


Article

Distributed Face Recognition Based on Load Balancing and Dynamic Prediction

Fangyuan Zou ^{1,†}, Jing Li ^{1,†} and Weidong Min ^{2,*} 

¹ School of Information Engineering, Nanchang University, Nanchang 330031, China; ncuzoufangyuan@163.com (F.Z.); jingli@ncu.edu.cn (J.L.)

² School of Software, Nanchang University, Nanchang 330047, China

* Correspondence: minweidong@ncu.edu.cn; Tel.: +86-791-8830-4080

† The first two authors contributed equally to this work.

Received: 11 December 2018; Accepted: 4 February 2019; Published: 24 February 2019



Featured Application: A distributed face recognition system can quickly handle a large number of face recognition tasks, but the problem of load imbalance may lead to time delay and sharp increase of CPU utilization. To deal with the problem, this paper proposes a dynamic load balancing method based on prediction, which can effectively alleviate the time delay and sharp increase of CPU utilization. Distributed face recognition can be widely used in many important research fields and industries such as public security, customs, finance, security. Also, the method proposed in this paper can help provide new ideas for solving task assignment and load balancing problems in other distributed systems such as speech recognition, fingerprint recognition, and vehicle identification.

Abstract: With the dramatic expansion of large-scale videos, traditional centralized face recognition methods cannot meet the demands of time efficiency and expansibility, and thus distributed face recognition models were proposed. However, the number of tasks at the agent side is always dynamic, and unbalanced allocation will lead to time delay and a sharp increase of CPU utilization. To this end, a new distributed face recognition framework based on load balancing and dynamic prediction is proposed in this paper. The framework consists of a server and multiple agents. When performing face recognition, the server is used to recognize faces, and other operations are performed by the agents. Since the changes of the total number of videos and the number of pedestrians affect the task amount, we perform load balancing with an improved genetic algorithm. To ensure the accuracy of task allocation, we use extreme learning machine to predict the change of tasks. The server then performs task allocation based on the predicted results sent by the agents. The experimental results show that the proposed method can effectively solve the problem of unbalanced task allocation at the agent side, and meanwhile alleviate time delay and the sharp increase of CPU utilization.

Keywords: face recognition; distributed system; load balancing; dynamic prediction; extreme learning machine; genetic algorithm

1. Introduction

Face recognition [1] is a hot topic in the computer vision research field and has been widely applied in various fields, such as identity authentication, security, and access control systems. With the rapid development of internet, more and more data are emerging and being shared, including increasing video data. The centralized face recognition system needs to deal with all the data, operations, and processing tasks, which is always limited by the performance of the central computer. To overcome the limitation, a higher-performance computing architecture based on GPU (Graphics Processing Unit)

is proposed [2]. Instead, some researchers turned to designed distributed systems for face recognition. Min et al. [3] accomplished behavior analysis of pedestrians through lost items in a distributed architecture and obtained good performance. The distributed face recognition system can not only synchronize face recognition to multiple servers and clients, but also deal with multiple videos at the same time. It is highly efficient and can handle a large-scale face dataset very well.

Jiang et al. [4] proposed a distributed parallel face recognition system which allocates a large-scale face database to multiple subordinate servers, and executes recognition simultaneously by these subordinate servers. This system not only can reduce overhead when adding or subtracting nodes in the array structure, but also increase infinite scalability. However, it is only suitable for large-scale face databases, and takes up a lot of hardware resources. A distributed face recognition model in wireless sensor networks was proposed in [5,6]. The model divides a facial image into four sub-images based on facial features, and then extracts and identifies the features of sub-images separately. Similarly, another distributed framework based on multi-block local binary patterns and data fusion was proposed in [7], which divides the face area into four parts, namely two eyes, nose and mouth. Then, the multi-scale block local binary patterns are extracted from these regions to obtain both locally and globally informative features, and the distributed framework is introduced to accelerate the recognition process. It can improve the efficiency of face recognition in various lighting environments without reducing the recognition accuracy. Nevertheless, in this framework, the division of the face area is generally empirical and requires accurate manual segmentation. What is more, it also needs to deal with the problem of dividing face regions to ensure the illumination in each area is uniform.

The uniform distributed face recognition system [8] could quickly recognize faces in videos based on the client/server technology. It improves the face recognition efficiency by processing each video frame in parallel, but can require heavy computer resources. The abovementioned distributed face recognition systems can improve the face recognition efficiency with multiple servers or clients, but at the price of increasing the computational complexity, and essentially they all achieved face recognition based on images rather than videos. To this end, the agent-based distributed face recognition model was proposed in [9]. The model consists of three layers, i.e., a central agent host layer for face detection, a neighboring agent host layer for feature extraction, and a remote agent host layer for face matching in parallel computing environments. In this way, the system has improved scalability and can handle a growing amount of tasks. However, when the amount of tasks among agents is quite different, the agent with more tasks will have time delay and a sharp increase in CPU utilization, thus reducing the overall efficiency. Since the number of pedestrians appearing in the video changes over time, during the task assignment process, the amount of tasks handled by the agent changes dynamically. These changes will result in an unbalanced task allocation for the agent, which means dynamic task balancing is very important for the agent side.

In view of these problems, a distributed face recognition framework based on load balancing and dynamic prediction is proposed in this paper. The distributed framework is composed of a server and multiple agents. The agents are responsible for face detection, tracking and feature extraction in the videos, and then pass obtained data to the server. The server performs face matching and recognition, and then returns recognition results to the agents. Considering the problem of unbalanced load among multiple agents, an improved genetic algorithm is proposed for task balancing, which is essentially similar to task scheduling. Genetic algorithms are commonly used for solving task scheduling problems [10–12], and are also suitable for solving load balancing problems [13–15]. The algorithms are very good at global searching, and can quickly search the optimal solution in the solution space without falling into a local optimal solution. In addition, they have the capability of parallel processing, which can easily perform distributed computing and speed up the solution. However, the above-mentioned genetic algorithms have the disadvantages of premature convergence and instability, and reducing diversity when choosing to preserve good chromosomes. To alleviate these problems, this paper improves the genetic operation in the genetic algorithms by limiting the crossover probability and mutation probability. Moreover, a threshold function is added to reduce the

number of iterations. In order to allocate tasks to the agents more accurately and dynamically, we use extreme learning machine (ELM) to predict the amount of agent-based tasks before equalization. Afterwards, task allocation is performed according to the predicted value. Experimental results show that the optimization method can effectively improve the time efficiency and scalability of the agent-based distributed face recognition framework.

The rest of this paper is organized as follows. Section 2 introduces the related works. Section 3 illustrates the distributed face recognition model. Section 4 provides the improved genetic algorithm for load balancing. Section 5 mainly describes the performance optimization method of the distributed face recognition framework. Section 6 discusses experiments and results. Section 7 concludes.

2. Related Work

Face recognition [16–18] is an important research topic in computer vision and has received substantial attention from both research communities and the industry. Nowadays, face recognition has been able to recognize faces with high accuracy in real-time monitoring [19–22], e.g., face recognition time attendance machines in some companies or supermarkets, intelligent alarm systems of public security organs to track the suspects, intelligent prisoner alarm systems for prison management, security check in train stations or the airport, and so on. Because of its convenient acquisition, fast recognition and user friendliness, face recognition has become a major and widely applied technology of identity recognition [23–25].

With the gradually increasing scale of videos, the traditional centralized face recognition methods are unable to meet the task demand in time efficiency and expansibility. The distributed face recognition model has the advantages of fast processing speed and scalability, and can be well applied to the fields such as wireless sensor networks [26] and cloud computing [27]. Liu et al. [28] proposed a distributed wireless face recognition system, which integrates multiple kernel discriminant analysis with face recognition, and performs an iterative scheme for kernel parameter optimization. The system can achieve high recognition accuracy, but is only suitable for the recognition of face images. Liang et al. [29] proposed a distributed low-cost IP surveillance system based on a parallel face detection algorithm, which can control multiple cameras to perform multi-angle recognition on a single pedestrian, but cannot perform multi-pedestrian recognition. Hinojos et al. [30] described a distributed computing framework called Blue Hoc that can perform face recognition through a mobile phone, but with low-computational efficiency. Rajeshwari et al. [31] proposed a distributed face recognition system for face recognition in video streams. The system has many clients, each of which can perform face recognition independently. Since the recognition is conducted in images, it is inefficiency and costs lots of computer resources.

Recognizing faces is a relatively simple task for humans, but very difficult for machines. Complicated algorithms and a large number of processing tasks make the computer load extremely large, which not only reduces the CPU utilization rate, but also increases the recognition time. In order to solve these problems, it is necessary to balance the amount of agent-based tasks. The genetic algorithms are widely applied to solving task equilibrium [32–34], which can get the optimal solution through random search. Nonetheless, the traditional algorithms have the disadvantages of inequality and premature convergence. Cheng et al. [35] and Kaliappan et al. [36] proposed to use dynamic genetic algorithms to solve the problem of load balancing clusters in mobile ad hoc networks. The former achieves dynamic optimization by integrating several immigrants, memory, multi-population schemes and their combinations into the standard GA. The latter considers distance and energy parameters to select the optimal cluster head and applies it to new environments. Therefore, the genetic algorithm can be used to solve the task balancing problem of agents in this paper.

Because the number of tasks on the agent side is dynamic, we cannot just balance the tasks when the agent is initialized. Instead, we can carry out dynamic task division on the agent side and predict the amount of agent tasks by using extreme learning machine (ELM) [37–39]. ELM can transform the training of a single hidden layer neural network into solving a linear system by initializing the input

weight and the bias of the hidden layer randomly, and then obtain the determined output weight. Compared with the traditional feedforward neural networks, ELM learns much faster and can obtain the minimum training error and the weight norm. Shamshirband et al. [40] studied the effect of Weibull parameters by applying ELM to the estimated wind speed distribution. Ray et al. [41] presented an accurate hybrid fault location technique for underground power cable. The method uses S-transform to extract useful features and ELM to estimate the fault distance within the selected features. Therefore, a high measurement accuracy and robust fault location technique can be obtained. ELMs were also used for gas flow measurement [42], which can analytically determine the output weights of networks and provide high metering accuracy at fast learning speed as well as require least human intervention. In addition, the ELM-based prediction models achieved good accuracy in other applications, such as price forecast [43,44], haze forecast [45] and photovoltaic power generation forecasting [46]. Motivated by its good prediction ability and robustness, this paper uses ELM to predict the amount of tasks on the agent side, so as to improve the accuracy of dynamic task partitioning.

In this paper, we propose a multi-agent-based distributed face recognition model that can simultaneously process multiple videos containing one or more pedestrians. Load balancing optimization is adopted for solving the problem of unbalanced task assignment between agents. In order to ensure the equalization accuracy, ELM is used to predict the amount of tasks so as to achieve dynamic task division. Experimental results show that the time efficiency and CPU utilization of the model can be improved through performance optimization.

3. The Distributed Face Recognition Model

As shown in Figure 1, the distributed face recognition model we consists of two parts: the server and the agents. Each agent can simultaneously handle multiple videos containing one or more pedestrians. In the video processing step, the agents are mainly responsible for face detection, tracking and feature extraction, and then send obtained data to the server; the server performs face recognition according to the feature information extracted by the agents, and then returns recognition results to the agents.

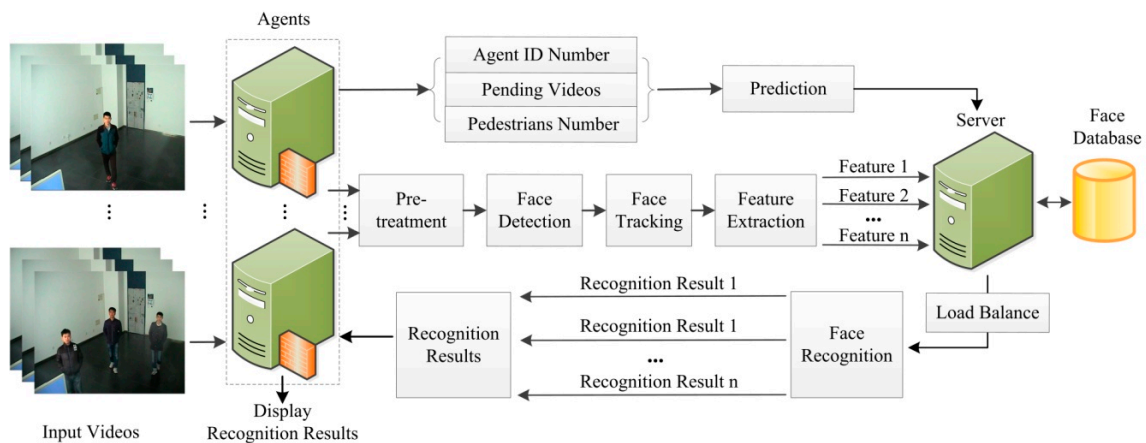


Figure 1. The proposed distributed face recognition model.

Traditional face recognition methods mainly focus on the optimization of the face recognition algorithms on static images, and the optimization of spatial and temporal probability models of real-time videos. Generally, all the processes are concentrated in a single server with finite processing capacity. This kind of method is referred to centralized face recognition, as shown in Figure 2a. In such a situation, server overload will cause with growing tasks, leading to the decline of information processing speed and system delay. Moreover, the security of face recognition cannot be guaranteed because the server, the camera and the dataset are all in the same local network. We compare the centralized system with the distributed system in Table 1.

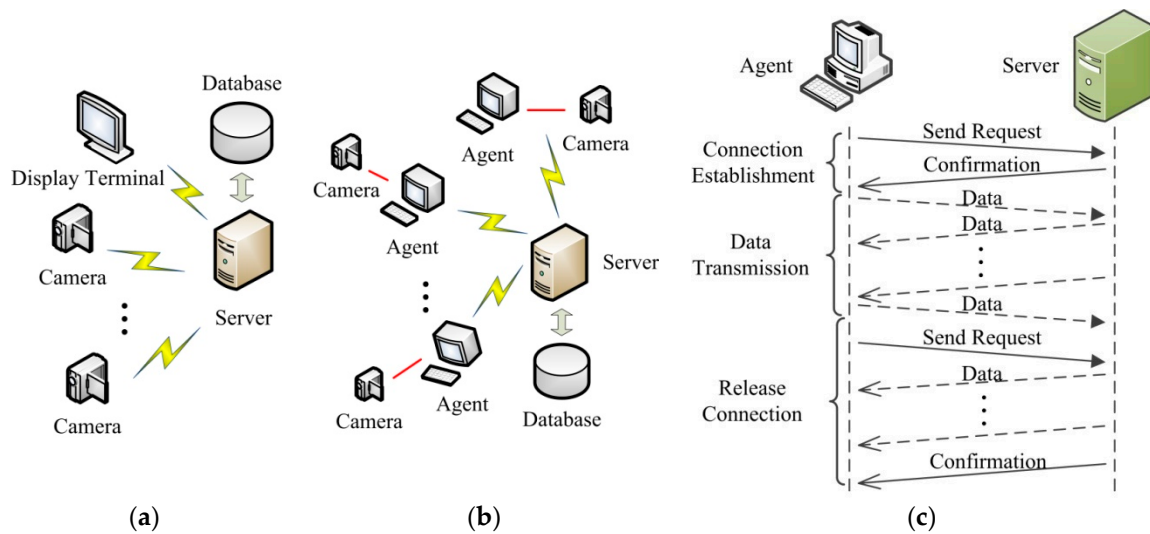


Figure 2. (a) Centralized face recognition; (b) distributed face recognition; (c) server-agent interaction in the distributed face recognition.

Table 1. The comparisons of two systems.

System	Central Server	Node Functions	Performances	Scalability	Fault Tolerance
Centralized system	One central controller	All nodes are passive for task allocation	Small throughput Poor Robustness Globally optimal	Poor	Poor
Distributed system	No controllers	All nodes are autonomous for task allocation	Large throughput Good scalability Locally optimal	Good	Good

Based on the above, we construct a distributed framework to improve the face recognition performance, as shown in Figure 2b. This framework includes two parts: the server and the agents. Figure 2c shows the interaction between the server and the agents. The server and the agents exchange information with each other through the Simple Object Access Protocol (SOAP). This is a simple object access protocol for exchanging structured information in a distributed environment. The process of information exchange is as follows. The agent sends a connection establishment request to the server. Once the server confirms, the connection is established, and the data are transmitted between the agent and the server. When the agent completes the transmission of all the data, it sends a connection release request to the server, and then the server either continues to transfer the data to the agent, or directly confirms the request and releases the connection.

The agent side consists of a video stream forwarding module and a face recognition module. For each agent, a number of face recognition tasks can be dealt with. The video stream forwarding module connects with the network camera to receive data stream. The face recognition module mainly implements some preparatory work for face recognition, including face detection, tracking and feature extraction. Then, the server side performs further face identification using extracted face features sent from the agents. The server saves the results to the dataset and finally determines whether to return the results to the face recognition module according to the type of monitoring authority. At the same time, it supports remote Web Client Access, and the user accessing the system through the browser can watch both real-time video streams and on-demand warning documents and tasks such as the definition and query.

The whole process of the real-time distributed video-based face recognition framework is given in Figure 3, which are: (1) face detection: detecting and marking the faces in video frames; (2) face tracking: regarding the coordinates of detected human faces as the initial state of the tracking target, and then

tracking the faces; (3) feature extraction: extracting the face features like principal components from the entire face or features like eye and lip from local feature points; (4) dimension reduction: reducing the extracted high-dimensional features to low-dimensional ones; and (5) face matching/recognition: comparing face features from the agents with the stored data in the dataset and returning recognition results to the agents. Herein, the first four steps are implemented on the agents, and the time taken for each step is also recorded. The server conducts Step 5 and records the time. The total time to process all the videos can be obtained as follows. Suppose that there are N tasks and M agents. Use $O(N_p)$ to represent the processing time on the agent side and $O(N_r)$ to represent the recognition time on the server side. Since $O(N_r) \ll O(N_p)$, $O(N_r)$ can be ignored. Therefore, the total time for processing all the videos can be represented as $O(N_p/M)$.

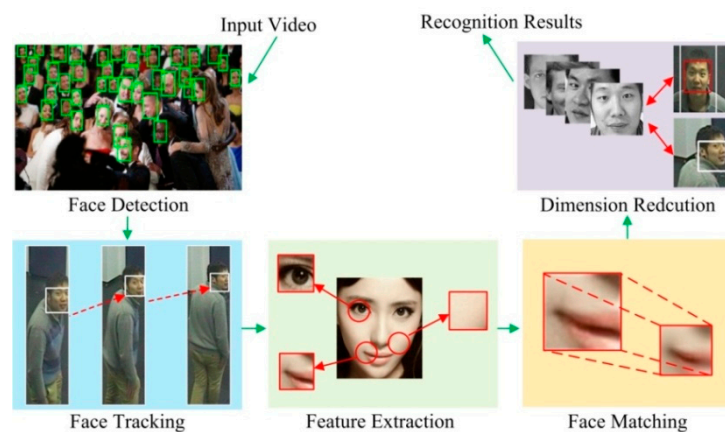


Figure 3. Key steps of the video-based face recognition scheme.

Face matching is conducted on the server. At first, the server sends the number of tasks to the agent. After the server receives the task request and agrees to the request, the agent establishes a connection between the server end and the agent end, and continuously sends the video stream data to the server once the recognition task starts. When the request is for face recognition, the agent detects and tracks pedestrians in video sequences, and then detects faces in the pedestrian bounding box. After the normalization, gray scaling, histogram equalization and dimensionality reduction on these face images, the extracted features are sent to the server and matched with the data previously stored in the dataset. At last, the server returns the recognition results to the agent. When the server sends a request for ending the task to the agent, the agent terminates the current face recognition module; when the server sends the request to disconnect, the agent stops sending video streams and keeps the waiting state. Besides, the distributed framework could enhance the security because the recognition results are only returned to the agent with the right to view the results, and different permissions of agents have different impacts on the face recognition results.

4. Load Balancing Based on an Improved Genetic Algorithm

In a distributed face recognition architecture, there are multiple agents that can handle multiple tasks simultaneously. However, if the tasks for each agent are not balanced, the agent with a large number of tasks will have time delay and sharp increase of CPU utilization, which affects the overall performance of the distributed system. In this paper, an improved genetic algorithm is used for load balancing to ensure the performance.

Assume that k agents are assigned with m videos, each of which involves one or more pedestrians. Here, $A = \{A_1, A_2, \dots, A_k\}$ and $IP = \{IP_1, IP_2, \dots, IP_k\}$ are used to represent agents and IP addresses, respectively, and IP_i is the IP address of the i -th agent A_i ($0 < i \leq k$). The set of videos is $V_i = \{V_{i1}, V_{i2}, \dots, V_{in_i}\}$, n_i ($0 < n_i \leq m$) is the number of videos in the i -th agent, and V_{ij} represents the j -th video in the i -th agent ($0 < j \leq n_i$). $P_{ij} = \{P_{ij}(1), P_{ij}(2), \dots, P_{ij}(n_{ij})\}$, where P_{ij} is the number

of pedestrians in the j -th video. The total number of pedestrians in all the videos processed by the i -th agent is $P_i = \{P_{i1}, P_{i2}, \dots, P_{in_i}\}$. $P_i = \sum_{k=1}^{n_{ik}} P_{ik}(n_{ik})$, P_{ik} is the number of pedestrians in the k -th video. The video processing time of K agents is represented by $T = \{T_1, T_2, \dots, T_K\}$, and T_i is calculated as follows:

$$T_i = \sum_{j=1}^{n_i} t_{ij}, \tag{1}$$

where t_{ij} is the time to process one frame of the j -th video in the i -th agent, and $t_{ij} = t_p + P_{ik} * t_r$. Here, t_p is the processing time of the agent that includes detection, tracking, feature extraction, and transmission time from the agent to the server; t_r is the face recognition time, which is obtained by the server.

Hence, the average time \bar{T}_i to process a frame in a video is calculated as follows:

$$\bar{T}_i = \frac{\max_{1 \leq i \leq K} T_i}{\sum_{i=1}^K \sum_{j=1}^{n_i} n_{f_{ij}}}, \quad (i = 1, 2, \dots; j = 1, 2, \dots, n_i), \tag{2}$$

where n_i is the total number of videos processed by the i -th agent, $n_{f_{ij}}$ is the total number of frames of the j -th video processed by the i -th agent. The average time \bar{t}_p for processing a pedestrian is calculated by:

$$\bar{t}_p = \frac{\max_{1 \leq i \leq K} T_i}{\sum_{i=1}^K \sum_{j=1}^{n_i} P_{ij} * n_{f_{ij}}}. \tag{3}$$

First, we initialize the chromosomes in the genetic algorithm. Each gene node contains three parameters: IP_i , V_{ij} , and P_{ij} . The i -th chromosome is represented by H_i . The chromosome initialization is as follows:

$$H_i = \{(IP_i, V_{i1}, P_{i1}), (IP_i, V_{i2}, P_{i2}), \dots, (IP_i, V_{ij}, P_{ij}), (IP_i, V_{in_i}, P_{in_i})\}. \tag{4}$$

The fitness function F_i of the chromosome is defined by

$$F_i = c / \sum_{j=1}^{n_i} P_{ij} \quad (0 < c < 1). \tag{5}$$

The roulette algorithm is used to select chromosomes to produce the next generation. The probability that an individual is selected is proportional to its fitness function value. The selection formula $P_h(i)$ is given by

$$P_h(i) = F_i / \sum_{i=1}^k F_i. \tag{6}$$

In the genetic process, chromosome crossover is performed by exchanging some genes of two chromosomes to construct new chromosomes. In order to avoid local optimization caused by premature convergence and protect the population diversity, the operation of controlling individual differences between crosses are added to the crossover probability P_c and the mutation probability P_m according to the adaptation value of the chromosome. The formulas for P_c and P_m are as follows:

$$P_c = k_1 \frac{\bar{F}}{\text{MaxF}_1} + k_2 \frac{\bar{F}}{\text{MaxF}_2} \tag{7}$$

$$P_m = k_3 \frac{\bar{F}}{\text{MaxF}_1} + k_4 \frac{\text{MinF}}{\bar{F}}, \tag{8}$$

where MaxF_1 is the largest fitness value in the chromosome, MaxF_2 is the second largest one, MinF is the minimum value, \bar{F} is the mean value, k_1, k_2, k_3 and k_4 are random numbers between zero and one.

The migration is added in order to improve the convergence rate and reduce the gap between the chromosomes. It is implemented by migrating the largest gene in the worst chromosome to the best

one. A stopping criterion chooses a large number of iterations. Also, a threshold is set to obtain the optimal solution. The formula is described as follows:

$$\bar{F}/\text{MaxF} < \varepsilon \ (\varepsilon < 1). \tag{9}$$

By using the improved genetic algorithm to balance tasks with agents, the efficiency and scalability of distributed face recognition can be enhanced.

5. Performance Optimization of Distributed Face Recognition

The improved genetic algorithm can optimize the performance of the distributed face recognition model and balance the amount of task between agents. However, this optimization is only suitable for videos with small changes in number of pedestrians, and cannot dynamically adjust the number of tasks. In order to further optimize load balancing and improve the task allocation accuracy of the server, ELM is used to predict the number of pedestrians.

5.1. Dynamic Prediction Based on Extreme Learning Machine

Extreme Learning Machine (ELM) is based on single hidden layer feedforward neural networks (SLFNs). ELM can randomly initialize input weights and offsets and obtain corresponding output weights, avoiding complicated manual tuning and attaining faster learning speed. It also has the advantages of parallel distributed processing, high fault tolerance and adaptive learning capability.

In an n-L-m single hidden layer feedforward neural structure, given N samples $(x_i, t_i) \in \mathbb{R}^n \times \mathbb{R}^m$ in the training set, the input node $x_i = [x_{i1}, x_{i2}, \dots, x_{in}]^T \in \mathbb{R}^n$, the output node $t_i = [t_{i1}, t_{i2}, \dots, t_{im}]^T \in \mathbb{R}^m$, the single hidden layer neural network is as follows:

$$\sum_{j=1}^L \beta_j f(w_j \cdot x_i + b_j) = y_i \ (i = 1, 2, \dots, n), \tag{10}$$

where the input weight is $w_j = [w_{j1}, w_{j2}, \dots, w_{jn}]^T$, β_j represents the output weight, b_j is the offset of the i-th hidden node, $f(x)$ is the activation function, $w_j \cdot x_i$ is the inner product of w_j and x_i . When the output error of the network is minimal, it can be expressed as $\sum_{j=1}^n \beta_j \|y_i - t_i\| = 0$. Therefore, it must have β_j, w_j and b_j , which make the Equation (8) established.

$$\sum_{j=1}^n \beta_j f(w_j \cdot x_i + b_j) = t_i \ (i = 1, 2, \dots, n). \tag{11}$$

By representing Equation (8) by a matrix, we have

$$H\beta = T. \tag{12}$$

The expressions of H, β , and T are as follows:

$$H = \begin{bmatrix} f(w_1 \cdot x_1 + b_1) & \dots & f(w_L \cdot x_1 + b_L) \\ \dots & \dots & \dots \\ f(w_1 \cdot x_n + b_1) & \dots & f(w_L \cdot x_n + b_L) \end{bmatrix} \tag{13}$$

$$\beta = [\beta_1 \beta_2 \dots \beta_L]^T \tag{14}$$

$$T = [t_1 t_2 \dots t_n]^T. \tag{15}$$

The training process of ELM is equivalent to solving $\hat{\beta}$, which is the least-squares solution of Equation (9).

$$\|H\hat{\beta} - T\| = \min_{\beta} \|H\beta - T\| \tag{16}$$

This is equivalent to minimizing the loss function C, which is given by

$$C = \sum_{i=1}^n \left(\sum_{j=1}^L \beta_j f(w_j \cdot x_i + b_j) - t_i \right)^2 \tag{17}$$

Then, the solution is $\hat{\beta} = H^\dagger T$, where H^\dagger is the Moore-Penrose generalized inverse matrix of the hidden layer matrix H.

Since the number of pedestrians in a video processed by the agent changes dynamically, after processing the video into f frames, we use ELM to predict the number. Here, x_i is the frame number of the video, that is, the agent adds the number of the video frames into the input matrix while processing each video frame. The predicted results are expressed by $y = [y_1, y_2, \dots, y_n]^T$ ($n = 1, 2, \dots, L$). The transfer function used in this paper is a sigmoid (.) function.

$$f(x) = \frac{1}{1 + e^{-x}} \tag{18}$$

We also improve $\hat{\beta}$ by adding a coefficient matrix $A = E + H^\dagger H$, where E is an $L \times L$ square matrix, and the elements in the square matrix are all 1. Least-square solution is $\hat{\beta} = A^{-1} H T$, and the prediction matrix y is given as follows:

$$y = A \hat{\beta} H^{-1} \tag{19}$$

5.2. Dynamic Load Balancing Optimization

In this paper, we use ELM to predict the number of pedestrians. Firstly, the agent records the number of pedestrians in the video and uses the number in the first n frames as a prediction sample. After getting the output weight by ELM, the number of pedestrians in the video is calculated according to Equation (16). Then, the server performs the task balancing operation according to the prediction results sent by the agent.

The \bar{P}_{ij} represents the number of pedestrians predicted by ELM in the j-th video processed by the i-th agent, the average time for processing a video frame is \bar{t}_{ij} , and the number of videos processed by the i-th agent is n_i , the sum of the average time for the i-th agent to process a frame of each video is:

$$T_i = \sum_{j=1}^{n_i} \bar{t}_{ij} \tag{20}$$

The average time spent by K agents to process a video is \bar{T} , which is given by

$$\bar{T} = \frac{1}{k} \sum_{i=1}^k T_i \tag{21}$$

In order to monitor the time variation for each agent processing a video, a fitness function for the i-th agent is defined as:

$$F_i = \frac{1}{c} \sqrt{(\bar{T})^2 - T_i} \quad c \in (1, \infty) \tag{22}$$

If a new pending video is added, the server will not perform the allocation process immediately, but will wait until the new allocation task is executed. According to Equation (19), the server will allocate the newly added video to the agent with the smallest adaptation value.

In order to evaluate the processing time for each video, a corresponding evaluation criterion is required. The processing time of the k-th agent is calculated by

$$T_k = \frac{1}{n_k} \sum_{j=1}^{n_k} T_{vj} \tag{23}$$

where n_k is the number of videos processed by the k -th agent, T_{V_j} is the processing time that the k -th agent processes the j -th video, which is denoted by

$$T_{V_j} = \sum_{i=1}^{m_j} P_i * T_{m_i}. \quad (24)$$

Here, m_j is the largest number of pedestrians appearing in the video, P_i is the probability of i pedestrians appearing in the video, T_{m_i} is the processing time, and P_i satisfies the following requirement:

$$\sum_{i=1}^{m_j} P_i = 1. \quad (25)$$

The input data are converted into a matrix according to $x_{ij} = \text{txt}(i + j - 1)$, where x_{ij} represents the element of the i -th row and j -th column in the input matrix, and $\text{txt}(\cdot)$ is the chaotic data in the txt file.

The main steps of the dynamic load balancing of the distributed face recognition framework based on ELM prediction is described as follows:

1. Initialization. Input the training sample set, randomly generate the weight w_i and input bias b_i , $i = 1, 2, \dots, L$.
2. Calculate the hidden layer output matrix H , the coefficient matrix A and the output layer weight $\hat{\beta}$.
3. Calculate the prediction matrix y according to Equation (19), and send the predicted value to the server.
4. The server initializes the chromosome based on obtained information from the agent.
5. Calculate the fitness value F_i of each domain according to Equation (5), and then perform selection operations according to Equation (6).
6. Calculate the threshold according to Equation (9). Stop if the threshold is satisfied, otherwise continue.
7. Calculated the crossover probability P_c according to Equation (7) to determine whether to perform the cross operation, and then repeat Step 5.
8. Calculate the mutation probability according to Equation (8). If the probability satisfies the mutation probability P_m , perform the mutation operation. Otherwise, perform the migration operation and repeat Step 5.
9. Go to Step 6 to continue.

6. Experimental Results and Analysis

In this paper, the server and the agents run on Intel (R) Core (TM) i7-3770m CPU@3.40GHz, 8GB RAM and 32-bit Windows 7-based computers. Here, we select the ORL face database [47,48] to demonstrate the effectiveness of the proposed distributed face recognition framework. The database has 400 images of 40 distinct subjects, and the image size is 92×112 . In order to identify the problem of poor multi-face recognition in the case of low resolution at a long range, we increase the number of face images in the ORL database to improve the recognition ability of the model. Since the ORL face database is relatively small, we collected 178 images from seven pedestrians and added them into the ORL face database. To further enrich the image training set, we implemented multi-scale transformation of face images in the database. Here, the size of each face image is scaled by the resize function, resulting in 3,398 face images in total after six scaling transformations (23×23 , 25×25 , 46×46 , 50×50 , 75×75 , 92×92). Some example face images collected by us are given in Figure 4.



Figure 4. Our newly collected dataset. There are 3398 images of 47 subjects in the database, where 400 images come from the original ORL database, and 2998 images are expanded by us.

Four videos for testing were collected by an Axis 215 PTZ web camera with a resolution of 704×576 , and the frame rate of the video was five frames per second. The number of pedestrians in a video dynamically changes from one to four. During the test, we can change the amount of task at the agent side by repeatedly loading the testing videos. To facilitate the exploration for the optimal solution in the search space, the value of cross probability is set to be 0.8. By setting different mutation probabilities, the average processing time of one frame and the CPU utilization are given in Table 2.

Table 2. The average time for processing one frame and CPU utilization to the mutation probability.

Mutation Probability	0.01	0.05	0.1	0.2	0.4	0.5
Average time for processing one frame (s)	0.0542	0.0474	0.0458	0.0423	0.0494	0.0512
CPU utilization	0.3391	0.3300	0.3255	0.3125	0.3141	0.3238

As shown in Table 2, when the mutation probability increases from 0.01 to 0.2, the average time for processing a frame and CPU utilization of the server decreases. As the mutation probability continues to increase, the algorithm is prevented from converging, and then the average time for processing a frame and CPU utilization of the server increases. Hence, the mutation probability is set to be 0.2 during the simulations.

Figure 5 shows the iterative curve of the algorithm in load balancing. The iteration process can be terminated when the change of average fitness and maximum fitness tends to be stable. So, According to Equation (9), iteration terminates when ϵ is close to 1. This shows the convergence speed of the genetic algorithm in finding global optimal approximate solutions.

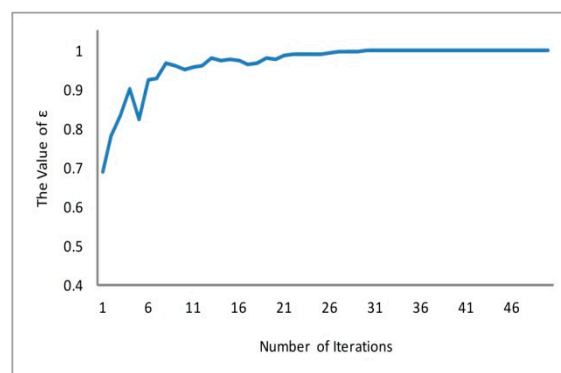


Figure 5. Iterative curve of the algorithm.

Figure 6 shows the recognition results displayed by the agent. For example, marker “1: Soda” shows the order of appearance and the name of the recognized pedestrian.

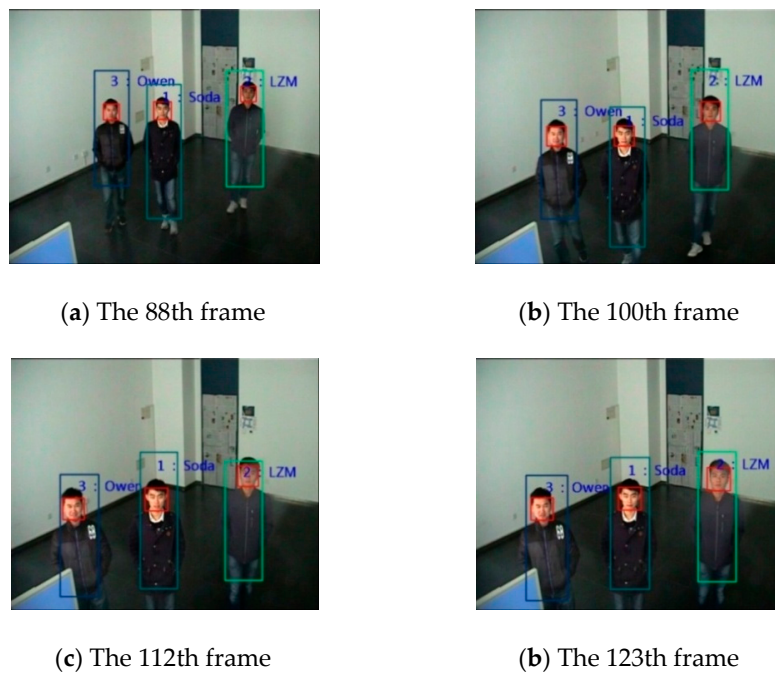


Figure 6. The recognition results.

Here, in order to analyze the relationship between the processing time and the number of pedestrians in the video, we randomly select two agents for experimentation. Each agent handles one video, each of which is with a different number of pedestrians, and one agent processes only one video at a time. The processing time of different operations in each frame are given in Table 3. The average processing time on the agent side includes detection, tracking, feature extraction, and transmission time from the agent to the server.

Table 3. The average processing time on the agent side, the recognition time on the server side, and the transmission time from agents to the server with different number of pedestrians.

Number of Pedestrians	2	3	4	5	6
Average time on the agent side (s)	0.0213	0.0318	0.0466	0.0538	0.0648
Average time on the server side (ms)	0.1492	0.2688	0.3156	0.3674	0.4566
Transmission time from agents to server (s)	0.0089	0.0149	0.0216	0.0301	0.0365

As shown in Table 3, the average processing time for a frame at both server side and agent side increases with more pedestrians. By comparison, the recognition time of the server is much shorter than the processing time of one agent. When dealing with pedestrians in a frame, the agent needs to detect, track, and extract features from a pedestrian, and then send the features to the server for identification. Afterwards, the same operation is performed on the next pedestrian in the frame. The operation loops until the features of all the pedestrians in a frame are extracted and transmitted to the server. The more pedestrians, the more tasks the agent needs to process and transmits to the server, and the more pedestrians the server needs to identify. Therefore, the total time for an agent processing a frame and the recognition time of the server are longer.

6.1. Analysis of Load Balancing Based on Genetic Algorithm

In order to avoid human interference and make the agent’s task amount more objectively, the experiments were simulated by randomly assigning 6, 8, 12, 16, 20, 24, and 28 videos to four agents. The total number of pedestrians in all the videos handled by four agents was 10, 16, 24, 31, 37, 42, and 49. In the experiment, each agent first counts the number of videos it processes and the number

of pedestrians in each video. The statistics are then sent to the server by agents. Finally, the server reassigns the videos to each agent by applying load balancing algorithms. Therefore, the results of the experiment are mainly affected by the equalization algorithm. Load balancing algorithms are the particle swarm optimization (PSO), the improved genetic algorithm (MGA), and the random partition method (RPM). Here, we analyze the performance of the load balancing algorithm by comparing the processing time and CPU utilization. The results are shown in Figure 7.

Figure 7a shows the average time for processing a frame in a video. As the number of pedestrians increase, the average time becomes longer. Generally, after using the MGA for task equalization, the average time is reduced by 20.36% compared to the RPM and 9.71% compared to the PSO. The PSO occasionally experiences instability during task processing, which leads to long average time, and the MGA has advantages in stability.

Figure 7b shows the average time to identify a pedestrian in one video frame. With the number of pedestrian increases, the average time for processing a pedestrian becomes shorter.

Figure 7c shows the comparison of CPU utilization on the server side. The server-side CPU utilization is affected by the number of times the connection function is executed. The agent calls a connection function for each face image. Therefore, the more times the server executes the connection function, the higher its CPU is utilized. By using MGA, the average CPU utilization in the agent side is about 24.85% lower than RPM's and 10.63% lower than PSO's.

Figure 7d shows the comparison of the maximum CPU utilization among the four agents. When the total number of pedestrians in the video reaches to 49, the maximum CPU utilization at the agent side is almost 100% by using the RPM, whereas that using the PSO's is 76.73%. The MGA's is 67.65%, which is obviously lower than the previous two methods.

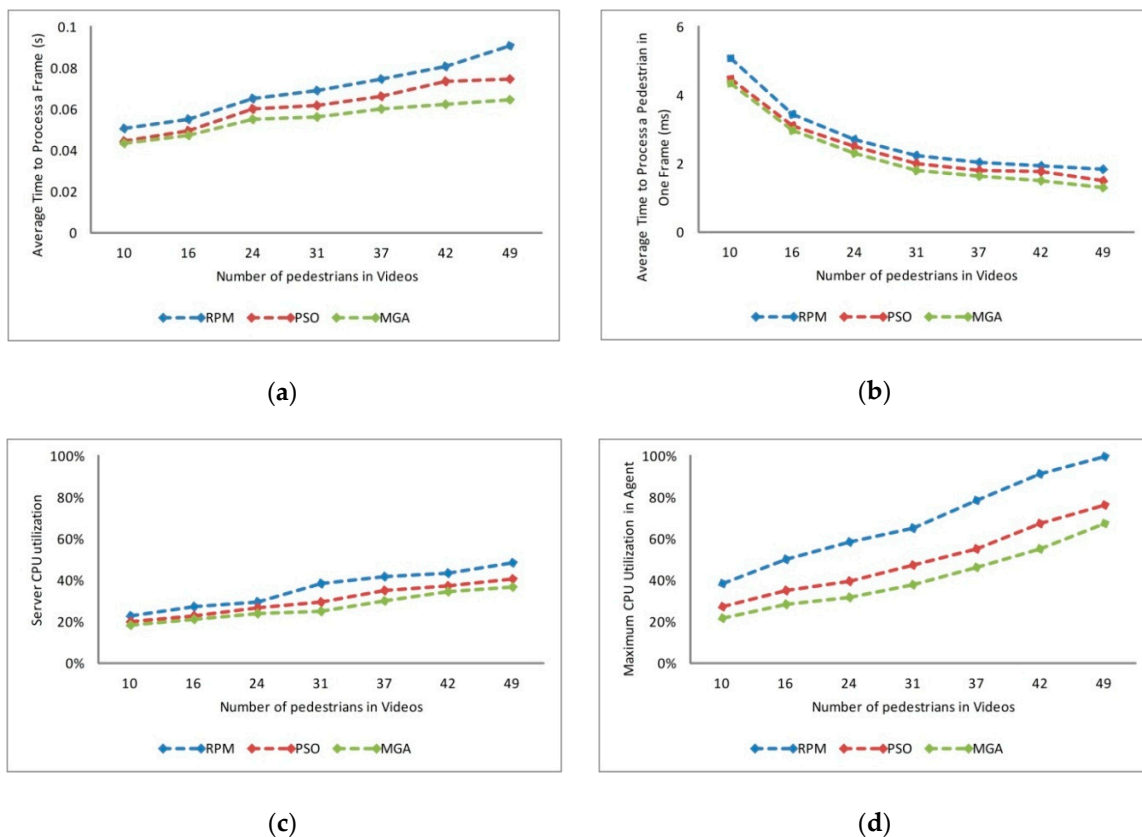


Figure 7. Performance comparison of processing time and CPU utilization for different number of pedestrians in videos. (a) Comparison of the average time for processing a frame; (b) comparison of the average time to identify a pedestrian; (c) comparison of the CPU utilization on the server side; (d) comparison of the maximum CPU utilization among the four agents.

From the above experimental results, we can see that compared to RPM, both PSO and MGA can balance the agent tasks better, and can effectively improve the efficiency of the agent-based distributed face recognition model. Here, RPM randomly distributes videos to each agent. As the number of videos increases, their actual allocation effect is getting closer to assigning an average number of videos to each agent. However, this method does not take into account the number of pedestrians in the video, and thus easily to cause an excessive number of pedestrians to be processed by an agent, resulting in a longer total processing time. PSO is similar to the genetic algorithm. Firstly, a random allocation scheme is determined, and then the optimal task allocation scheme is searched through iteration to obtain a shorter total video processing time. PSO has a fast convergence speed and simple operation, but it is easy to fall into the local optimal solution, and there is no good way to find the best equilibrium solution for the agent as a whole. MGA has a strong global search capability to accurately find the optimum solution from all possible solutions. In order to find the optimum solution quickly, the algorithm guarantees the diversity of the population by accepting the worse chromosomes with a certain probability in iterations. In order not to miss the optimal solution, the threshold function is set as the stopping criterion according to Equation (9). Therefore, the global optimization can be better achieved by MGA for load balancing, so that the total processing time is the shortest.

In the experiment, the total processing time is the maximum processing time of the agent. The larger the number of pedestrians assigned, the larger the task amount the agent needs to handle. Therefore, the total processing time and the processing time of one frame are made longer. The more tasks there are, the more computer resources are consumed, and the utilization of the agent's CPU is also increased. Similarly, as the total number of pedestrian increases, the number of faces that the server needs to identify increases, so the CPU utilization of the server increases. By comparing the experimental results, it is obvious that with more balanced allocation, the proposed MGA obtains better performance than both PSO and RPM.

Here, when there are 16 pedestrians in the video, all three methods have experienced the problem that the processing time is too long, while in another test the discontinuity has disappeared. This may be due to a problem with the network transmission or the server's response causing the task to be handled abnormally. Therefore, some experimental suggestions are provided. It is necessary to keep the network transmission as smooth as possible and close some programs that are not used to release the memory or processor space. In addition, it is necessary to eliminate problems such as bad sectors of the hard disk and excessive CPU hardware temperature.

6.2. Analysis of Prediction Based on ELM

To verify that ELM can be used to predict pedestrians accurately, a set of chaotic sequences is used for testing. Chaotic sequences, usually generated by mapping functions such as cat map and logistic map, are commonly used to test the approximation and prediction ability of an algorithm. They have similar properties as random signals, like arbitrary changes with time and non-periodic. ELM used for prediction is based on the framework of n - L - m , where $n = 12$, $L = 20$, and $m = 1$. The first 990 chaotic data points in the txt file are used as training data. Then, the values of chaotic sequences for testing from the 990th frame to the 1000th frame are predicted. The vertical axis of Figure 8a is the value of a chaotic sequence. As we can see, the predicted value is very close to the actual value, and the mean absolute percentage error is 6.91%, which means that the ELM can predict the change of data well.

In the testing video for pedestrian prediction, the number of pedestrians dynamically changes from zero to four. Figure 8b shows the predicted results of the number of pedestrians in the video. Obviously, using ELM to predict the number of pedestrians obtains a good result.

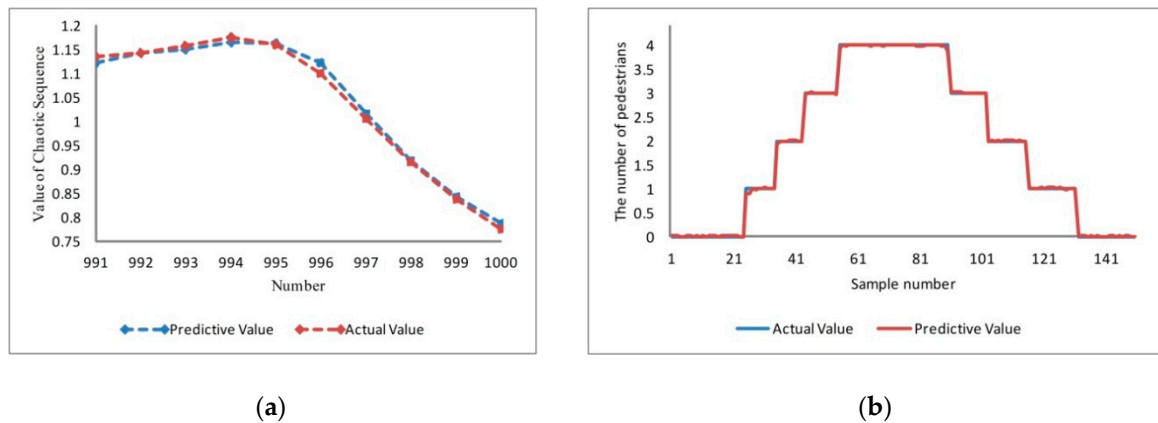


Figure 8. (a) Comparison of the actual value and the predicted value from Number 991 to Number 1000; (b) comparison of actual and predicted numbers of pedestrians in the video.

6.3. Analysis of Dynamic Optimization Based on Prediction

Here, two methods are used to compare the average time for processing each video frame under different conditions. One is random allocation, which distributes videos to the agents by the RPM; the other is dynamic optimization allocation, which assigns videos to the agents using the MGA with prediction.

The experiments are simulated by assigning 4, 6, 8, 10, 12, and 14 videos to two agents, and the results are shown in Figure 9a. In random allocation, the processing time increases as the number of videos increases. In the dynamic optimization allocation, the amount of task at the agent side is always balanced. As the total number of videos increases, the processing time gradually approaches a certain value. When the total number of videos reaches the highest level, the processing time in dynamic optimization is 27.18% lower than in random allocation.

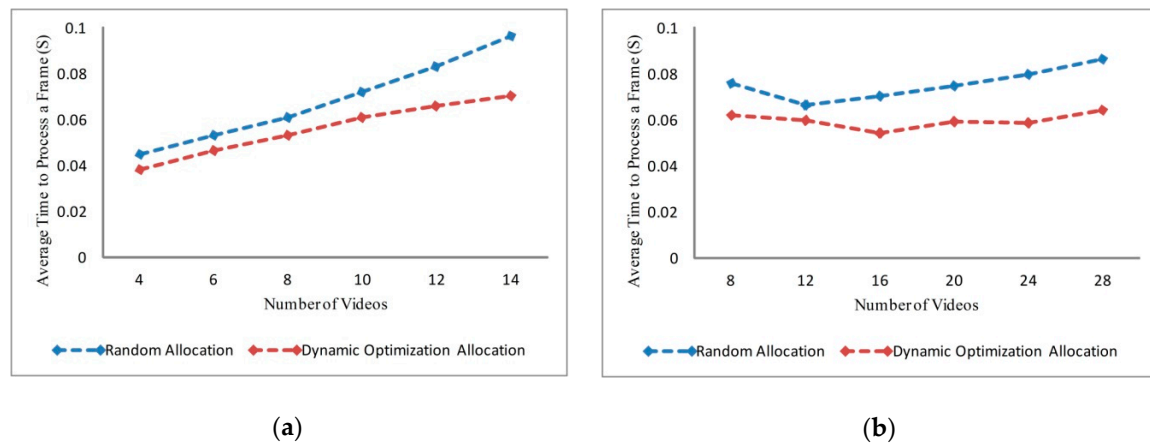


Figure 9. (a) Comparison of the average time for processing each frame when performing random allocation and dynamic optimization allocation; (b) comparison of the average time of processing each frame when performing random allocation and dynamic optimization allocation in the case of four agents.

Also, the experiments are simulated by assigning 8, 12, 16, 18, 20, and 28 videos to four agents, and the results are shown in Figure 9b. In random allocation, the processing time increases rapidly as the number of videos increases. In the dynamic optimization allocation, as the number of videos increases, the processing time grows gently. The processing time in dynamic optimization is 20.44% to 26.03% lower than in random allocation.

7. Conclusions and Future Work

This paper presents a new distributed face recognition framework based on load balancing and dynamic prediction. The framework is composed of one server and multiple agents. The agents perform the front operations of face recognition, such as preprocessing, face detection and feature extraction; while the server mainly performs face recognition. During the recognition process, the amount of task at the agent side is always dynamic. Therefore, it is difficult for the server to allocate tasks balanced, and it also leads to a long processing time and a sharp increase in CPU utilization. To alleviate these problems, we perform load balancing on the agents. In order to make the load balancing accurate and effective, we use extreme learning machine to predict the number of tasks at the agent side. Then the server allocates the task volume of the agent based on the predicted information. The experimental results show that compared with the particle swarm optimization and random partition method, the improved genetic algorithm has better performance in load balancing. Also, the newly proposed optimization method effectively improves the performance and scalability of distributed face recognition.

Author Contributions: All authors of the paper have made significant contributions to this work. F.Z. and J.L. contributed equally to this paper, conceived the idea of work, implemented algorithms, analyzed the experiment data, and wrote the manuscript. W.M. led the project, directed the writing, and revised the paper.

Funding: This work was supported by the National Natural Science Foundation of China under grants 61762061, 61703198, and 61463032, and the Natural Science Foundation of Jiangxi Province, China under grants 20161ACB20004 and 2018ACB21014.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Patel, R.; Yagnik, S.B. A literature survey on face recognition techniques. *Int. J. Comput. Trends Technol.* **2013**, *5*, 189–194.
2. Poli, G.; Saito, J.H.; Mari, J.F.; Zorzan, M.R. Processing Neocognitron of Face Recognition on High Performance Environment Based on GPU with CUDA Architecture. Presented at the 20th International Symposium on Computer Architecture and High Performance Computing, Campo Grande, Brazil, 29 October–1 November 2008.
3. Min, W.D.; Zhang, Y.; Li, J.; Xu, S.P. Recognition of pedestrian activity based on dropped-object detection. *Signal Process.* **2018**, *44*, 238–252. [[CrossRef](#)]
4. Jiang, C.; Su, G.; Liu, X. A distributed parallel system for face recognition. In Proceedings of the Fourth International Conference on Parallel and Distributed Computing, Applications and Technologies, Chengdu, China, 29 August 2003; pp. 797–800.
5. Yan, Y.; Osadciw, L.A. Distributed wireless face recognition system. *Multimedia Content Access: Algorithms and Syst.* **2009**, *6820*, 68200A-1–68200A-12. [[CrossRef](#)]
6. Razzak, M.I.; Khan, M.K.; Alghathbar, K.; Park, J.H. Energy efficient distributed face recognition in wireless sensor network. *Wirel. Pers. Commun.* **2011**, *60*, 571–582. [[CrossRef](#)]
7. Zhang, Z.; Guo, Y.; Song, G.Z. A distributed face recognition framework based on data fusion. *Int. J. Database Theor. Appl.* **2014**, *7*, 87–98. [[CrossRef](#)]
8. Ali, U.; Bilal, M. Video based parallel face recognition using gabor filter on homogeneous distributed systems. Presented at the 2006 IEEE International Conference on Engineering of Intelligent Systems, Islamabad, Pakistan, 22–23 April 2006; pp. 1–5.
9. Chetty, G.; Sharma, D. Distributed face recognition: A multiagent approach. In Proceedings of the International Conference on Knowledge-Based and Intelligent Information and Engineering Systems, Bournemouth, UK, 9–11 October 2006; pp. 1168–1175.
10. Omara, F.A.; Arafa, M.M. Genetic algorithms for task scheduling problem. *J. Parallel Distrib. Comput.* **2010**, *70*, 13–22. [[CrossRef](#)]
11. Lu, H.; Niu, R.; Liu, J.; Zhu, Z. A chaotic non-dominated sorting genetic algorithm for the multi-objective automatic test task scheduling problem. *Appl. Soft Comput.* **2013**, *13*, 2790–2802. [[CrossRef](#)]

12. Zhan, Z.H.; Zhang, G.Y.; Gong, Y.J.; Zhang, J. Load balance aware genetic algorithm for task scheduling in cloud computing. In Proceedings of the Asia-Pacific Conference on Simulated Evolution and Learning, Dunedin, New Zealand, 15–18 December 2014; pp. 644–655.
13. Makasarwala, H.A.; Hazari, P. Using genetic algorithm for load balancing in cloud computing. Presented at the 2016 8th International Conference on Electronics, Computers and Artificial Intelligence, Ploiesti, Romania, 30 June–2 July 2016; pp. 1–6.
14. Siar, H.; Kiani, K.; Chronopoulos, A.T. A combination of game theory and genetic algorithm for load balancing in distributed computer systems. *Int. J. Adv. Intell. Paradigms* **2017**, *9*, 82. [[CrossRef](#)]
15. Balakrishnan, S.M.; Nagarajan, S.K. Performance analysis of ant colony optimization and genetic algorithm for cloud load balancing. *Int. J. Pharm. Technol.* **2016**, *8*, 26092–26100.
16. Min, W.; Fan, M.; Li, J.; Han, Q. Real-time face recognition based on face pre-identification detection and multi-scale classification. *IET Comput. Vision* **2018**. [[CrossRef](#)]
17. Wen, X.; Wen, J. Improved the minimum squared error algorithm for face recognition by integrating original face images and the mirror images. *Optik* **2016**, *127*, 883–889. [[CrossRef](#)]
18. Dang, K.; Sharma, S. Review and comparison of face detection algorithms. Presented at the 2017 7th International Conference on Cloud Computing, Data Science & Engineering-Confluence, Noida, India, 12–13 January 2017.
19. Hajati, F.; Tavakolian, M.; Gheisari, S.; Gao, Y.; Mian, A.S. Dynamic texture comparison using derivative sparse representation: Application to video-based face recognition. *IEEE Trans. Hum.-Mach. Syst.* **2017**, *47*, 970–982. [[CrossRef](#)]
20. Opitz, A.; Kriechbaumzabini, A. Evaluation of face recognition technologies for identity verification in an eGate based on operational data of an airport. Presented at 2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance, Karlsruhe, Germany, 25–28 August 2015.
21. Ujire, N.S.S.; Manipal, S.C.M.; Ujire, A.G.P. Intruder recognition system. *Electr. Electron. Eng.* **2015**, *5*, 6–12. [[CrossRef](#)]
22. Siswanto, A.R.S.; Nugroho, A.S.; Galinium, M. Implementation of face recognition algorithm for biometrics based time attendance system. Presented at the 2014 International Conference on ICT For Smart Society (ICISS), Bandung, Indonesia, 24–25 September 2014; pp. 149–154.
23. Yu, L.; Li, K. Application of Face Recognition Technology in the Exam Identity Authentication System. In Proceedings of the 3rd International Conference on Social Science and Management, Xi'an, China, 8–9 April 2017; pp. 684–688.
24. Okumura, A.; Hoshino, T.; Handa, S.; Yamada, E.; Tabuchi, M. Identity Verification of Ticket Holders at Large-scale Events Using Face Recognition. *J. Inf. Process.* **2017**, *25*, 448–458. [[CrossRef](#)]
25. Li, Q.; Li, T.; Xia, B.; Ni, M.; Liu, X.; Zhou, Q.; Qi, Y. First: Face identity recognition in smart bank. *Int. J. Semant. Comput.* **2016**, *10*, 569–591. [[CrossRef](#)]
26. Gaynor, P.; Coore, D. Distributed face recognition using collaborative judgement aggregation in a swarm of tiny wireless sensor nodes. In Proceedings of the Southeast Con 2015, Fort Lauderdale, FL, USA, 9–12 April 2015; pp. 1–6.
27. Monteiro, C.E.; Trevelin, L.C. Studies of computing techniques for performing face recognition with a focus in the crowds: A distributed architecture based on cloud computing. Presented at 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications, Marrakech, Morocco, 17–20 November 2015; pp. 1–5.
28. Liu, X.Z.; Yang, G. Distributed Face Recognition Using Multiple Kernel Discriminant Analysis in Wireless Sensor Networks. *Int. J. Distrib. Sens. Netw.* **2014**, *2014*, 1–7. [[CrossRef](#)]
29. Liang, W.-Y.; Chiou, C.W.; Chen, Y.-L.; Weng, C.-Y. Design of a Parallel Face Detection Algorithm for Distributed Low Cost IP-based Surveillance Systems. *J. Conver. Inf. Technol.* **2011**, *6*, 306–318. [[CrossRef](#)]
30. Hinojos, G.; Leon, P.L.D. Face recognition using distributed, mobile computing. Presented at the 2014 IEEE International Conference on Acoustic, Speech and Signal Processing, Florence, Italy, 4–9 May 2014; pp. 2179–2183.
31. Rajeshwari, J.; Karibasappa, K. Face recognition in video systems on homogeneous distributed systems. *Int. J. Adv. Comput. Math. Sci.* **2013**, *4*, 143–147.
32. Effatparvar, M.; Garshasbi, M.S. A genetic algorithm for static load balancing in parallel heterogeneous systems. *Procedia-Soc. Behav. Sci.* **2014**, *129*, 358–364. [[CrossRef](#)]

33. Dasgupta, K.; Mandal, B.; Dutta, P.; Mandal, J.K.; Dam, S. A genetic algorithm (ga) based load balancing strategy for cloud computing. *Procedia Technol.* **2013**, *10*, 340–347. [[CrossRef](#)]
34. Nourzadeh, R.; Effatparvar, M. A genetic-fuzzy algorithm for load balancing in multiprocessor systems. *Int. J. Comput. Appl. Technol.* **2014**, *101*, 39–42. [[CrossRef](#)]
35. Cheng, H.; Yang, S.; Cao, J. Dynamic genetic algorithms for the dynamic load balanced clustering problem in mobile ad hoc networks. *Expert Syst. Appl.* **2013**, *40*, 1381–1392. [[CrossRef](#)]
36. Kaliappan, M.; Augustine, S.; Paramasivan, B. Enhancing energy efficiency and load balancing in mobile ad hoc network using dynamic genetic algorithms. *J. Netw. Comput. Appl.* **2016**, *73*, 35–43. [[CrossRef](#)]
37. Huang, G.B.; Zhu, Q.Y.; Siew, C.K. Extreme learning machine: Theory and applications. *Neurocomputing* **2006**, *70*, 489–501. [[CrossRef](#)]
38. Ding, S.; Zhao, H.; Zhang, Y.; Xu, X.; Nie, R. Extreme learning machine: Algorithm, theory and applications. *Artif. Intell. Rev.* **2013**, *44*, 103–115. [[CrossRef](#)]
39. Deng, C.W.; Huang, G.B.; Xu, J.; Tang, J.X. Extreme learning machines: New trends and applications. *Sci. Chin. Inf. Sci.* **2015**, *58*, 1–16. [[CrossRef](#)]
40. Shamsirband, S.; Mohammadi, K.; Tong, C.; Petković, D.; Porcu, E.; Mostafaeipour, A.; Ch, S.; Sedaghat, A. Application of extreme learning machine for estimation of wind speed distribution. *Clim. Dyn.* **2015**, *46*, 2025. [[CrossRef](#)]
41. Ray, P.; Mishra, D. Application of extreme learning machine for underground cable fault location. *Int. Trans. Electr. Energy Syst.* **2016**, *25*, 3227–3247. [[CrossRef](#)]
42. Qin, L.; Hu, L.; Mao, K.; Chen, W.; Fu, X. Application of extreme learning machine to gas flow measurement with multipath acoustic transducers. *Flow Meas. Instrum.* **2016**, *49*, 31–39. [[CrossRef](#)]
43. Xiao, C.; Dong, Z.; Xu, Y.; Meng, K.; Zhou, X.; Zhang, X. Rational and self-adaptive evolutionary extreme learning machine for electricity price forecast. *Memetic Comput.* **2016**, *8*, 223–233. [[CrossRef](#)]
44. Yu, L.; Dai, W.; Tang, L. A novel decomposition ensemble model with extended extreme learning machine for crude oil price forecasting. *Eng. Appl. Artif. Intell.* **2016**, *47*, 110–121. [[CrossRef](#)]
45. Zhu, X.; Ni, Z.; Cheng, M.; Jin, F.; Li, J.; Weckman, G. Selective ensemble based on extreme learning machine and improved discrete artificial fish swarm algorithm for haze forecast. *Appl. Intell.* **2017**, *48*, 1757–1775. [[CrossRef](#)]
46. Tang, P.; Chen, D.; Hou, Y. Entropy method combined with extreme learning machine method for the short-term photovoltaic power generation forecasting. *Chaos Solitons Fractals* **2016**, *89*, 243–248. [[CrossRef](#)]
47. Wang, Z.; Wang, E.; Wang, S.; Ding, Q. Multimodal Biometric System Using Face-Iris Fusion Feature. *J. Comput.* **2011**, *6*, 931–938. [[CrossRef](#)]
48. Wang, K. Implementation of face cartoon maker system based on android. Presented at the 2013 Fourth International Conference on Intelligent Control & Information Processing, Beijing, China, 9–11 June 2013; pp. 193–198.

