Creativity and Innovation
2025, VOL. 9, NO. 1, 117-124
DOI: 10.47297/wspciWSP2516-252724.20250901

Research on Edge-based Face Recognition Systems

Zixin Cai

Dalian Maritime University, Houston International Institute, Dalian 116026, China

ABSTRACT

With the continuous advancement of artificial intelligence technology, deep learning-based face detection and recognition systems have demonstrated their importance in various scenarios. However, existing models still face limitations in computational power and energy consumption when deployed on edge devices. Therefore, this research focuses on developing a face recognition system optimized for edge-side intelligent chips, with the core objectives of achieving model lightweighting, efficiency optimization, and maintaining high recognition accuracy. This study analyzes the challenges of deploying existing models on edge devices. Based on this analysis, we select lightweight models RetinaFace and MobileFaceNet as the foundation. Techniques such as self-attention mechanisms, quantization, and pruning are employed to reduce model parameters without compromising recognition accuracy, thereby optimizing resource utilization and adapting to the computational constraints of edge devices.To validate the performance of the proposed model, experiments are conducted on the LFW (Labeled Faces in the Wild) standard face dataset. The results demonstrate that the optimized model maintains high accuracy while significantly reducing model size and inference time, meeting the deployment requirements of edge devices.

KEYWORDS

Face Recognition; Model Optimization; Edge Computing; Deep Learning; Self-Attention Mechanism; Quantify; Pruning

1 Introduction

The aim of this research is to explore and implement an efficient face recognition system for edge end intelligent chip. Our goal is to develop a face recognition system that is both efficient and accurate, enabling fast response on edge devices with low energy consumption.

In the digital era, with the rapid development of information technology, the demand for identity verification technologies continues to grow. As an efficient and contactless identification method, face recognition has demonstrated unique value. Traditional face recognition systems often rely on centralized cloud computing resources, which suffer from privacy concerns, bandwidth usage, and latency. The rise of edge computing provides a new solution by processing data near the source, reducing reliance on centralized clouds and enhancing real-time performance and privacy protection. However, edge devices typically have limited computational and storage resources, posing higher demands for deploying high-accuracy face recognition models [1]. In recent years, deep learning has become a key driver of face recognition technology. Deep convolutional neural networks (CNN) have shown exceptional performance in feature extraction and classification, greatly improving recognition accuracy. However, deep learning models often require substantial computational resources, limiting their application on edge devices. Thus, designing lightweight face recognition models that maintain high accuracy while adapting to edge-device resource constraints has become a critical research focus [2].

2 Methods for Optimizing Face Recognition Systems

2.1 Quantization

In deep learning models, weights and activations are typically stored as floating-point numbers, leading to large model sizes and intensive computations. Quantization converts these values into lower-precision representations (e. g., 8-bit integers, INT8) to compress and accelerate models.

The quantization process involves analyzing the distribution of model weights and activations. By determining the dynamic range of data, floating-point values are mapped to a smaller set of integers. This process can be expressed as a mapping function Q from the space of floating-point numbers R to the space of integers Zq:

$$Q{:}R \rightarrow Z_q \tag{1}$$

Quantization involves two steps: scaling and rounding. First, the value of the floating-point number is scaled by a scaling factor S to a predetermined range:

$$V' = S \cdot V \tag{2}$$

Where V is the original floating-point value and V 'is the scaled value. Next, apply the round operation to convert the scaled value to the nearest integer:

$$Q(V) = round(V') \tag{3}$$

Quantitative operation will traverse the weight of the model and activation, converts each floating-point value to the corresponding integer.

2.2 Pruning

Overparameterization is a common technique to improve model performance in deep learning, but it often leads to large and computationally expensive models. Pruning techniques aim to reduce the complexity of a model while trying to maintain its performance by removing some weights from a neural network. The pruning process first identifies those weights that have a small impact on the model output. These weights can be any parameter in the neural network, including but not limited to convolution kernels, bias terms, or weights in fully connected layers. The importance of the weights can be evaluated by several criteria, such as the magnitude of the weights, specific analysis of gradient-based methods [3]. In the iterative process of pruning convolutional windows, each iteration involves sorting all the convolutional windows (with the sorting metric being the weight parameters regularized by L1 in the convolutional kernels). The convolutional windows with the lowest metrics after sorting are discarded to achieve the goal of pruning. Then, the model is trained using the pruned convolutional windows, and this process is repeated continuously. Pruning these low-contributing neurons will cause a certain loss in the model's accuracy. Therefore, the pruned model usually requires more training to ensure a certain level of performance. Therefore, the pruning of the model is an iterative process; the iterative process is the alternating repetition of pruning and model training.

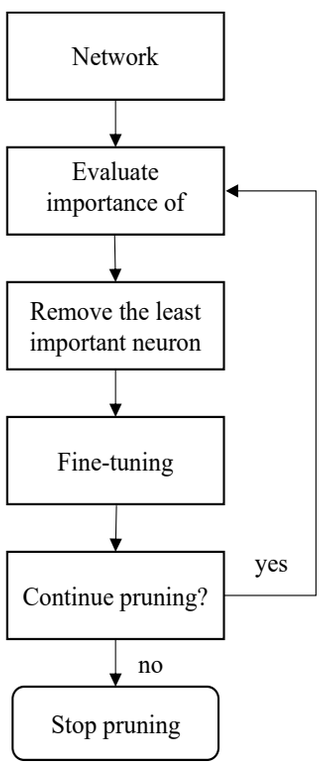

Figure 1 The basic steps of pruning [Owner-draw]

The pruning algorithm sets unimportant weights to zero or directly removes them from the model. This process can be represented as a mapping from the original set of weights W to the pruned set of weights W':

$$W' = P(W) \tag{4}$$

Where P is the pruning operation, which removes or zeroizes unimportant parts of the set of weights W.

2.3 Self-Attention Mechanism

In the self-attention model, we first represent the input sequence X as a sequence of feature vectors x_i, where each x_i belongs to the original space L and is mapped to a high-dimensional feature space R^{nf}. This mapping process can be expressed as

$$\phi: L \rightarrow R^{nf} \tag{5}$$

The core of the self-attention mechanism is to calculate the attention weight of each element to all other elements in the sequence. This process involves three main vector representations: Query, Key, and Value, which are all obtained from

the input feature vector x_i through different linear transformations.

For any two elements x_i and x_j in the sequence, the self-attention mechanism first computes the dot product of their corresponding query and key to obtain a raw attention score:

$$Score\left(x_i, x_j\right) = \phi_Q\left(x_i\right)^T \cdot \phi_K\left(x_j\right) \tag{6}$$

The final attention weight α_{ij} is then obtained by applying the softmax function to normalize all raw attention scores:

$$\alpha_{ij} = \frac{\exp\left(Score\left(x_i, x_j\right)\right)}{\sum_{k-1}^{n} \exp\left(Score\left(x_i, x_k\right)\right)} \tag{7}$$

Next, the value $\phi_V(x_j)$ of each element is multiplied with its corresponding attention weight and summed to obtain the weighted output feature vector o_i:

$$o_i = \sum_{j=1}^{n} \alpha_{ij} \cdot \phi_V\left(x_j\right) \tag{8}$$

Finally, the output feature vector o_i represents the aggregate information of the i-th element in the sequence after considering all other elements [4].

3 Face Recognition Model

3.1 RetinaFace

As an advanced face detection algorithm, RetinaFace provides a novel perspective to tackle the task of face detection in images. The core of the RetinaFace mechanism is to utilize a multi-task learning framework to simultaneously predict the face score, face frame, face key points, and the 3D position and correspondence of each face pixel. This process involves several key network architectures: the Feature Pyramid Network (FPN), the Context Module, and the multi-task loss function. For any potential face region in the image, RetinaFace first extracts a multi-scale feature map through the feature pyramid network, and then uses the context module to enhance the representation ability of the features. Specifically, the feature pyramid network combines different levels of feature maps in a top-down and lateral connection manner to adapt to face detection tasks of different sizes. Next, RetinaFace partitions the image through a series of anchors, and performs classification and regression tasks on each anchor. The classification task is used to determine whether the anchor contains a face, while the regression task is used to accurately estimate the position, size, and pose of the face. In addition, RetinaFace also introduces a facial key point regression task to further improve the accuracy of face detection. Finally, RetinaFace through a novel multitasking loss function to balance the training between different tasks, the loss of loss function including classification, bounding box key points return loss, return loss and return loss. This design enables RetinaFace to maintain high robustness and accuracy in complex scenes.

In multi-task learning, RetinaFace's loss function can be expressed as follows.

$$L = L_{cls}\left(p_i, p_i^*\right) + \lambda_1 p_i^* L_{box}\left(t_i, t_i^*\right) + \lambda_2 p_i^* L_{pts}\left(l_i, l_i^*\right) + \lambda_3 p_i^* L_{mesh}\left(v_i, v_i^*\right) \tag{9}$$

Where *Lcls* is the classification loss, *Lbox* is the bounding box regression loss, *Lpts* is the keypoint regression loss, and *Lmesh* is the 3D face mesh regression loss. *pi* denotes whether the anchor box contains the prediction of the face, *pi*∗ is the true label, $\lambda 1, \lambda 2, \lambda 3$ are the weight coefficients for different tasks [5].

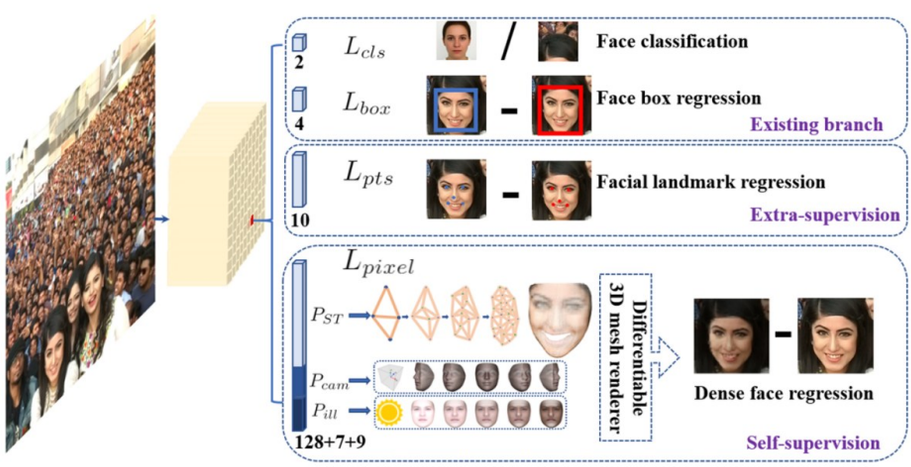

Figure 2 RetinaFace Network structure [5] Y. Zhang, Y. Liu, and Q. Yang, "A Survey of Multi-Task Learning," arXiv preprint arXiv:1905.00641, 2019. [Online]. Available:https://arxiv.org/pdf/1905.00641.

3.2 MobileFaceNet

As a lightweight network specifically optimized for face recognition tasks, MobileFaceNet provides an efficient perspective to deal with facial feature extraction. The core of MobileFaceNet model involves several key network structures: Depthwise Separable Convolution (DCNN), Global Depthwise Convolution (GDConv), and lightweight network design.

For the input face image, MobileFaceNet first extracts features by depthwise separable convolution. This step utilizes the Inverted Residual structure in MobileNetV2 to reduce the amount of calculation and the number of parameters of the model. The calculation process of depthwise separable convolution can be expressed as follows.

$$F_{out} = Pointwise(Depthwise(F_{in})) \tag{10}$$

Where *Fin* is the input feature map, Depthwise represents the depth-wise convolution operation, and Pointwise represents the pointwise convolution operation.

Then, MobileFaceNet uses Global depth convolution to replace the traditional Global Average Pooling layer (GAP). This innovation enables the model to better capture the spatial information in facial features. Global depth convolutions (GDConv) can be calculated as follows.

$$Gm = \sum_{i=1}^{W} \sum_{j=1}^{H} K_m(i,j) \cdot F(i,j,m) \tag{11}$$

Where *F* is the input feature map, *Km* is the depth convolution kernel of the *m*-th channel, and *Gm* is the *m*-th channel of the output feature map.

Finally, MobileFaceNet uses a linear layer to convert the output of the global depth convolution into a fixed-length feature vector, which can be used for subsequent face matching and recognition tasks. This step can be represented by the following formula:

$$z = W \cdot G + b \tag{12}$$

Where *G* is the output of the global depth convolution, *W* and *b* are the weights and biases of the linear layer, respectively [6].

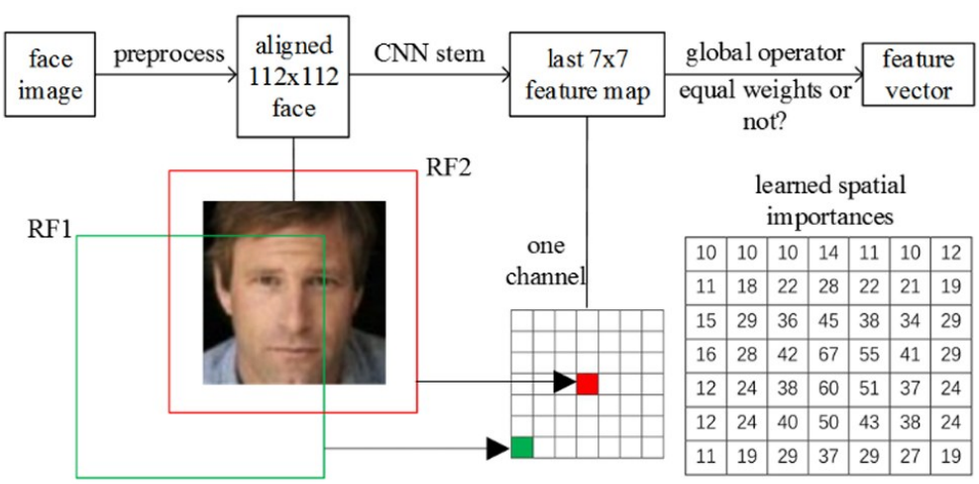

Figure 3 MobileFaceNet Network structure [2] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," arXiv preprint arXiv:1804.07573, 2019. [Online]. Available:https://arxiv.org/pdf/1804.07573

4 Design of face recognition system

4.1 Face recognition system frame design

The overall framework of face recognition system mainly includes two core modules: face information acquisition module and face recognition module [7]. As shown in FIG. 4, the workflow of the system can be divided into two main stages.

On the one hand, the system collects personnel image data and constructs the initial personnel image database. Then, the face detection algorithm is used to locate the face area from the image and generate the face feature vector. Finally, the face feature database is constructed as a reference benchmark for subsequent recognition tasks.

On the other hand, the system collects the image information of the person to be recognized, and performs the same preprocessing operation on it, including face detection and feature extraction, to obtain the face feature vector to be recognized. Finally, the system compares the feature vector to be recognized with the feature vector in the feature database, judges the matching result according to the preset threshold, and outputs the final recognition conclusion.

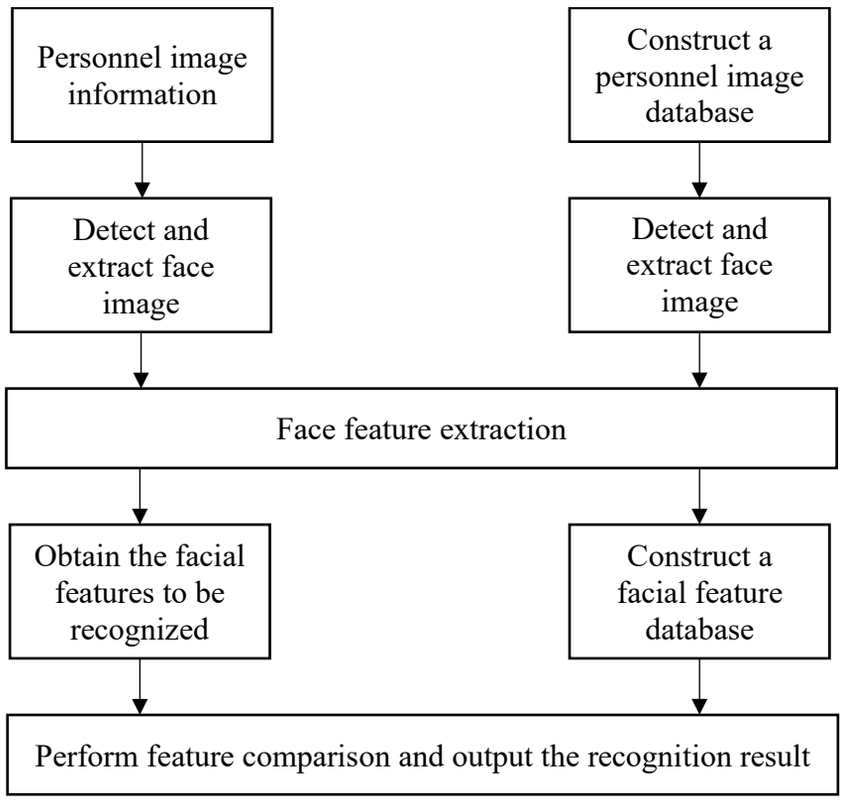

Figure 4 Diagram of the facial recognition system implementation [Owner-draw]

4.2 Design of the facial information acquisition module

The core function of the facial information acquisition module is to collect and extract the facial feature vectors to be recognized, and integrate them into the newly built local facial feature database to provide data support for subsequent face recognition tasks. FIG. 5 shows the workflow of the module.

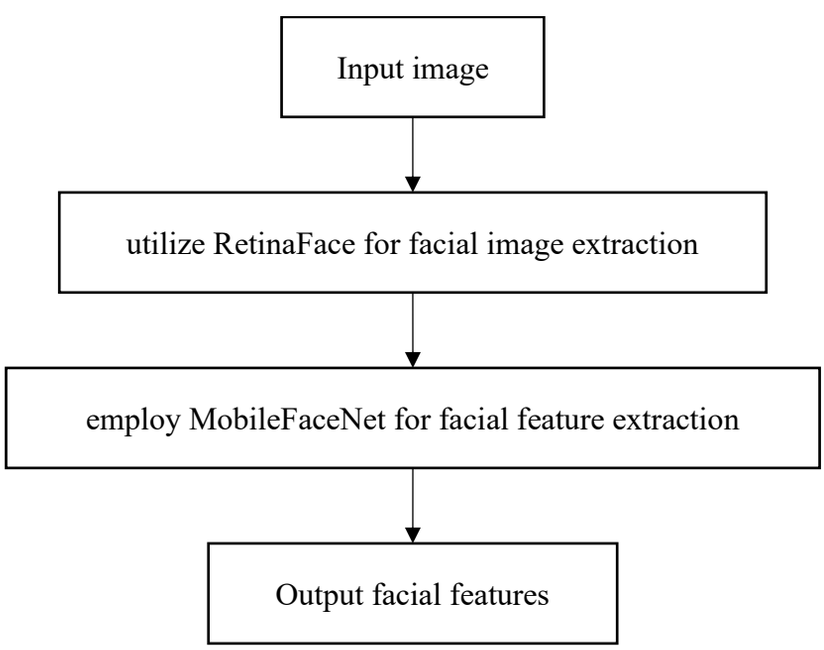

Figure 5 Facial information collection flowchart [Owner-draw]

The specific work and implementation methods are as follows:

1) Personnel image acquisition module. The system establishes the corresponding relationship between the person name label and the image data for the input image, and forms a preliminary person image data set.

2) Face detection and extraction module. The face detection and extraction function are implemented based on the RetinaFace model. The model accurately locates the face area through bounding box regression, and outputs the

coordinates of facial key points. The detected face images are standardized and stored in the local face image database to provide a data basis for subsequent feature extraction.

3) Facial feature extraction module. For the local face image database, the MobileFaceNet model is used to extract the face feature vector. The proposed model significantly reduces the computational complexity while ensuring the accuracy of feature extraction. After each face image passes through the network, it will be mapped into a 128-dimensional feature vector, which can effectively represent the key feature information of the face.

4) Building blocks of face feature database. The database uses a structured storage scheme to store 128-dimensional feature vectors associated with their corresponding name tag information. Specifically, feature vectors are stored line by line in the form of floating-point numbers, and label information is recorded in string format. Both are persistently stored in standardized txt file format to ensure data portability and scalability.

4.3 Design of Face Recognition Module

The core function of the face recognition module is to realize the identity recognition of the person to be detected. It extracts face feature information through the image of the person to be detected for comparison. As shown in Figure 6, the workflow of this module can be divided into the following steps.

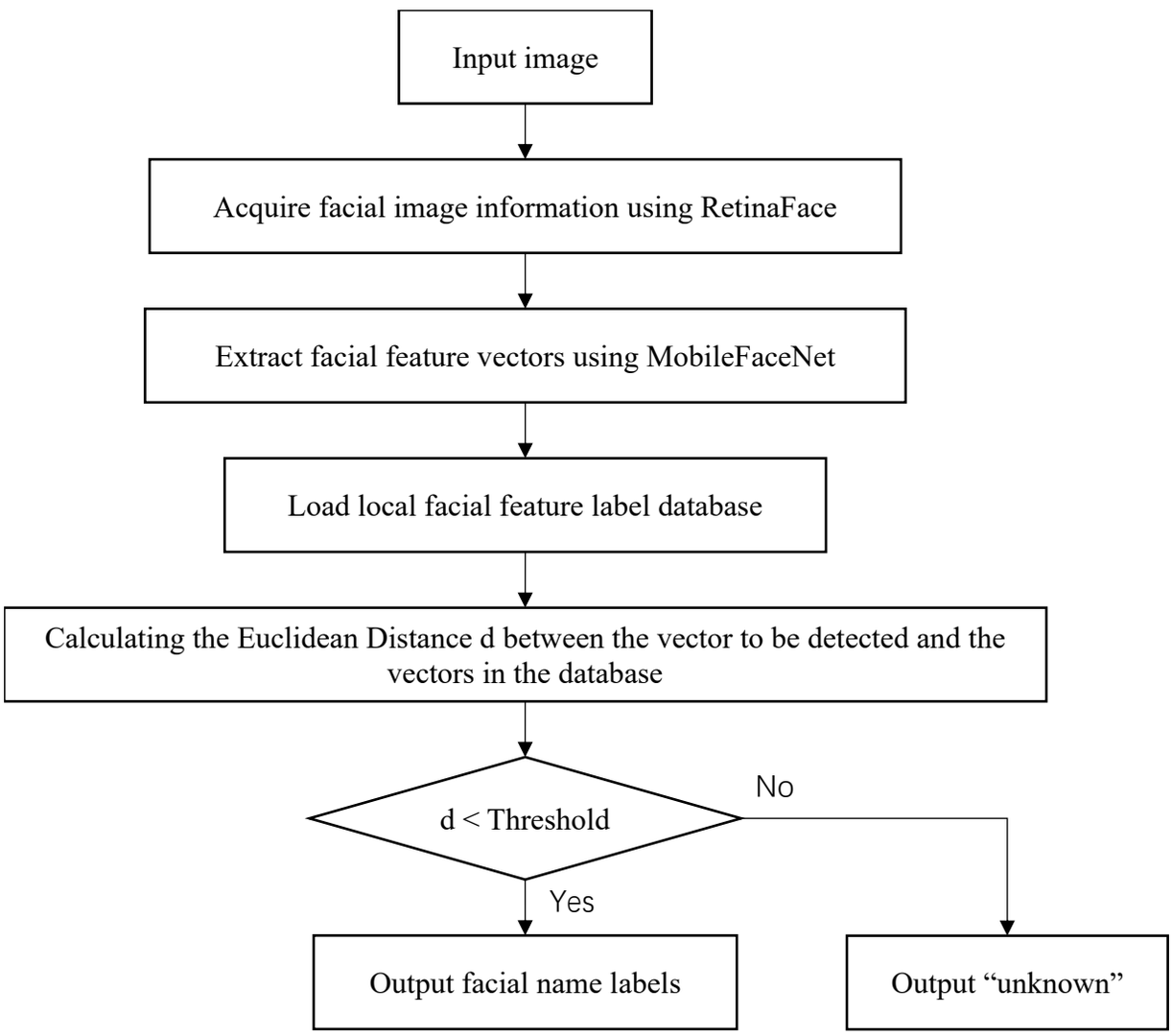

Figure 6 Facial recognition flowchart [Owner-draw]

1) Facial feature library loading module. The system first loads the local facial feature database generated by the face information acquisition module. The person name tag information is loaded in JSON format, and the face feature vector is loaded in matrix form through NumPy scientific computing library to improve the operation efficiency of feature comparison.

2) Feature matching algorithm. The module uses feature matching algorithm based on distance measurement to realize identity recognition. Specifically, the system calculates the Euclidean Distance between the feature vector of the

face to be detected and all the registered vectors in the feature database, which is used as the similarity measure. By finding the minimum distance value, the most probable matching result is determined. The mathematical expression for this method is:

$$d(x,y) = \sqrt{\Sigma(x\_i - y\_i)\text{^}2} \tag{13}$$

Where x represents the feature vector to be detected, and y represents the reference vector in the feature library.

3) Determination and output of recognition results. The system determines the matching results by comparing the Euclidean distance d between the face vector to be detected and the feature vector of the local face database loaded with the set threshold. If the minimum Euclidean distance d satisfies $d < \tau$, it is judged as a successful matching, and the corresponding name tag information is output. Otherwise, it is determined as an unknown person and returns the "unknown" flag [8].

5 Experimental Methodology

5.1 Experimental Environment

To verify the performance of the proposed face recognition system based on RetinaFace and MobileFaceNet, this study designs a systematic face recognition experiment. the widely recognized face recognition benchmark dataset LFW (Labeled Faces in the Wild) is used as the test data. The experimental environment configuration is shown in Table 1, which is divided into two parts: hardware and software.

Table 1 Configuration Information Table

Categories	Name	Configuration
Hardware	CPU	Intel(R) Core(TM) i7-12650H
	GPU	NVIDIA GeForce RTX 4060
Software	Operating System	Windows 11
	Programming Language	Python
	Deep Learning Framework	TensorFlow

5.2 Experimental results and analysis

5.2.1 Detection speed test

In this system, the detection speed is evaluated by calculating the running time of the face detection module, and the system performance is measured by counting the number of images processed per second (f/s). According to relevant research, when the detection speed of the application reaches 20 f/s or more, it can meet the real-time requirements. To verify the performance of the system, the detection method of a fixed number of face images is used in the experiment. The detection speed is evaluated by counting the total detection time and calculating the average number of images processed per second. The experimental data comes from the LFW, and 2000 images are randomly selected for detection each time, and a total of 4 groups of independent tests are performed. The experimental results are shown in Table 2.

Table 2 Experimental Results of Detection Speed

Group	Number of Images	Time Consumed /s
1	2000	60.52
2	2000	63.43
3	2000	62.17
4	2000	65.34

According to the analysis of the statistical results in Table 2, the average time consumed by the system in detecting 2000 images is 63 seconds, the average detection time of a single image is 31 milliseconds, and the detection speed reaches more than 30 f/s. The experimental results show that the system can efficiently process the face detection task and fully meet the needs of real time detection.

5.2.2 Detection accuracy test

In this experiment, the face recognition accuracy of the system is evaluated by the test set image recognition, and the method is to determine whether the given image pair belongs to the same identity. The LFW was used in the experiment, and 2000 pairs of face images were randomly selected to construct the test set, of which 50% were positive sample pairs of the same identity and 50% were negative sample pairs of different identities. The Euclidean distance between the feature vectors was calculated, and the discrimination threshold was set for matching determination. The experimental

results show that (as shown in Table 3), when the threshold is set to 0.60, the system achieves the optimal recognition accuracy of 98.75%.

Table 3 Face Recognition Rate under Different Thresholds

Threshold	Accuracy Rate /%
0.5	96.15
0.55	97.51
0.6	98.75
0.65	97.60
0.7	95.54

To verify the superiority of the proposed fusion algorithm based on RetinaFace and MobileFaceNet in recognition accuracy, the current mainstream face recognition models FaceNet and DeepFace are selected as the comparison benchmarks. The unified LFW test set (2000 pairs of samples) is used in the experiment, and the comparison test is carried out under the same hardware environment. The Euclidean distance measurement method is used for feature matching, and the threshold is uniformly set to 0.60. As shown in Table 4, the recognition accuracy of the proposed algorithm is better than that of the comparison models.

Table 4 Recognition Accuracy of Different Models

Model	Accuracy Rate /%
FaceNet	97.85
DeepFace	98.13
Proposed Model	98.75

6 Conclusion

This research is devoted to the development of an efficient and lightweight face recognition system for edge devices. Aiming at the limitations of edge devices in computing power, power consumption and real-time performance, an optimization scheme based on RetinaFace and MobileFaceNet is proposed. The balance between model performance and resource efficiency is achieved through quantization, pruning and self-attention mechanism. Experimental results show that the optimized system performs well on LFW: the detection speed meets the real-time requirements. The recognition accuracy reaches 98.75%. These results confirm that the system has significant performance.

References

[1] P. Sun, "Research and application progress of face recognition technology," Sci. & Tech. Communication, 11, 24, 130-131, 2019.

[2] J. Wang, K. Zou, and Y. Chen, "Face recognition based on convolutional neural network," Computer Knowledge and Technology, 12, 29, 187-190, 2016.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res, 15, 1, 1929-1958, 2014. [Online]. Available: https://www. semanticscholar. org/paper/Dropout% 3A-a-simple-way-to-prevent-neural-networks-Srivastava-Hinton/34f25a8704614163c4095b3ee2fc969b60de4698?p2df

[4] A. Vaswani et al., "Attention is all you need," in Advances in Neural Information Processing Systems, 2017, 5998–6008.

[5] Y. Zhang, Y. Liu, and Q. Yang, "A Survey of Multi-Task Learning," arXiv preprint arXiv:1905.00641, 2019. [Online]. Available: https://arxiv.org/pdf/1905.00641.

[6] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," arXiv preprint arXiv:1804.07573, 2019. [Online]. Available: https://arxiv.org/pdf/1804.07573.

[7] G. Zhang, J. He, and W. Gao, "Research on face recognition based on deep learning," Wireless Internet Technol, 16, 19, 133-135, 2019.

[8] L. Deng, G. Xu, M. Li, et al., "Fast face recognition based on deep learning and multi-hash similarity weighting," Computer Science, 47, 9, 163–168, 2020.