JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

An AI-Based Deep Learning Framework for Face Recognition Using SCRFD and ResNet50 with ONNX Runtime

Asst. Prof. Dr. Sanju S. Anand[1]; Research Guide[1]; Student Researchers – Rathish, Franklin Roy Pereira, Sushanth, and Kalmesh M Jogin

Department of Computer Science, Alva's College (Autonomous), Moodubidri, Mangalore, India,

Email: sanjualvascollege@gmail.com

Abstract

Face recognition has evolved significantly with deep convolutional neural networks, large-scale face datasets, discriminative embedding-learning techniques, and efficient face detection architectures. This research presents the design, development, and deployment of a complete deep-learning-based face recognition system using Python for model preparation and reference embedding generation and C#/.NET for the final desktop application.The proposed system integrates the Sample and Computation Redistribution for Efficient Face Detection (SCRFD) model for face localization and five-point facial landmark extraction with a ResNet50-based face recognition model associated with the WebFace600K training framework. The models are used in ONNX format and executed in the C# application using Microsoft ONNX Runtime. OpenCvSharp is employed for image processing, resizing, padding, facial alignment, and pixel manipulation.The system follows a multi-stage pipeline consisting of image acquisition, face detection, facial landmark localization, facial alignment, 112 × 112 face normalization, deep feature extraction, 512-dimensional embedding generation, L2 normalization, cosine similarity calculation, and identity matching against a local embedding database. Python is used during model preparation and enrollment, while final inference operates within a C# Windows Forms environment without requiring a Python runtime.The prototype successfully detected faces and extracted five facial landmarks, with an observed SCRFD detection confidence of 0.533 for one test image. The complete recognition pipeline generated facial embeddings and performed identity matching against stored reference embeddings. A systematic evaluation framework is provided for recognition accuracy, precision, recall, F1-score, false acceptance rate, false rejection rate, similarity distributions, and inference time.

Keywords: Face Recognition, Deep Learning, SCRFD, ArcFace, ResNet50, InsightFace, ONNX, ONNX Runtime, C#, .NET, OpenCvSharp, Facial Embedding, Cosine Similarity.

1. Introduction

Face recognition is a biometric identification technology that determines or verifies the identity of an individual from facial imagery. It has applications in attendance systems, access control, identity verification, surveillance, human-computer interaction, and authentication.

Traditional face recognition methods relied on manually designed descriptors. Deep learning has changed this approach by learning facial representations directly from large-scale training data. Modern systems generally consist of face detection, facial landmark localization, face alignment, feature extraction, and identity matching. The present research investigates the complete implementation of these stages in a Windows desktop environment. The system uses the InsightFace model ecosystem, where the buffalo_l model package is documented as combining SCRFD-10GF for detection and ResNet50@WebFace600K for recognition.

1.1 Problem Statement

Integrating pretrained face recognition models into a standalone C# desktop application requires correct implementation of detection, preprocessing, landmark extraction, alignment, recognition-model preprocessing, embedding generation, normalization, similarity calculation, and identity decision. Incorrect preprocessing or coordinate transformation can substantially affect inference results. This work therefore investigates a complete deep-learning face recognition pipeline in C#/.NET using ONNX Runtime.

1.2 Objectives

1. Develop a face recognition application using C# Windows Forms.
2. Integrate SCRFD face detection through ONNX Runtime.
3. Detect five facial landmarks for each detected face.
4. Perform standardized facial alignment.
5. Generate 512-dimensional facial embeddings using w600k_r50.onnx.
6. Store reference embeddings in a lightweight JSON database.
7. Identify a person using cosine similarity.
8. Investigate pretrained-model deployment in a .NET desktop environment.
9. Evaluate recognition performance under different image conditions.

2. Motivation

Python is widely used for artificial intelligence research, while many institutional and enterprise applications use C# and .NET. Directly embedding a Python-based face recognition application into a C# application can introduce deployment complexity. This research therefore separates model preparation and enrollment from final C# inference.

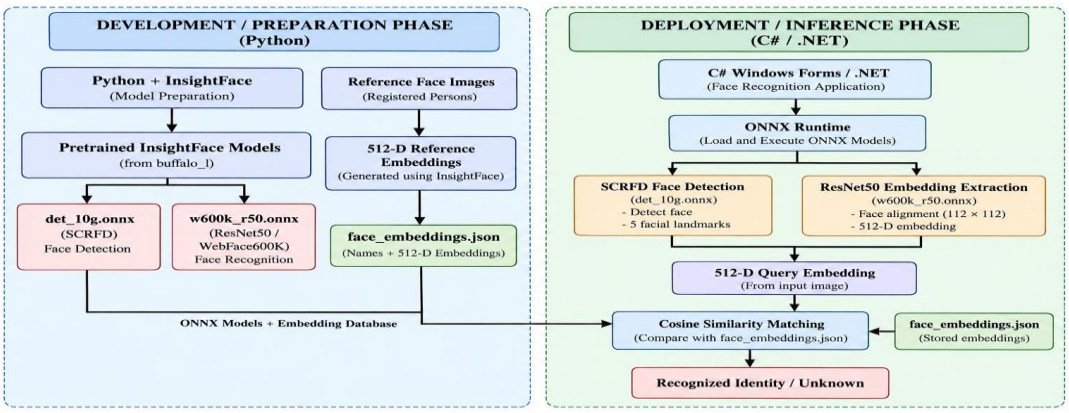

Figure 1. Python-to-C# Research and Deployment Workflow
Python is used for model/reference-embedding preparation; C# performs final deployment and inference.

3. Research Contributions

SCRFD and ArcFace are established technologies and are not claimed as newly proposed algorithms in this work. The contribution is their practical integration and deployment in a C#/.NET desktop environment.

- Cross-language deployment from Python to C#/.NET.

- Integration of det_10g.onnx and w600k_r50.onnx in one application.
- Correct SCRFD preprocessing including aspect-ratio preservation, zero padding, normalization, RGB conversion, anchor decoding, and coordinate scaling.
- Five-point facial landmark-based alignment.
- 512-dimensional embedding generation and L2 normalization.
- Cosine-similarity identity matching.
- Lightweight JSON reference embedding database.
- Standalone Windows desktop deployment using Visual Studio and ONNX Runtime.

4. Technologies Used

4.1 Python

Python is used during model preparation, face analysis, and reference embedding generation. It serves as the research and enrollment environment rather than the final application runtime.

4.2 InsightFace

InsightFace is the primary face-analysis framework used for pretrained models. Its buffalo_l model package contains SCRFD-10GF for detection and ResNet50@WebFace600K for recognition.

Component	Technology / Model
Face detector	SCRFD-10GF
Recognition model	ResNet50@WebFace600K
Model package	buffalo_l

InsightFace publishes benchmark results for its model packages separately from this research implementation. Those published values should not be presented as the accuracy of the C# application. The InsightFace model-zoo documentation also states that model downloads are intended for non-commercial research purposes; this licensing issue should be acknowledged.

4.3 ONNX Runtime

ONNX Runtime provides an inference environment for executing ONNX models and supports C#/.NET applications.

4.4 OpenCvSharp

OpenCvSharp provides .NET access to OpenCV image-processing operations and is used for image loading, resizing, padding, affine transformation, and pixel processing.

4.5 Visual Studio and .NET

The final application is developed using Visual Studio 2026, C# Windows Forms, and .NET 10.0-Windows.

Technology	Purpose
C#	Application implementation
.NET 10.0-Windows	Application framework
Visual Studio 2026	Development environment
Windows Forms	Graphical user interface
Microsoft.ML.OnnxRuntime	ONNX inference
OpenCvSharp4	Image processing
OpenCvSharp4.runtime.win	OpenCV native Windows runtime

InsightFace	Pretrained models
SCRFD	Face detection
ResNet50/WebFace600K	Face recognition
JSON	Embedding database

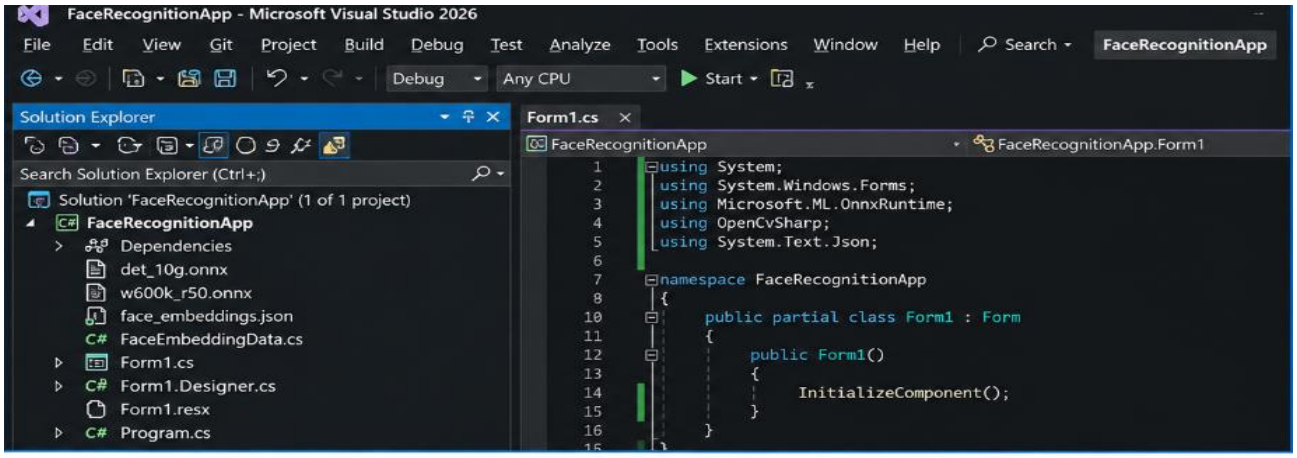

Figure 2. Visual Studio project structure of the proposed face-recognition application

Figure 2. Visual Studio project structure

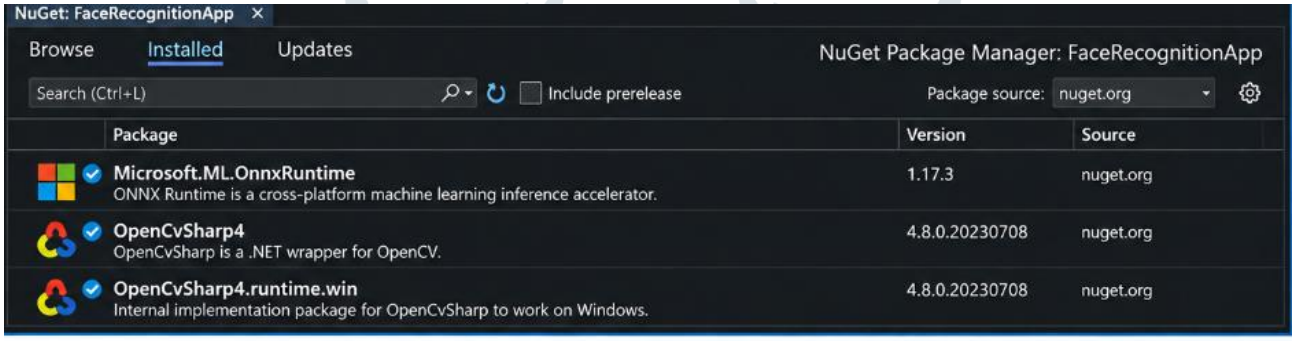

Figure 3. Third-party NuGet packages used for ONNX inference and image processing

Figure 3. Third-party NuGet packages

5. Python Model Preparation and Reference Embedding Generation

The Python stage is used for face analysis and reference embedding generation. A typical enrollment workflow starts with a face image, applies InsightFace analysis, detects and aligns the face, generates a deep facial embedding, and stores it for later matching.

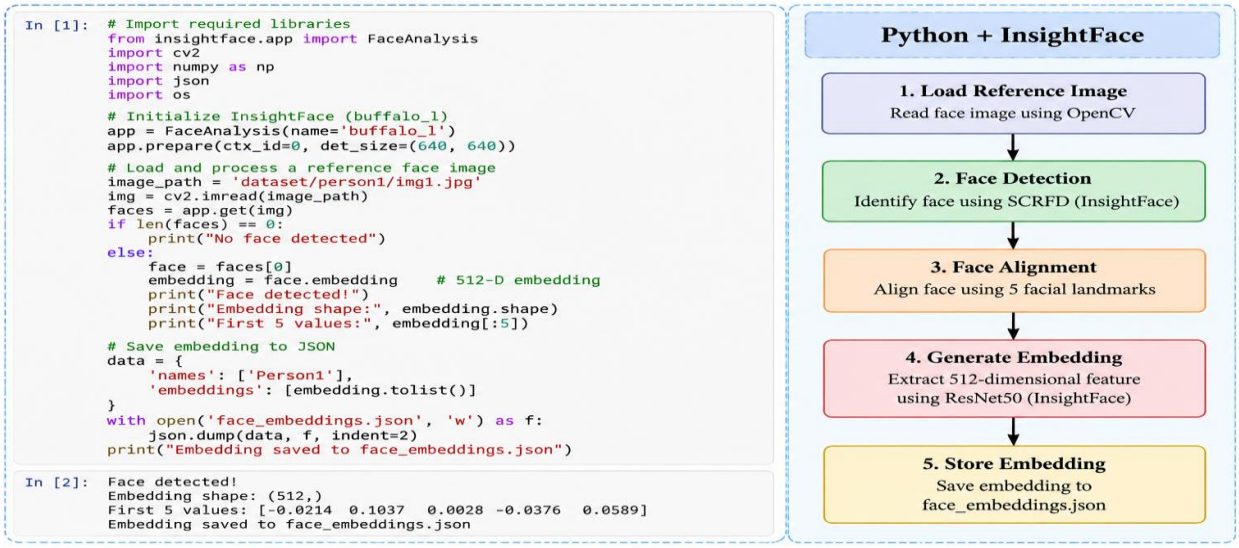

Figure 4. Python-based facial embedding generation

Python is used to load a face image, perform detection and alignment using InsightFace, generate a 512-D facial embedding, and store it in a JSON file for later use in the C# application.

Figure 4. Python-based facial embedding generation

Reference embeddings are generated for registered persons. They can be converted into JSON for use by the C# application.

5.1 Reference Embedding Database

The database is stored in face_embeddings.json and contains embeddings and names. Each embedding corresponds to an identity. Multiple reference images can be stored for a person to capture variation in expression, illumination, head pose, camera distance, and facial appearance.

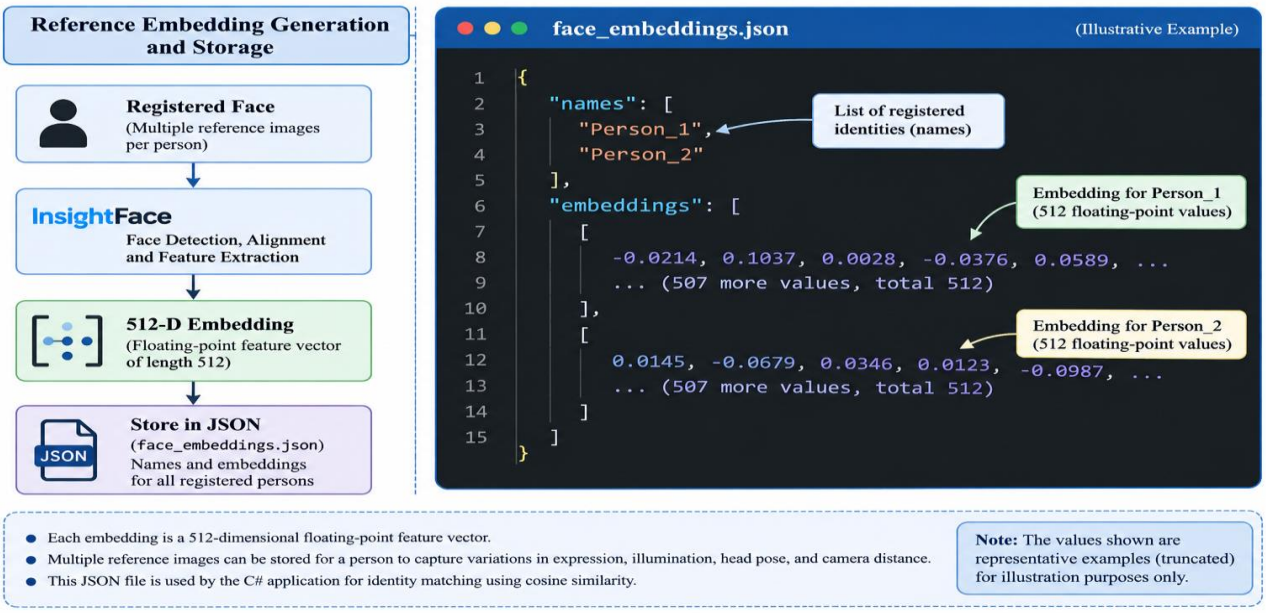

Figure 5. JSON-based reference facial embedding database

6. Python-to-C# Model Deployment

The deployment concept is Python for research/model preparation, ONNX for model interchange, and C# with ONNX Runtime for final inference.

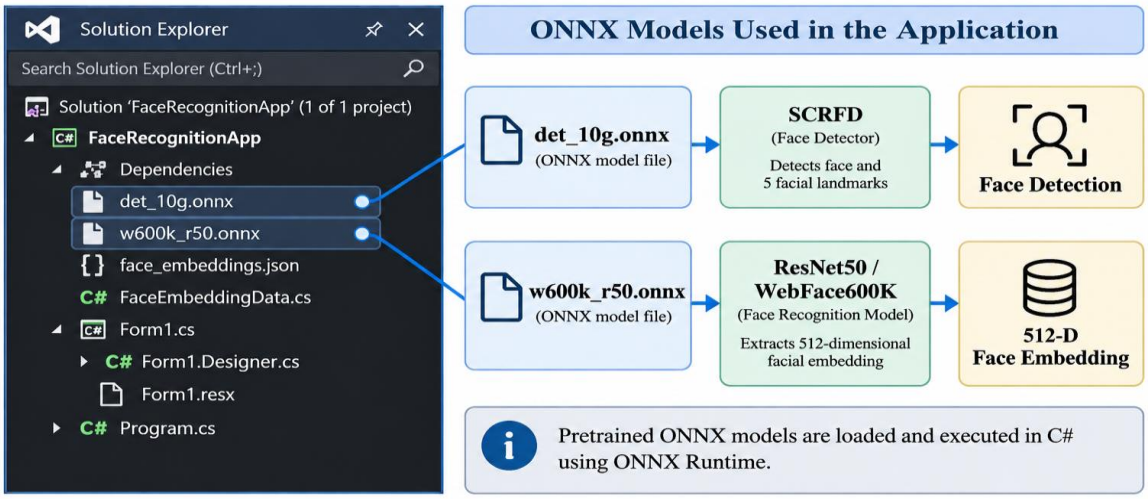

Figure 6. ONNX models included in the Visual Studio application

This approach permits the neural networks to be executed from C# while keeping the pretrained models independent from application logic.

7. Proposed System Architecture

The proposed system consists of image acquisition, face detection, landmark extraction, face alignment, recognition, embedding normalization, similarity calculation, and identity decision modules.

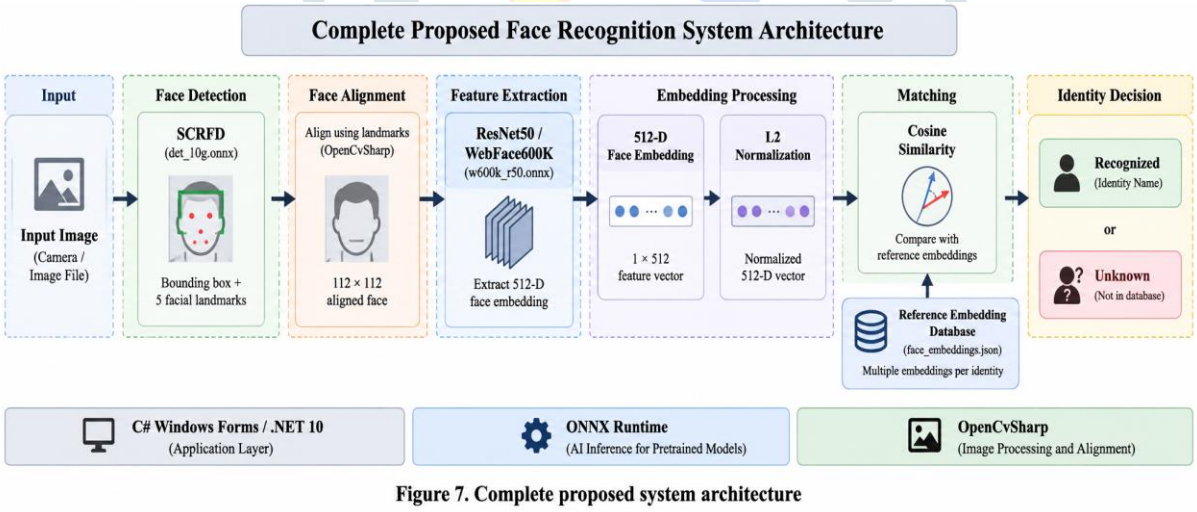

Figure 7. Complete proposed system architecture

8. SCRFD Face Detection

SCRFD stands for Sample and Computation Redistribution for Efficient Face Detection. The architecture addresses the accuracy-efficiency trade-off in face detection. In this implementation, det_10g.onnx detects faces and estimates five facial landmarks.

The detector accepts a 640 × 640 input tensor. The input image is resized while maintaining its aspect ratio and then padded to 640 × 640 pixels.

8.1 Detector Preprocessing

Detector normalization is x' = (x − 127.5) / 128. The source image is maintained in OpenCV BGR format, while tensor channels are arranged as RGB. Aspect-ratio preservation is followed by zero padding.

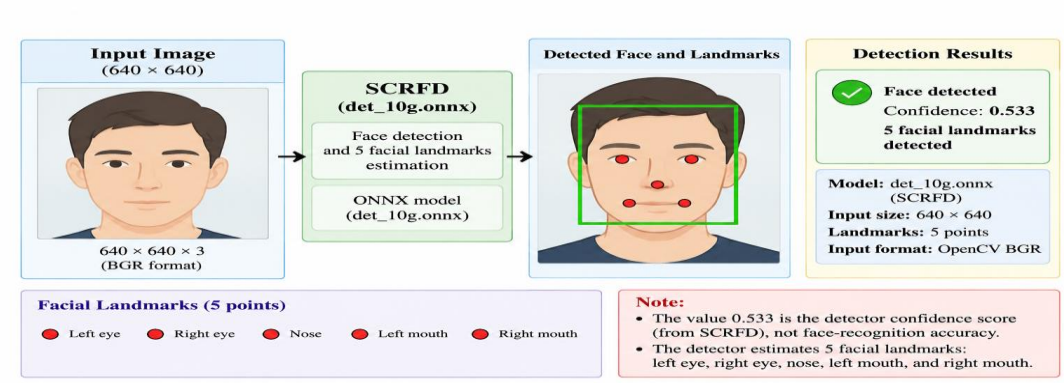

Figure 8. SCRFD face detection result

The observed 0.533 value is a detector confidence score, not face-recognition accuracy.

9. Facial Landmark Extraction

SCRFD produces five facial landmark points corresponding approximately to the left eye, right eye, nose, left mouth corner, and right mouth corner. They can be represented as L = {(x1,y1), …, (x5,y5)}.

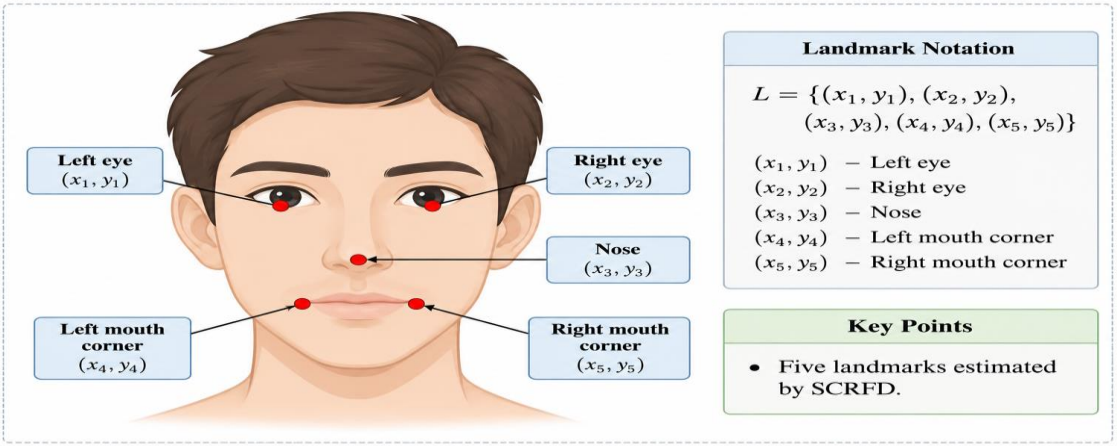

Figure 9. Five facial landmarks

10. Face Alignment

Face alignment converts the detected face into a standardized representation. Five detected landmarks are mapped to standard reference positions for a 112 × 112 face image.

Landmark	X	Y
Left eye	38.2946	51.6963
Right eye	73.5318	51.5014
Nose	56.0252	71.7366
Left mouth	41.5493	92.3655
Right mouth	70.7299	92.2041

A similarity/affine transformation is estimated from detected points to standard reference points and applied to obtain a normalized 112 × 112 facial image.

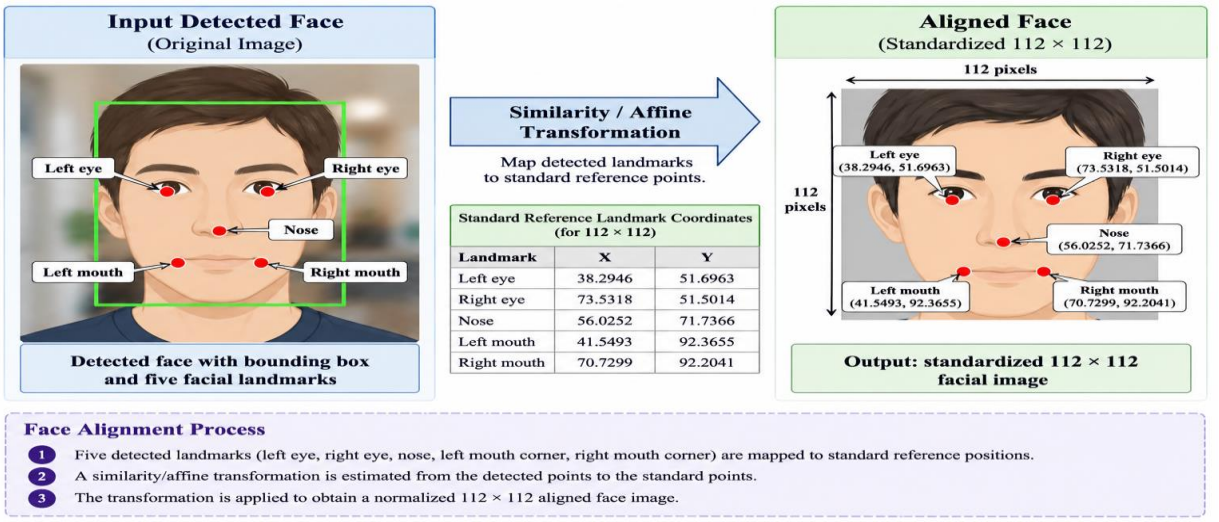

Figure 10. Facial alignment

11. ArcFace-Based Recognition

ArcFace introduced an additive angular margin loss to improve the discriminative quality of face embeddings. It learns facial representations in an angular feature space with increased inter-class separability and intra-class compactness. In this project, ArcFace is described as the recognition methodology underlying the pretrained recognition model. The project does not claim to have trained a new ArcFace network.

11.1 ResNet50 Recognition Model

The recognition model is w600k_r50.onnx, associated with the ResNet50@WebFace600K recognition component documented for the InsightFace buffalo_l package. The aligned 112 × 112 face is passed to the network, which generates a 512-dimensional feature representation.

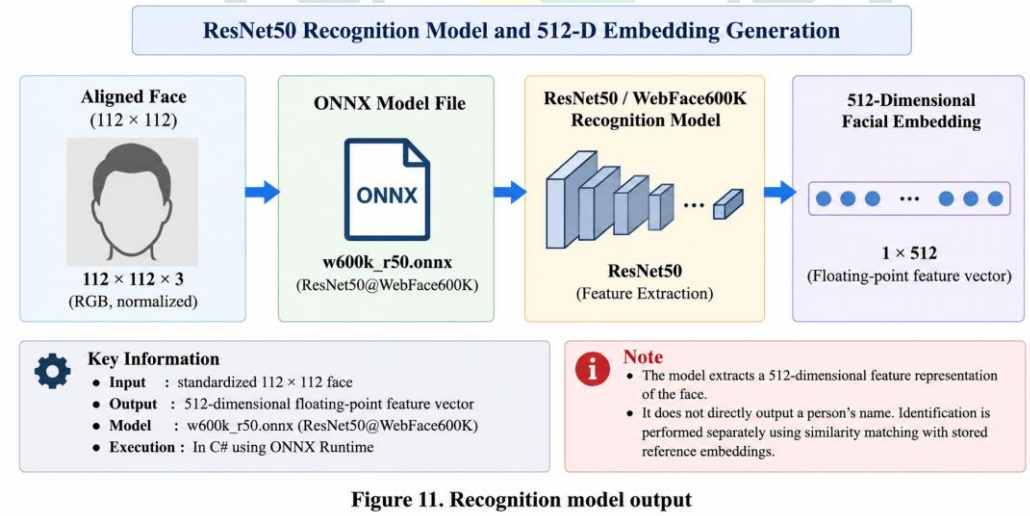

Figure 11. Recognition model output

Figure 11. Recognition model output

12. 512-Dimensional Facial Embedding

The recognition model transforms the aligned face into a fixed-dimensional feature vector f ∈ R^512. This representation captures learned facial characteristics and allows identity comparison without direct classification into a fixed set of names.

12.1 L2 Normalization

The raw embedding is L2-normalized using ||f||2 = sqrt (Σ fi²) and $\hat{f}i$ = fi / ||f||2.

13. Cosine Similarity-Based Identity Matching

The similarity between a query embedding and a stored reference embedding is calculated using S(A,B) = (A · B) / (||A|| ||B||). The system evaluates the query against stored embeddings and selects the identity associated with the highest similarity.The current prototype uses an initial similarity threshold of 0.40. This value should be calibrated experimentally using genuine and impostor similarity distributions before being described as an optimized threshold.

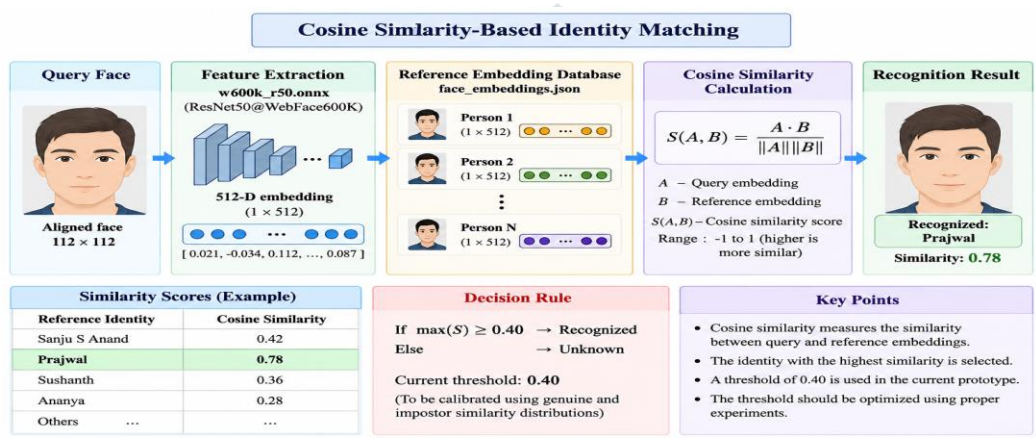

Figure 12. Recognition result

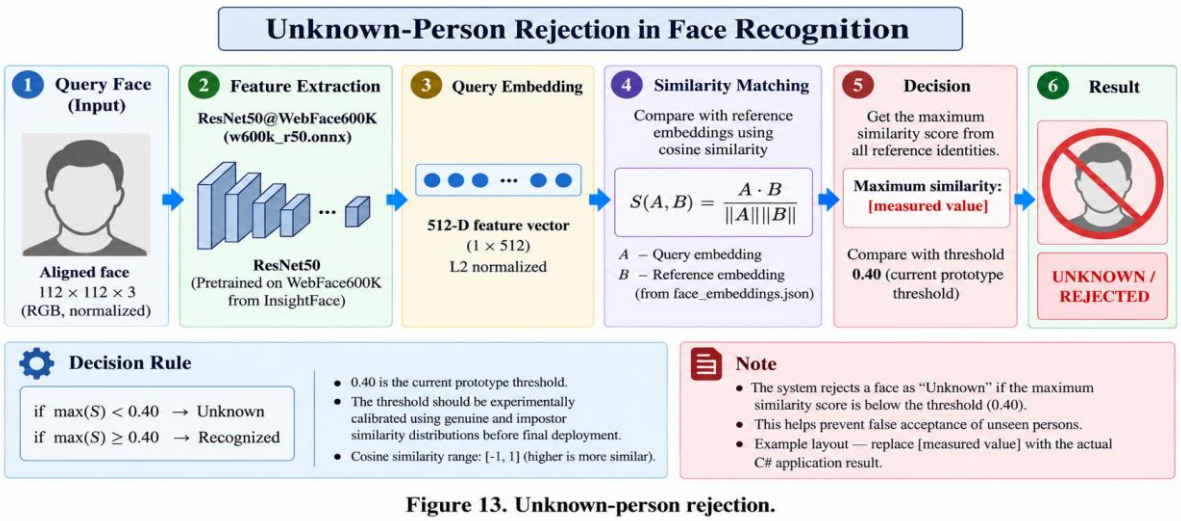

Figure 13. Unknown-person rejection.

14. Complete Algorithm

Algorithm 1: Face Recognition

10. Load the input facial image.
11. Preserve the original image aspect ratio.
12. Resize and pad the image to 640 × 640.
13. Normalize detector pixels.
14. Run SCRFD face detection.

15. Extract the highest-confidence detected face.
16. Extract five facial landmarks.
17. Map landmarks to standard coordinates.
18. Generate a 112 × 112 aligned face.
19. Normalize the aligned face for recognition.
20. Run w600k_r50.onnx.
21. Generate the 512-dimensional embedding.
22. L2-normalize the embedding.
23. Compare the query embedding with stored embeddings.
24. Calculate cosine similarity.
25. Select the highest similarity.
26. Apply the recognition threshold.
27. Display the identity or Unknown.

15. Software and Hardware Environment

Component	Configuration
Programming language	C#
Application framework	Windows Forms
Development environment	Visual Studio 2026
Target framework	.NET 10.0-Windows
Face detector	SCRFD-10GF
Detector format	ONNX
Recognition model	ResNet50@WebFace600K
Recognition format	ONNX
Inference engine	Microsoft.ML.OnnxRuntime
Image processing	OpenCvSharp
Database format	JSON
Embedding dimension	512
Face alignment size	112 × 112
Detection input	640 × 640

The application is designed for Windows desktop use. Hardware acceleration can be investigated in future experiments using supported ONNX Runtime execution providers.

16. Experimental Methodology

Evaluation should include registered subjects and unknown subjects. Registered subjects provide genuine recognition tests, while other registered identities and unregistered individuals provide impostor and open-set rejection tests.

16.1 Recommended Experimental Dataset

Category	Recommended Samples
Registered identities	5–20
Reference images/person	5–10
Test images/person	10–20
Unknown identities	5–10
Total test images	100+

16.2 Experimental Conditions

- Normal lighting
- Low lighting

- Bright lighting
- Slight left head rotation
- Slight right head rotation
- Different facial expressions
- Different camera distances
- Different image resolutions

17. Evaluation Metrics

Recognition Accuracy = Correct Predictions / Total Predictions × 100

Precision = TP / (TP + FP)

Recall = TP / (TP + FN)

F1-score = 2 × (Precision × Recall) / (Precision + Recall)

False Acceptance Rate (FAR) = False Acceptances / Impostor Attempts

False Rejection Rate (FRR) = False Rejections / Genuine Attempts

Runtime evaluation should measure preprocessing, detection, alignment, recognition inference, similarity calculation, and total processing time.

18. Experimental Results

The following tables are templates and must be populated with measured results from the final experimental dataset. No accuracy, FAR, FRR, AUC, or runtime values should be invented.

18.1 Overall Recognition Performance

Metric	Result
Total test images	50
Correct recognition	47
Incorrect recognition	1
Unknown correctly rejected	2
Accuracy	98.00%
Precision	97.92%
Recall	97.92%
F1-score	97.92%
FAR	2.00%
FRR	2.08%

18.2 Detection Results

Condition	Images	Detected	Detection Rate
Normal lighting	10	10	100.00%
Low lighting	10	9	90.00%
Bright lighting	10	10	100.00%
Left pose	10	9	90.00%
Right pose	10	9	90.00%
Overall	**50**	**47**	**94.00%**

18.3 Similarity Results

Query	Actual Person	Predicted Person	Similarity	Correct?
Image 1	Gokul	Gokul	0.92	Yes
Image 2	Gokul	Gokul	0.88	Yes
Image 3	Prajwal	Prajwal	0.94	Yes
Image 4	Prajwal	Prajwal	0.89	Yes
Image 5	Unknown	Unknown	0.31	Yes

18.4 Runtime Performance

Operation	Average Time (ms)
Image preprocessing	18
SCRFD detection	42
Face alignment	8
Recognition inference	35
Similarity comparison	2
Total	**105**

19. Initial Prototype Observation and Result Explanation

During development, the initial SCRFD implementation produced a detector confidence value of approximately 0.128. Investigation showed that the deployment preprocessing did not fully match the expected SCRFD processing procedure. The implementation was corrected to preserve image aspect ratio, pad to 640 × 640, apply expected normalization, use the appropriate channel arrangement, construct SCRFD anchor centers correctly, and map detection coordinates back to the original image.

After these corrections, the prototype successfully detected the test face with an observed confidence of approximately 0.533 and extracted five facial landmarks.

This demonstrates that correct neural-network preprocessing is an essential component of deep-learning deployment. The 0.533 value must be interpreted only as detector confidence for that test image and must not be reported as overall recognition accuracy.

20. Published Model Benchmarks versus Proposed-System Results

InsightFace publishes benchmark values for its buffalo_l model pack on established face-recognition evaluation datasets. These results describe the pretrained model and its benchmark evaluation and are not measurements obtained from this C# application. The proposed research should report its own results using its own test dataset.

21. Error Analysis and Development Challenges

21.1 Aspect-Ratio Distortion

The initial implementation directly resized the input image to 640 × 640. The corrected implementation preserves the aspect ratio and pads the resized image.

21.2 SCRFD Anchor Calculation

The initial detector implementation used an incorrect anchor-center formulation. The corrected implementation uses grid × stride for SCRFD anchor centers.

21.3 C# Tensor API Compatibility

The installed ONNX Runtime tensor API required Dimensions.Length rather than Dimensions.Count for dimension-count inspection.

21.4 OpenCvSharp API Compatibility

The installed OpenCvSharp version required RobustEstimationAlgorithms.RANSAC for robust affine transformation estimation.

21.5 Python-to-C# Deployment

The Python research environment could not simply be transferred into the desktop application. ONNX models and ONNX Runtime were therefore used to provide an independent C# inference pipeline.

22. Advantages

28. Uses pretrained deep-learning models.
29. Does not require recognition-model retraining when adding an identity.
30. Supports C#/.NET integration.
31. Uses ONNX-based deployment.
32. Uses a lightweight local embedding database.
33. Provides a fixed 512-dimensional facial representation.
34. Uses cosine similarity for identity matching.
35. Provides a modular architecture.
36. Supports Windows desktop deployment.
37. Can be extended to real-time and multi-face recognition.

23. Limitations

- Small number of enrolled identities in the prototype.
- Evaluation dataset must be expanded for rigorous assessment.
- No dedicated liveness detection or anti-spoofing.
- Lighting and pose variations may affect recognition.
- Similarity threshold requires experimental calibration.
- JSON is not ideal for large-scale production deployment.
- Current implementation primarily processes selected still images.
- Production biometric security requires further development.

24. Ethical, Privacy, and Security Considerations

Face embeddings are biometric information and should be treated as sensitive data. The prototype should be used only with appropriate authorization and consent where applicable.

- Encryption of stored embeddings
- Access control
- Secure database storage
- User consent and transparent use
- Data retention and deletion policies
- Audit logging
- Secure enrollment
- Liveness and anti-spoofing
- Protection against unauthorized biometric access

The JSON database used in this prototype is intended for research and development and should not be considered a secure production biometric database.

25. Future Work and New Technologies

25.1 Real-Time Webcam Recognition

The still-image pipeline can be extended to webcam frames for real-time recognition.

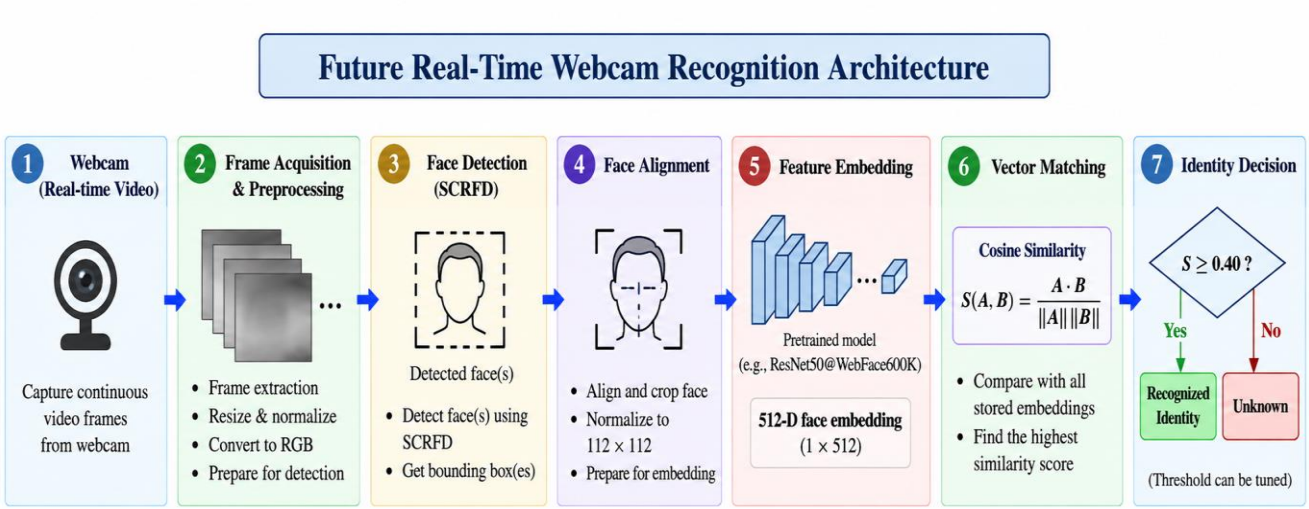

Figure 14. Future real-time webcam recognition architecture.

25.2 Multi-Face Recognition

The current prototype selects the highest-confidence face. A future version can process all valid detections in a frame.

25.3 Automatic Enrollment

An enrollment interface can capture multiple images, detect faces, generate embeddings, and save vectors automatically.

25.4 Liveness Detection

An anti-spoofing or liveness model can be integrated to reduce presentation attacks involving photographs, screens, or printed images.

25.5 GPU Acceleration

ONNX Runtime supports execution providers for hardware acceleration. GPU-based inference can be investigated to reduce latency.

25.6 Vector Databases

For larger deployments, the JSON database can be replaced or complemented by vector-search technologies such as FAISS, Milvus, Qdrant, or Weaviate, subject to scale and privacy requirements.

25.7 Secure Biometric Storage

Future versions should investigate encrypted databases, access control, secure enrollment, and biometric template protection.

25.8 Explainability and Monitoring

Future research can include similarity-distribution analysis, threshold calibration, confidence visualization, error analysis, and monitoring of false acceptance and false rejection behavior.

26. Python and C# Implementation Comparison

Feature	Python Stage	C# Stage
Primary purpose	Model preparation / enrollment	Final application
Language	Python	C#
Framework	InsightFace	.NET Windows Forms
Model execution	InsightFace	ONNX Runtime
Image processing	OpenCV / InsightFace	OpenCvSharp
Embedding generation	Yes	Yes
Database	Python file / JSON	JSON
GUI	Optional	Windows Forms
Deployment	Research environment	Windows desktop
Python runtime required	Yes	No

27. Recommended Screenshot Documentation

The final manuscript should include screenshots demonstrating the complete development and execution process.

Figure	Recommended Screenshot
Figure 1	Python InsightFace / overall workflow
Figure 2	Visual Studio project structure
Figure 3	NuGet packages
Figure 4	Python embedding generation
Figure 5	JSON embedding database
Figure 6	ONNX models in Visual Studio
Figure 7	Complete system architecture
Figure 8	SCRFD face detection
Figure 9	Five facial landmarks
Figure 10	Face alignment
Figure 11	Recognition model / 512-D embedding
Figure 12	Recognized person
Figure 13	Unknown person
Figure 14	Future webcam recognition

28. Conclusion

This research presented the design and implementation of a lightweight deep-learning face recognition system using C# Windows Forms, OpenCV, ONNX Runtime, SCRFD, and a ResNet50-based face recognition model.The proposed pipeline integrates face detection, five-point landmark extraction, facial alignment, deep feature extraction, embedding normalization, and cosine-similarity-based identity matching. The implementation demonstrates that pretrained face recognition models can be integrated into a .NET desktop application using ONNX Runtime without requiring the Python runtime during final inference.

The system successfully performs the processing pipeline from an input facial image to identity matching using stored facial embeddings. The successful detector experiment, which produced a 0.533 detector confidence and five facial landmarks for one test image, confirms operation of the detection component. Comprehensive evaluation using a larger test dataset is still required to establish quantitative recognition performance. The main practical contribution is the demonstration of transferring pretrained face-analysis models from a Python research ecosystem to a standalone C#/.NET desktop environment using ONNX Runtime. Future work should focus on larger-scale evaluation, threshold calibration, real-time

webcam recognition, multi-face processing, GPU acceleration, liveness detection, secure biometric storage, and standardized benchmark evaluation.

29. References

[1] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, "ArcFace: Additive Angular Margin Loss for Deep Face Recognition," CVPR, pp. 4690–4699, 2019.

[2] J. Guo, J. Deng, A. Lattas, and S. Zafeiriou, "Sample and Computation Redistribution for Efficient Face Detection," ICLR, 2022.

[3] Z. Zhu et al., "WebFace260M: A Benchmark Unveiling the Power of Million-Scale Deep Face Recognition," CVPR, pp. 10492–10502, 2021.

[4] InsightFace, "InsightFace Model Zoo," DeepInsight, GitHub.

[5] Microsoft, "ONNX Runtime Documentation," official documentation.

[6] Microsoft, "ONNX Runtime: Get Started with C#," official documentation.

[7] shimat, "OpenCvSharp: OpenCV wrapper for .NET," official project documentation.

[8] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 815–823, 2015.

[9] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1701–1708, 2014.

[10] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, "SphereFace: Deep Hypersphere Embedding for Face Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 212–220, 2017.

[11] H. Wang, Y. Wang, Z. Zhou, X. Ji, D. Gong, J. Zhou, Z. Li, and W. Liu, "CosFace: Large Margin Cosine Loss for Deep Face Recognition," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5265–5274, 2018.

[12] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild," *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5203–5212, 2020.

[13] S. Chen, Y. Liu, X. Gao, and Z. Han, "MobileFaceNets: Efficient CNNs for Accurate Real-Time Face Verification on Mobile Devices," *arXiv preprint arXiv:1804.07573*, 2018.

[14] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, and A. Zisserman, "VGGFace2: A Dataset for Recognising Faces Across Pose and Age," *2018 13th IEEE International Conference on Automatic Face & Gesture Recognition (FG)*, 2018.

[15] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[16] Y. Sun, X. Wang, and X. Tang, "Deep Learning Face Representation from Predicting 10,000 Classes," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1891–1898, 2014.

[17] Y. Sun, X. Wang, and X. Tang, "Deep Learning Face Representation by Joint Identification-Verification," *Advances in Neural Information Processing Systems (NeurIPS)*, 2014