcface.onnx", input_size=(112, 112),
debug=False)

   db = load_db(cfg)

   person_dir = cfg.crops_dir / name
   if cfg.save_crops:
       person_dir.mkdir(parents=True, exist_ok=True)

   base_samples: List[np.ndarray] = load_existing_samples_from_crops(cfg, emb, person_dir)
   new_samples: List[np.ndarray] = []

   status_msg = ""
   if base_samples:
       status_msg = f"Loaded {len(base_samples)} existing samples from disk."

   auto = False
   last_auto = 0.0

   cap = cv2.VideoCapture(0)
   if not cap.isOpened():
       raise RuntimeError("Failed to open camera.")

   cv2.namedWindow(cfg.window_main, cv2.WINDOW_NORMAL)
   cv2.namedWindow(cfg.window_aligned, cv2.WINDOW_NORMAL)
   cv2.resizeWindow(cfg.window_aligned, 240, 240)

   print("\nEnrollment started.")
   if base_samples:
       print(f"Re-enroll mode: found {len(base_samples)} existing samples in {person_dir}/")
   print("Tip: stable lighting, move slightly left/right, different expressions.")
   print("Controls: SPACE=capture, a=auto, s=save, r=reset NEW, q=quit\n")

   t0 = time.time()
   frames = 0
   fps: Optional[float] = None
```

```python
try:
    while True:
        ok, frame = cap.read()
        if not ok:
            break

        vis = frame.copy()
        faces = det.detect(frame, max_faces=1)

        aligned: Optional[np.ndarray] = None

        if faces:
            f = faces[0]

            # draw bbox + kps
            cv2.rectangle(vis, (f.x1, f.y1), (f.x2, f.y2), (0, 255, 0), 2)
            for (x, y) in f.kps.astype(int):
                cv2.circle(vis, (int(x), int(y)), 3, (0, 255, 0), -1)

            aligned, _ = align_face_5pt(frame, f.kps, out_size=(112, 112))
            cv2.imshow(cfg.window_aligned, aligned)
        else:
            cv2.imshow(cfg.window_aligned, np.zeros((112, 112, 3), dtype=np.uint8))

        # auto capture
        now = time.time()
        if auto and aligned is not None and (now - last_auto) >= cfg.auto_capture_every_s:
            r = emb.embed(aligned)
            new_samples.append(r.embedding)
            last_auto = now
            status_msg = f"Auto captured NEW ({len(new_samples)})"

            if cfg.save_crops:
                fn = person_dir / f"{int(now * 1000)}.jpg"
                cv2.imwrite(str(fn), aligned)

        # FPS
        frames += 1
        dt = time.time() - t0
        if dt >= 1.0:
            fps = frames / dt
            frames = 0
            t0 = time.time()

        if fps is not None:
            cv2.putText(vis, f"FPS: {fps:.1f}", (10, vis.shape[0] - 12),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2, cv2.LINE_AA)

        draw_status(
            vis,
            name=name,
            base_count=len(base_samples),
            new_count=len(new_samples),
            needed=cfg.samples_needed,
            auto=auto,
            msg=status_msg,
        )

        cv2.imshow(cfg.window_main, vis)
        key = cv2.waitKey(1) & 0xFF

        if key == ord("q"):
            break

        if key == ord("a"):
            auto = not auto
            status_msg = f"Auto mode {'ON' if auto else 'OFF'}"

        if key == ord("r"):
            new_samples.clear()
            status_msg = "NEW samples reset (existing kept)."
            if key == ord(" "):  # SPACE
                if aligned is None:
                    status_msg = "No face detected. Not captured."
                else:
                    r = emb.embed(aligned)
                    new_samples.append(r.embedding)
                    status_msg = f"Captured NEW ({len(new_samples)})"

                    if cfg.save_crops:
                        fn = person_dir / f"{int(time.time() * 1000)}.jpg"
                        cv2.imwrite(str(fn), aligned)

            if key == ord("s"):
                total = len(base_samples) + len(new_samples)
                if total < max(3, cfg.samples_needed // 2):
                    status_msg = f"Not enough total samples to save (have {total})."
                    continue

                all_samples = base_samples + new_samples
                template = mean_embedding(all_samples)
                db[name] = template

                meta = {
                    "updated_at": time.strftime("%Y-%m-%d %H:%M:%S"),
                    "embedding_dim": int(template.size),
                    "names": sorted(db.keys()),
                    "samples_existing_used": int(len(base_samples)),
                    "samples_new_used": int(len(new_samples)),
                    "samples_total_used": int(len(all_samples)),
                    "note": "Embeddings are L2-normalized vectors. Matching uses cosine similarity.",
                }
                save_db(cfg, db, meta)

                status_msg = f"Saved '{name}' to DB. Total identities: {len(db)}"
                print(status_msg)

                # reload base from disk so UI matches reality
                base_samples = load_existing_samples_from_crops(cfg, emb, person_dir)
                new_samples.clear()

    finally:
        cap.release()
        cv2.destroyAllWindows()


if __name__ == "__main__":
    main()
```

A8: src/evaluate.py

```python
# src/evaluate.py
"""
evaluate.py
Threshold tuning / evaluation using enrollment crops (aligned 112x112).
Assumptions:
- Enrollment crops exist under: data/enroll/<name>/*.jpg
- Crops are aligned (112x112) already (as saved by enroll.py / haar_5pt pipeline)
- Uses ArcFaceEmbedderONNX from embed.py (your working embedder)
Outputs:
- Prints summary stats for genuine/impostor cosine distances
- Suggests a threshold based on a target FAR
Run:
 python -m src.evaluate
"""

from __future__ import annotations

from dataclasses import dataclass
from pathlib import Path
from typing import Dict, List, Tuple

import cv2
import numpy as np

from .embed import ArcFaceEmbedderONNX

# -------------------------
# Config
# -------------------------

@dataclass
class EvalConfig:
   enroll_dir: Path = Path("data/enroll")
   min_imgs_per_person: int = 5
   max_imgs_per_person: int = 80       # cap for speed
   target_far: float = 0.01            # 1% FAR target
   thresholds: Tuple[float, float, float] = (0.10, 1.20, 0.01)  # start, end, step

   # Optional sanity constraints
   require_size: Tuple[int, int] = (112, 112)

# -------------------------
# Math
# -------------------------
def cosine_similarity(a: np.ndarray, b: np.ndarray) -> float:
   a = a.reshape(-1).astype(np.float32)
   b = b.reshape(-1).astype(np.float32)
   # embeddings are already L2-normalized in embed, so dot is cosine
   return float(np.dot(a, b))


def cosine_distance(a: np.ndarray, b: np.ndarray) -> float:
   # distance = 1 - cosine similarity
   return 1.0 - cosine_similarity(a, b)

# -------------------------
# IO
# -------------------------
def list_people(cfg: EvalConfig) -> List[Path]:
   if not cfg.enroll_dir.exists():
       raise FileNotFoundError(f"Enroll dir not found: {cfg.enroll_dir}. Run enroll.py first.")
   return sorted([p for p in cfg.enroll_dir.iterdir() if p.is_dir()])


def _is_aligned_crop(img: np.ndarray, req: Tuple[int, int]) -> bool:
   h, w = img.shape[:2]
   return (w, h) == (int(req[0]), int(req[1]))


def load_embeddings_for_person(
   embedder: ArcFaceEmbedderONNX,
   person_dir: Path,
   cfg: EvalConfig,
) -> List[np.ndarray]:
   imgs = sorted(list(person_dir.glob("*.jpg")))[: cfg.max_imgs_per_person]
   embs: List[np.ndarray] = []

   for img_path in imgs:
       img = cv2.imread(str(img_path))
       if img is None:
           continue

       # If someone accidentally saved non-aligned crops, skip them (keeps eval clean)
       if cfg.require_size is not None and not _is_aligned_crop(img, cfg.require_size):
           continue

       res = embedder.embed(img)
       embs.append(res.embedding)

   return embs

# -------------------------
# Eval
# -------------------------
def pairwise_distances(embs_a: List[np.ndarray], embs_b: List[np.ndarray], same: bool) -> List[float]:
   dists: List[float] = []
   if same:
       for i in range(len(embs_a)):
           for j in range(i + 1, len(embs_a)):
               dists.append(cosine_distance(embs_a[i], embs_a[j]))
   else:
       for ea in embs_a:
           for eb in embs_b:
               dists.append(cosine_distance(ea, eb))
   return dists


def sweep_thresholds(genuine: np.ndarray, impostor: np.ndarray, cfg: EvalConfig):
   t0, t1, step = cfg.thresholds
   thresholds = np.arange(t0, t1 + 1e-9, step, dtype=np.float32)

   # FAR: impostor accepted => dist <= thr |  FRR: genuine rejected  => dist > thr
   results = []
   for thr in thresholds:
       far = float(np.mean(impostor <= thr)) if impostor.size else 0.0
       frr = float(np.mean(genuine > thr)) if genuine.size else 0.0
       results.append((float(thr), far, frr))
   return results


def describe(arr: np.ndarray) -> str:
   if arr.size == 0:
       return "n=0"
   return (
       f"n={arr.size}  mean={arr.mean():.3f}  std={arr.std():.3f}  "
       f"p05={np.percentile(arr, 5):.3f}  p50={np.percentile(arr, 50):.3f}  p95={np.percentile(arr,
95):.3f}"
   )


def main():
   cfg = EvalConfig()

   embedder = ArcFaceEmbedderONNX(
       model_path="models/embedder_arcface.onnx",
       input_size=(112, 112),
       debug=False,
   )

   people_dirs = list_people(cfg)
   if len(people_dirs) < 1:
       print("No enrolled people found.")
       return
   # Load embeddings per person
   per_person: Dict[str, List[np.ndarray]] = {}
   for pdir in people_dirs:
       name = pdir.name
       embs = load_embeddings_for_person(embedder, pdir, cfg)
       if len(embs) >= cfg.min_imgs_per_person:
           per_person[name] = embs
       else:
           print(f"Skipping {name}: only {len(embs)} valid aligned crops (need >=
{cfg.min_imgs_per_person}).")

   names = sorted(per_person.keys())
   if len(names) < 1:
       print("Not enough data to evaluate. Enroll more samples.")
       return

   # Genuine
   genuine_all: List[float] = []
   for name in names:
       genuine_all.extend(pairwise_distances(per_person[name], per_person[name], same=True))

   # Impostor
   impostor_all: List[float] = []
   for i in range(len(names)):
       for j in range(i + 1, len(names)):
           impostor_all.extend(pairwise_distances(per_person[names[i]], per_person[names[j]],
same=False))

   genuine = np.array(genuine_all, dtype=np.float32)
   impostor = np.array(impostor_all, dtype=np.float32)

   print("\n=== Distance Distributions (cosine distance = 1 - cosine similarity) ===")
   print(f"Genuine (same person):   {describe(genuine)}")
   print(f"Impostor (diff persons): {describe(impostor)}")

   results = sweep_thresholds(genuine, impostor, cfg)

   # Choose threshold with FAR <= target_far and minimal FRR
   best = None
   for thr, far, frr in results:
       if far <= cfg.target_far:
           if best is None or frr < best[2]:
               best = (thr, far, frr)

   print("\n=== Threshold Sweep ===")
   stride = max(1, len(results) // 10)
   for thr, far, frr in results[::stride]:
       print(f"thr={thr:.2f}  FAR={far*100:5.2f}%  FRR={frr*100:5.2f}%")

   if best is not None:
       thr, far, frr = best
       print(
           f"\nSuggested threshold (target FAR {cfg.target_far*100:.1f}%): "
           f"thr={thr:.2f}  FAR={far*100:.2f}%  FRR={frr*100:.2f}%"
       )
   else:
       print(
           f"\nNo threshold in range met FAR <= {cfg.target_far*100:.1f}%. "
           "Try widening threshold sweep range or collecting more varied samples."
       )

   # Extra: recommend a similarity-style threshold too
   if best is not None:
       sim_thr = 1.0 - best[0]
       print(f"\n(Equivalent cosine similarity threshold ~ {sim_thr:.3f}, since sim = 1 - dist)")

   print()


if __name__ == "__main__":
   main()
```

A9: src/recognize.py

```
# src/recognize.py
"""
Multi-face recognition (CPU-friendly) using your now-stable pipeline:

Haar (multi-face) -> FaceMesh 5pt (per-face ROI) -> align_face_5pt (112x112)
-> ArcFace ONNX embedding -> cosine distance to DB -> label each face.

Run:
 python -m src.recognize

Keys:
 q : quit
 r : reload DB from disk (data/db/face_db.npz)
 +/- : adjust threshold (distance) live
 d : toggle debug overlay

Notes:
- We run FaceMesh on EACH Haar face ROI (not the full frame). This avoids the
 "FaceMesh points not consistent with Haar box" problem and enables multi-face.
- DB is expected from enroll: data/db/face_db.npz (name -> embedding vector)
- Distance definition: cosine_distance = 1 - cosine_similarity.
 Since embeddings are L2-normalized, cosine_similarity = dot(a,b).
"""

from __future__ import annotations

import time
import json
from dataclasses import dataclass
from pathlib import Path
from typing import Dict, List, Optional, Tuple

import cv2
import numpy as np
import onnxruntime as ort

try:
   import mediapipe as mp
except Exception as e:
   mp = None
   _MP_IMPORT_ERROR = e

# Reuse your known-good alignment method (you said alignment is OK now)
from .haar_5pt import align_face_5pt


# -------------------------
# Data
# -------------------------

@dataclass
class FaceDet:
   x1: int
   y1: int
   x2: int
   y2: int
   score: float
   kps: np.ndarray  # (5,2) float32 in FULL-frame coords


@dataclass
class MatchResult:
   name: Optional[str]
   distance: float
   similarity: float
   accepted: bool


# -------------------------
# Math helpers
```

```python
# ------------------------

def cosine_similarity(a: np.ndarray, b: np.ndarray) -> float:
   a = a.reshape(-1).astype(np.float32)
   b = b.reshape(-1).astype(np.float32)
   return float(np.dot(a, b))


def cosine_distance(a: np.ndarray, b: np.ndarray) -> float:
   return 1.0 - cosine_similarity(a, b)


def _clip_xyxy(x1: float, y1: float, x2: float, y2: float, W: int, H: int) -> Tuple[int, int, int, int]:
   x1 = int(max(0, min(W - 1, round(x1))))
   y1 = int(max(0, min(H - 1, round(y1))))
   x2 = int(max(0, min(W - 1, round(x2))))
   y2 = int(max(0, min(H - 1, round(y2))))
   if x2 < x1:
       x1, x2 = x2, x1
   if y2 < y1:
       y1, y2 = y2, y1
   return x1, y1, x2, y2


def _bbox_from_5pt(
   kps: np.ndarray,
   pad_x: float = 0.55,
   pad_y_top: float = 0.85,
   pad_y_bot: float = 1.15,
) -> np.ndarray:
   """
   Build a nicer face-like bbox from 5 points with asymmetric padding.
   kps: (5,2) in full-frame coords
   """
   k = kps.astype(np.float32)
   x_min = float(np.min(k[:, 0]))
   x_max = float(np.max(k[:, 0]))
   y_min = float(np.min(k[:, 1]))
   y_max = float(np.max(k[:, 1]))

   w = max(1.0, x_max - x_min)
   h = max(1.0, y_max - y_min)

   x1 = x_min - pad_x * w
   x2 = x_max + pad_x * w
   y1 = y_min - pad_y_top * h
   y2 = y_max + pad_y_bot * h
   return np.array([x1, y1, x2, y2], dtype=np.float32)


def _kps_span_ok(kps: np.ndarray, min_eye_dist: float) -> bool:
   """
   Minimal geometry sanity:
   - eyes not collapsed
   - mouth generally below nose
   """
   k = kps.astype(np.float32)
   le, re, no, lm, rm = k
   eye_dist = float(np.linalg.norm(re - le))
   if eye_dist < float(min_eye_dist):
       return False
   if not (lm[1] > no[1] and rm[1] > no[1]):
       return False
   return True

# ------------------------
# DB helpers
# ------------------------

def load_db_npz(db_path: Path) -> Dict[str, np.ndarray]:
   if not db_path.exists():
       return {}
   data = np.load(str(db_path), allow_pickle=True)
```

```
    out: Dict[str, np.ndarray] = {}
    for k in data.files:
        out[k] = np.asarray(data[k], dtype=np.float32).reshape(-1)
    return out

# -------------------------
# Embedder (same as embed_new)
# -------------------------

class ArcFaceEmbedderONNX:
    """
    ArcFace-style ONNX embedder.
    Input: 112x112 BGR -> internally RGB + (x-127.5)/128, NCHW float32.
    Output: (1,D) or (D,)
    """

    def __init__(
        self,
        model_path: str = "models/embedder_arcface.onnx",
        input_size: Tuple[int, int] = (112, 112),
        debug: bool = False,
    ):
        self.model_path = model_path
        self.in_w, self.in_h = int(input_size[0]), int(input_size[1])
        self.debug = bool(debug)

        self.sess = ort.InferenceSession(model_path, providers=["CPUExecutionProvider"])
        self.in_name = self.sess.get_inputs()[0].name
        self.out_name = self.sess.get_outputs()[0].name

        if self.debug:
            print("[embed] model:", model_path)
            print("[embed] input:", self.sess.get_inputs()[0].name, self.sess.get_inputs()[0].shape,
self.sess.get_inputs()[0].type)
            print("[embed] output:", self.sess.get_outputs()[0].name, self.sess.get_outputs()[0].shape,
self.sess.get_outputs()[0].type)

    def _preprocess(self, aligned_bgr_112: np.ndarray) -> np.ndarray:
        img = aligned_bgr_112
        if img.shape[1] != self.in_w or img.shape[0] != self.in_h:
            img = cv2.resize(img, (self.in_w, self.in_h), interpolation=cv2.INTER_LINEAR)

        rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB).astype(np.float32)
        rgb = (rgb - 127.5) / 128.0
        x = np.transpose(rgb, (2, 0, 1))[None, ...]
        return x.astype(np.float32)

    @staticmethod
    def _l2_normalize(v: np.ndarray, eps: float = 1e-12) -> np.ndarray:
        v = v.astype(np.float32).reshape(-1)
        n = float(np.linalg.norm(v) + eps)
        return (v / n).astype(np.float32)

    def embed(self, aligned_bgr_112: np.ndarray) -> np.ndarray:
        x = self._preprocess(aligned_bgr_112)
        y = self.sess.run([self.out_name], {self.in_name: x})[0]
        emb = np.asarray(y, dtype=np.float32).reshape(-1)
        return self._l2_normalize(emb)

# -------------------------
# Multi-face Haar + FaceMesh(ROI) 5pt
# -------------------------

class HaarFaceMesh5pt:
    def __init__(
        self,
        haar_xml: Optional[str] = None,
        min_size: Tuple[int, int] = (70, 70),
        debug: bool = False,
    ):
        self.debug = bool(debug)
        self.min_size = tuple(map(int, min_size))
        if haar_xml is None:
            haar_xml = cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
        self.face_cascade = cv2.CascadeClassifier(haar_xml)
        if self.face_cascade.empty():
            raise RuntimeError(f"Failed to load Haar cascade: {haar_xml}")

        if mp is None:
            raise RuntimeError(
                f"mediapipe import failed: {_MP_IMPORT_ERROR}\n"
                f"Install: pip install mediapipe==0.10.21"
            )

        # IMPORTANT: we run FaceMesh on ROI (one face per ROI), so max_num_faces=1
        self.mesh = mp.solutions.face_mesh.FaceMesh(
            static_image_mode=False,
            max_num_faces=1,
            refine_landmarks=True,
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5,
        )

        # 5pt indices (same as your working file)
        self.IDX_LEFT_EYE = 33
        self.IDX_RIGHT_EYE = 263
        self.IDX_NOSE_TIP = 1
        self.IDX_MOUTH_LEFT = 61
        self.IDX_MOUTH_RIGHT = 291

    def _haar_faces(self, gray: np.ndarray) -> np.ndarray:
        faces = self.face_cascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            flags=cv2.CASCADE_SCALE_IMAGE,
            minSize=self.min_size,
        )
        if faces is None or len(faces) == 0:
            return np.zeros((0, 4), dtype=np.int32)
        return faces.astype(np.int32)  # (x,y,w,h)

    def _roi_facemesh_5pt(self, roi_bgr: np.ndarray) -> Optional[np.ndarray]:
        H, W = roi_bgr.shape[:2]
        if H < 20 or W < 20:
            return None
        rgb = cv2.cvtColor(roi_bgr, cv2.COLOR_BGR2RGB)
        res = self.mesh.process(rgb)
        if not res.multi_face_landmarks:
            return None

        lm = res.multi_face_landmarks[0].landmark
        idxs = [self.IDX_LEFT_EYE, self.IDX_RIGHT_EYE, self.IDX_NOSE_TIP, self.IDX_MOUTH_LEFT,
self.IDX_MOUTH_RIGHT]

        pts = []
        for i in idxs:
            p = lm[i]
            pts.append([p.x * W, p.y * H])

        kps = np.array(pts, dtype=np.float32)

        # enforce left/right ordering
        if kps[0, 0] > kps[1, 0]:
            kps[[0, 1]] = kps[[1, 0]]
        if kps[3, 0] > kps[4, 0]:
            kps[[3, 4]] = kps[[4, 3]]

        return kps

    def detect(self, frame_bgr: np.ndarray, max_faces: int = 5) -> List[FaceDet]:
        H, W = frame_bgr.shape[:2]
        gray = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2GRAY)

        faces = self._haar_faces(gray)
        if faces.shape[0] == 0:
            return []

        # sort by area desc, keep top max_faces
        areas = faces[:, 2] * faces[:, 3]
        order = np.argsort(areas)[::-1]
        faces = faces[order][:max_faces]

        out: List[FaceDet] = []

        for (x, y, w, h) in faces:
            # expand ROI a bit for FaceMesh stability
            mx, my = 0.25 * w, 0.35 * h
            rx1, ry1, rx2, ry2 = _clip_xyxy(x - mx, y - my, x + w + mx, y + h + my, W, H)
            roi = frame_bgr[ry1:ry2, rx1:rx2]

            kps_roi = self._roi_facemesh_5pt(roi)
            if kps_roi is None:
                if self.debug:
                    print("[recognize] FaceMesh none for ROI -> skip")
                continue

            # map ROI kps back to full-frame coords
            kps = kps_roi.copy()
            kps[:, 0] += float(rx1)
            kps[:, 1] += float(ry1)

            # sanity: eye distance relative to Haar width
            if not _kps_span_ok(kps, min_eye_dist=max(10.0, 0.18 * float(w))):
                if self.debug:
                    print("[recognize] 5pt geometry failed -> skip")
                continue

            # build bbox from kps (centered)
            bb = _bbox_from_5pt(kps, pad_x=0.55, pad_y_top=0.85, pad_y_bot=1.15)
            x1, y1, x2, y2 = _clip_xyxy(bb[0], bb[1], bb[2], bb[3], W, H)

            out.append(
                FaceDet(
                    x1=x1, y1=y1, x2=x2, y2=y2,
                    score=1.0,
                    kps=kps.astype(np.float32),
                )
            )

        return out

# ------------------------
# Matcher
# ------------------------

class FaceDBMatcher:
    def __init__(self, db: Dict[str, np.ndarray], dist_thresh: float = 0.34):
        self.db = db
        self.dist_thresh = float(dist_thresh)

        # pre-stack for speed
        self._names: List[str] = []
        self._mat: Optional[np.ndarray] = None
        self._rebuild()

    def _rebuild(self):
        self._names = sorted(self.db.keys())
        if self._names:
            self._mat = np.stack([self.db[n].reshape(-1).astype(np.float32) for n in self._names], axis=0)
# (K,D)
        else:
            self._mat = None

    def reload_from(self, path: Path):
        self.db = load_db_npz(path)
        self._rebuild()
```

```
   def match(self, emb: np.ndarray) -> MatchResult:
       if self._mat is None or len(self._names) == 0:
           return MatchResult(name=None, distance=1.0, similarity=0.0, accepted=False)

       e = emb.reshape(1, -1).astype(np.float32)  # (1,D)
       # cosine similarity since both sides are normalized: sim = dot
       sims = (self._mat @ e.T).reshape(-1)  # (K,)
       best_i = int(np.argmax(sims))
       best_sim = float(sims[best_i])
       best_dist = 1.0 - best_sim
       ok = best_dist <= self.dist_thresh

       return MatchResult(
           name=self._names[best_i] if ok else None,
           distance=float(best_dist),
           similarity=float(best_sim),
           accepted=bool(ok),
       )

# -------------------------
# Demo
# -------------------------

def main():
   db_path = Path("data/db/face_db.npz")

   det = HaarFaceMesh5pt(
       min_size=(70, 70),
       debug=False,
   )

   embedder = ArcFaceEmbedderONNX(
       model_path="models/embedder_arcface.onnx",
       input_size=(112, 112),
       debug=False,
   )

   db = load_db_npz(db_path)
   matcher = FaceDBMatcher(db=db, dist_thresh=0.34)  # from your evaluate_new output

   cap = cv2.VideoCapture(0)
   if not cap.isOpened():
       raise RuntimeError("Camera not available")

   print("Recognize (multi-face). q=quit, r=reload DB, +/- threshold, d=debug overlay")

   t0 = time.time()
   frames = 0
   fps: Optional[float] = None
   show_debug = False

   while True:
       ok, frame = cap.read()
       if not ok:
           break

       faces = det.detect(frame, max_faces=5)
       vis = frame.copy()

       # compute fps
       frames += 1
       dt = time.time() - t0
       if dt >= 1.0:
           fps = frames / dt
           frames = 0
           t0 = time.time()

       # draw + recognize each face
       # show aligned thumbnails stacked on the RIGHT, but lower to avoid overlay with green text
       h, w = vis.shape[:2]
       thumb = 112
       pad = 8
       x0 = w - thumb - pad
```

```
    y0 = 80  # moved down to avoid your text overlay area
    shown = 0

    for i, f in enumerate(faces):
        # draw bbox + kps
        cv2.rectangle(vis, (f.x1, f.y1), (f.x2, f.y2), (0, 255, 0), 2)
        for (x, y) in f.kps.astype(int):
            cv2.circle(vis, (int(x), int(y)), 2, (0, 255, 0), -1)

        # align -> embed -> match
        aligned, _ = align_face_5pt(frame, f.kps, out_size=(112, 112))
        emb = embedder.embed(aligned)
        mr = matcher.match(emb)

        # label
        label = mr.name if mr.name is not None else "Unknown"
        line1 = f"{label}"
        line2 = f"dist={mr.distance:.3f} sim={mr.similarity:.3f}"

        # color: known green, unknown red
        color = (0, 255, 0) if mr.accepted else (0, 0, 255)

        cv2.putText(vis, line1, (f.x1, max(0, f.y1 - 28)), cv2.FONT_HERSHEY_SIMPLEX, 0.8, color, 2)
        cv2.putText(vis, line2, (f.x1, max(0, f.y1 - 6)), cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2)

        # aligned preview thumbnails (stack)
        if y0 + thumb <= h and shown < 4:
            vis[y0:y0 + thumb, x0:x0 + thumb] = aligned
            cv2.putText(
                vis,
                f"{i+1}:{label}",
                (x0, y0 - 6),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.55,
                color,
                2,
            )
            y0 += thumb + pad
            shown += 1

        if show_debug:
            # show kps coords quickly
            dbg = f"kpsLeye=({f.kps[0,0]:.0f},{f.kps[0,1]:.0f})"
            cv2.putText(vis, dbg, (10, h - 20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

    # overlay header
    header = f"IDs={len(matcher._names)}  thr(dist)={matcher.dist_thresh:.2f}"
    if fps is not None:
        header += f"  fps={fps:.1f}"
    cv2.putText(vis, header, (10, 28), cv2.FONT_HERSHEY_SIMPLEX, 0.75, (0, 255, 0), 2)

    cv2.imshow("recognize_new", vis)
    key = cv2.waitKey(1) & 0xFF

    if key == ord("q"):
        break
    elif key == ord("r"):
        matcher.reload_from(db_path)
        print(f"[recognize] reloaded DB: {len(matcher._names)} identities")
    elif key in (ord("+"), ord("=")):
        matcher.dist_thresh = float(min(1.20, matcher.dist_thresh + 0.01))
        print(f"[recognize] thr(dist)={matcher.dist_thresh:.2f}  (sim~{1.0-matcher.dist_thresh:.2f})")
    elif key == ord("-"):
        matcher.dist_thresh = float(max(0.05, matcher.dist_thresh - 0.01))
        print(f"[recognize] thr(dist)={matcher.dist_thresh:.2f}  (sim~{1.0-matcher.dist_thresh:.2f})")
    elif key == ord("d"):
        show_debug = not show_debug
        print(f"[recognize] debug overlay: {'ON' if show_debug else 'OFF'}")

cap.release()
cv2.destroyAllWindows()
```

```python
if __name__ == "__main__":
    main()
```

A10: src/haar_5pt.py

```python
# src/haar_5pt.py
"""
Haar face detection + practical 5-point landmarks (MediaPipe FaceMesh).
Why this works for you:
- Haar is fast and robust on CPU.
- MediaPipe FaceMesh confirms a real face and gives stable landmarks.
- We extract ONLY 5 keypoints:left_eye, right_eye, nose_tip, mouth_left, mouth_right
- We rebuild bbox from keypoints (centered), so no "aside" offset.
- We reject Haar false positives if FaceMesh doesn't produce landmarks.
Run:
 python -m src.haar_5pt
"""

from __future__ import annotations

from dataclasses import dataclass
from typing import Optional, Tuple, List

import cv2
import numpy as np

try:
   import mediapipe as mp
except Exception as e:
   mp = None
   _MP_IMPORT_ERROR = e

# -------------------------
# Data
# -------------------------

@dataclass
class FaceKpsBox:
   x1: int
   y1: int
   x2: int
   y2: int
   score: float
   kps: np.ndarray  # (5,2) float32

# -------------------------
# Helpers
# -------------------------
def _estimate_norm_5pt(kps_5x2: np.ndarray, out_size: Tuple[int, int] = (112, 112)) -> np.ndarray:
   """
   Build 2x3 affine matrix that maps your 5pts to ArcFace-style template.
   kps order must be: [Leye, Reye, Nose, Lmouth, Rmouth]
   """
   k = kps_5x2.astype(np.float32)

   # ArcFace 112x112 template (InsightFace standard)
   # Works well for ArcFace embedder models expecting 112x112
   dst = np.array([
       [38.2946, 51.6963],  # left eye
       [73.5318, 51.5014],  # right eye
       [56.0252, 71.7366],  # nose
       [41.5493, 92.3655],  # left mouth
       [70.7299, 92.2041],  # right mouth
   ], dtype=np.float32)

   out_w, out_h = int(out_size[0]), int(out_size[1])

   # If you choose a different output size, scale template
   if (out_w, out_h) != (112, 112):
       sx = out_w / 112.0
       sy = out_h / 112.0
       dst = dst * np.array([sx, sy], dtype=np.float32)

   # Similarity transform (rotation+scale+translation)
   M, _ = cv2.estimateAffinePartial2D(k, dst, method=cv2.LMEDS)
```

```python
    if M is None:
        # use eyes only
        M = cv2.getAffineTransform(
            np.array([k[0], k[1], k[2]], dtype=np.float32),
            np.array([dst[0], dst[1], dst[2]], dtype=np.float32),
        )
    return M.astype(np.float32)


def align_face_5pt(
    frame_bgr: np.ndarray,
    kps_5x2: np.ndarray,
    out_size: Tuple[int, int] = (112, 112)
) -> Tuple[np.ndarray, np.ndarray]:
    """
    Returns (aligned_bgr, M)
    """
    M = _estimate_norm_5pt(kps_5x2, out_size=out_size)
    out_w, out_h = int(out_size[0]), int(out_size[1])
    aligned = cv2.warpAffine(
        frame_bgr,
        M,
        (out_w, out_h),
        flags=cv2.INTER_LINEAR,
        borderMode=cv2.BORDER_CONSTANT,
        borderValue=(0, 0, 0),
    )
    return aligned, M
def _clip_box_xyxy(b: np.ndarray, W: int, H: int) -> np.ndarray:
    bb = b.astype(np.float32).copy()
    bb[0] = np.clip(bb[0], 0, W - 1)
    bb[1] = np.clip(bb[1], 0, H - 1)
    bb[2] = np.clip(bb[2], 0, W - 1)
    bb[3] = np.clip(bb[3], 0, H - 1)
    return bb


def _bbox_from_5pt(kps: np.ndarray, pad_x: float = 0.55, pad_y_top: float = 0.85, pad_y_bot: float =
1.15) -> np.ndarray:
    """
    Build a face bbox from 5 keypoints with asymmetric padding:
    - more forehead (top)
    - more chin (bottom)

    This tends to look "centered" and face-like.
    """
    k = kps.astype(np.float32)
    x_min = float(np.min(k[:, 0]))
    x_max = float(np.max(k[:, 0]))
    y_min = float(np.min(k[:, 1]))
    y_max = float(np.max(k[:, 1]))

    w = max(1.0, x_max - x_min)
    h = max(1.0, y_max - y_min)

    x1 = x_min - pad_x * w
    x2 = x_max + pad_x * w
    y1 = y_min - pad_y_top * h
    y2 = y_max + pad_y_bot * h
    return np.array([x1, y1, x2, y2], dtype=np.float32)


def _ema(prev: Optional[np.ndarray], cur: np.ndarray, alpha: float) -> np.ndarray:
    if prev is None:
        return cur.astype(np.float32)
    return (alpha * prev + (1.0 - alpha) * cur).astype(np.float32)


def _kps_span_ok(kps: np.ndarray, min_eye_dist: float = 12.0) -> bool:
    """
    Quick sanity filter on 5pt geometry:
    - eye distance must be reasonable
    - mouth should be below eyes (usually)
    """
    k = kps.astype(np.float32)
    le, re, no, lm, rm = k
    eye_dist = float(np.linalg.norm(re - le))
    if eye_dist < min_eye_dist:
        return False
    # mouth should generally be below nose
    if not (lm[1] > no[1] and rm[1] > no[1]):
        return False
    return True

# -------------------------
# Detector
# -------------------------
class Haar5ptDetector:
    def __init__(
        self,
        haar_xml: Optional[str] = None,
        min_size: Tuple[int, int] = (60, 60),
        smooth_alpha: float = 0.80,
        debug: bool = True,
    ):
        self.debug = bool(debug)
        self.min_size = tuple(map(int, min_size))
        self.smooth_alpha = float(smooth_alpha)

        # Haar cascade
        if haar_xml is None:
            haar_xml = cv2.data.haarcascades + "haarcascade_frontalface_default.xml"
        self.face_cascade = cv2.CascadeClassifier(haar_xml)
        if self.face_cascade.empty():
            raise RuntimeError(f"Failed to load Haar cascade: {haar_xml}")

        # MediaPipe FaceMesh
        if mp is None:
            raise RuntimeError(
                f"mediapipe import failed: {_MP_IMPORT_ERROR}\n"
                f"Install: pip install mediapipe==0.10.21"
            )

        self.mp_face_mesh = mp.solutions.face_mesh.FaceMesh(
            static_image_mode=False,
            max_num_faces=1,
            refine_landmarks=True,
            min_detection_confidence=0.5,
            min_tracking_confidence=0.5,
        )

        # FaceMesh landmark indices for 5 points | (commonly used set; works well in practice)
        self.IDX_LEFT_EYE = 33
        self.IDX_RIGHT_EYE = 263
        self.IDX_NOSE_TIP = 1
        self.IDX_MOUTH_LEFT = 61
        self.IDX_MOUTH_RIGHT = 291

        self._prev_box: Optional[np.ndarray] = None
        self._prev_kps: Optional[np.ndarray] = None

    def _haar_faces(self, gray: np.ndarray) -> np.ndarray:
        faces = self.face_cascade.detectMultiScale(
            gray,
            scaleFactor=1.1,
            minNeighbors=5,
            flags=cv2.CASCADE_SCALE_IMAGE,
            minSize=self.min_size,
        )
        if faces is None or len(faces) == 0:
            return np.zeros((0, 4), dtype=np.int32)
        # faces are (x,y,w,h)
        return faces.astype(np.int32)

    def _facemesh_5pt(self, frame_bgr: np.ndarray) -> Optional[np.ndarray]:
        H, W = frame_bgr.shape[:2]
        rgb = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2RGB)
        res = self.mp_face_mesh.process(rgb)
        if not res.multi_face_landmarks:
            return None

        lm = res.multi_face_landmarks[0].landmark

        idxs = [
            self.IDX_LEFT_EYE,
            self.IDX_RIGHT_EYE,
            self.IDX_NOSE_TIP,
            self.IDX_MOUTH_LEFT,
            self.IDX_MOUTH_RIGHT,
        ]
        pts = []
        for i in idxs:
            p = lm[i]
            pts.append([p.x * W, p.y * H])

        kps = np.array(pts, dtype=np.float32)  # (5,2)

        # Ensure left/right ordering for eyes & mouth | (FaceMesh usually already correct, but keep safe)
        if kps[0, 0] > kps[1, 0]:
            kps[[0, 1]] = kps[[1, 0]]
        if kps[3, 0] > kps[4, 0]:
            kps[[3, 4]] = kps[[4, 3]]

        return kps

    def detect(self, frame_bgr: np.ndarray, max_faces: int = 1) -> List[FaceKpsBox]:
        H, W = frame_bgr.shape[:2]
        gray = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2GRAY)

        faces = self._haar_faces(gray)
        if faces.shape[0] == 0:
            return []

        # pick largest Haar face
        areas = faces[:, 2] * faces[:, 3]
        i = int(np.argmax(areas))
        x, y, w, h = faces[i].tolist()

        # FaceMesh confirmation + 5pt
        kps = self._facemesh_5pt(frame_bgr)
        if kps is None:
            # reject Haar false positives
            if self.debug:
                print("[haar_5pt] Haar face found but FaceMesh returned none -> reject")
            return []

        # OPTIONAL: require FaceMesh points to fall reasonably inside Haar box (prevents random FaceMesh
on background)
        margin = 0.35  # generous, because Haar can be loose
        x1m = x - margin * w
        y1m = y - margin * h
        x2m = x + (1.0 + margin) * w
        y2m = y + (1.0 + margin) * h

        inside = (
            (kps[:, 0] >= x1m) & (kps[:, 0] <= x2m) &
            (kps[:, 1] >= y1m) & (kps[:, 1] <= y2m)
        )
        if inside.mean() < 0.60:
            if self.debug:
                print("[haar_5pt] FaceMesh points not consistent with Haar box -> reject")
            return []

        if not _kps_span_ok(kps, min_eye_dist=max(10.0, 0.18 * w)):
            if self.debug:
                print("[haar_5pt] 5pt geometry sanity failed -> reject")
            return []

        # Build centered bbox from keypoints (solves your "aside" offset)
        box = _bbox_from_5pt(kps, pad_x=0.55, pad_y_top=0.85, pad_y_bot=1.15)
        box = _clip_box_xyxy(box, W, H)

        # Smooth
        box_s = _ema(self._prev_box, box, self.smooth_alpha)
        kps_s = _ema(self._prev_kps, kps, self.smooth_alpha)

        self._prev_box = box_s.copy()
        self._prev_kps = kps_s.copy()

        x1, y1, x2, y2 = box_s.tolist()

        # Haar doesn't provide a probability; use a stable placeholder score
        score = 1.0

        return [
            FaceKpsBox(
                x1=int(round(x1)),
                y1=int(round(y1)),
                x2=int(round(x2)),
                y2=int(round(y2)),
                score=float(score),
                kps=kps_s.astype(np.float32),
            )
        ][:max_faces]

# -------------------------
# Demo
# -------------------------
def main():
    cap = cv2.VideoCapture(0)

    det = Haar5ptDetector(
        min_size=(70, 70),
        smooth_alpha=0.80,
        debug=True,
    )

    print("Haar + 5pt (FaceMesh) test. Press q to quit.")
    while True:
        ok, frame = cap.read()
        if not ok:
            break

        faces = det.detect(frame, max_faces=1)
        vis = frame.copy()

        if faces:
            f = faces[0]
            cv2.rectangle(vis, (f.x1, f.y1), (f.x2, f.y2), (0, 255, 0), 2)
            for (x, y) in f.kps.astype(int):
                cv2.circle(vis, (int(x), int(y)), 3, (0, 255, 0), -1)
            cv2.putText(
                vis,
                f"OK",
                (f.x1, max(0, f.y1 - 8)),
                cv2.FONT_HERSHEY_SIMPLEX,
                0.7,
                (0, 255, 0),
                2,
            )
        else:
            cv2.putText(vis, "no face", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)

        cv2.imshow("haar_5pt", vis)
        if (cv2.waitKey(1) & 0xFF) == ord("q"):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    main()
```