Hindawi
Scientific Programming
Volume 2018, Article ID 6509275, 16 pages
https://doi.org/10.1155/2018/6509275

Research Article

HCRCaaS: A Handwritten Character Recognition Container as a Service Based on QoS Guarantee Algorithm

Lei Li,[1] **Xue Gao**,[1,2] **and Lianwen Jin**[1,2]

[1]*School of Electronic and Information Engineering, South China University of Technology, Guangzhou 510641, China*
[2]*SCUT-Zhuihai Institute of Modern Industrial Innovation, South China University of Technology, Zhuhai 519000, China*

Correspondence should be addressed to Lei Li; eelilei@scut.edu.cn

Received 17 April 2018; Revised 23 July 2018; Accepted 31 July 2018; Published 5 September 2018

Academic Editor: Mario Alviano

Handwritten character recognition (HCR) is a mainstream mobile device input method that has attracted significant research interest. Although previous studies have delivered reasonable recognition accuracy, it remains difficult to directly embed the advanced HCR service into mobile device software and obtain excellent but fast results. Cloud computing is a relatively new online computational resource provider which can satisfy the elastic resource requirements of the advanced HCR service with high-recognition accuracy. However, owing to the delay sensitivity of the character recognition service, the performance loss in the traditional cloud virtualization technology (e.g., kernel-based virtual machine (KVM)) may impair the performance. In addition, the improper computational resource scheduling in cloud computing impairs not only the performance but also the resource utilization. Thus, the HCR online service is required to guarantee the performance and improve the resource utilization of the HCR service in cloud computing. To address these problems, in this paper, we propose an HCR container as a service (HCRCaaS) in cloud computing. We address several key contributions: (1) designing an HCR engine on the basis of deep convolution neutral networks as a demo for an advanced HCR engine with better recognition accuracy, (2) providing an isolated lightweight runtime environment for high performance and easy expansion, and (3) designing a greedy resource scheduling algorithm based on the performance evaluation to optimize the resource utilization under a quality of service (QoS) guaranteeing. Experimental results show that our system not only reduces the performance loss compared with traditional cloud computing under the advanced HCR algorithm but also improves the resource utilization appropriately under the QoS guaranteeing. This study also provides a valuable reference for other related studies.

1. Introduction

With the increasing number of mobile devices (e.g., smartphones, tablet computers, and laptops), the input method has become one of the most important applications. Thus, handwritten character recognition (HCR) technology, one of the main input methods for the smartphone, has received considerable research attention and has consequently improved in quality [1, 2]. Nevertheless, some advanced HCR algorithms are difficult to embed in mobile devices because of their resource capacity limitations and the time complexity of the algorithms. Furthermore, embedding bespoke HCR engines into applications is resource and effort intensive, limiting advanced HCR algorithm use and research by individual enterprises. Now, cloud computing [3, 4] provides an innovative networking application model with supercomputing resource capacity. This provides parallel framework to achieve high performance and also supports cross-platform clients [5], freeing clients from the limitations of the computational power and resources in local devices. Furthermore, cloud computing can always provide an elastic distributed resource that can be dynamically allocated to meet varying computing needs. Hence, offloading the HCR task to cloud computing is an effective way to address the conflict between resource capacity limitations and the time complexity of HCR in mobile devices.

However, the task offloading to cloud computing also brings a new challenge: using the pay-per-use [6, 7] model to adjust the resource size according to different workloads, which may cause an impairment in the quality of service (QoS) and resource utilization [8]. Especially for the delay sensitivity of HCR tasks, the complexity distribution architecture in cloud computing (e.g., Hadoop and Spark) is insufficient. In addition, the key technologies of cloud computing (e.g., computational resource virtualization and resource scheduling) are also important elements that impact the performance.

With these in mind, here, we propose an HCR container as a service (HCRCaaS) based on QoS guarantee policy, which not only provides an advanced HCR algorithm (e.g., a deep convolution neutral network (DCNN) [9]) to provide better recognition accuracy but also reduces the performance loss with container technology for the delay-sensitive requirement. To guarantee the QoS as well as high resource utilization, we propose a resource scheduling algorithm based on a performance evaluation under the resource scheduling greedy policy. Our main contributions are as follows:

(i) Designing an HCR engine based on DCNNs as a demo of the advanced HCR algorithm for high-recognition accuracy in cloud computing

(ii) Using containers to deploy the service in order to reduce the performance loss of the virtualization layer and easily expand the resources under different workloads

(iii) Designing a greedy resource scheduling algorithm based on the performance evaluation in order to improve resource utilization under the QoS guaranteeing

The rest of this paper is organized as follows: The related work is described in Section 2. The details of the overall architecture of the system and the communication architecture are presented in Section 3. A description of the HCR engine design and the experiments are presented in Section 4. The resource scheduling method is presented in Section 5. Our experimental design and results compared to traditional systems are presented in Section 6, and we conclude in Section 7.

2. Related Work

Some new systems providing cloud computing online machine learning services have recently been introduced. Triguero et al. [10] developed a MapReduce-based architecture to distribute functions and overcome the challenges of classifying large datasets. Wettinger et al. [11] proposed a new architecture that was different from the systematic classification of DevOps artifacts to model and deploy application topologies. Kaceniauskas et al. [12] developed cloud software services for patient-specific computational analyses of blood flow through the aortic valve on a private university cloud, while Anjum et al. [13] designed a cloud-based video analytics framework for the scalable and robust analysis of video streams based on cloud computing. Verbelen et al. [14] designed and evaluated graph partitioning algorithms that allocated software components to machines in the cloud. Tao et al. [15] proposed an image annotation scheme that transmitted mobile images compressed by Hamming-compressed sensing to the cloud. Tripathy and Mittal [16] designed and combined kernel and possibilistic approaches for image processing based on Hadoop, while Xia et al. [17] introduced a short-term traffic flow forecasting system also based on Hadoop. Xin et al. [18] proposed the novel "Adaptive Distributed Extreme Learning Machine" using MapReduce for distributed computing. Similarly, Zhang et al. [19] proposed a distributed algorithm for training the RBM model based on MapReduce. Thus, the task offloading to a cloud computing platform became a hot research field. To date, these proposed systems have not provided appropriate resource scheduling methods to improve the resource utilization or to guarantee the QoS.

To improve the resource utilization of cloud computing, Xia et al. [20] used a queueing model to evaluate the expected request completion time and rejection probability of a system. Chiang et al. [21] proposed an efficient green control algorithm based on three queueing models. The aim of their work was to find the proper parameters to reduce the power consumption. Du et al. [22] used a queueing model to analyze cloud computing resources. This model optimized the QoS of a video online service in order to reduce the queue length and time delay. To reduce the cost of a hybrid cloud computing platform, Li et al. [23] proposed minimizing the communication costs with an online dynamic provision algorithm based on a queueing model. Khazaei et al. [24] used an M/G/m/m+r queueing model to evaluate the performance of a cloud computing online service. On the basis of this research, they considered that the queueing model presented the relationship between the number of servers and the input buffer size and they could obtain important performance metrics including the task blocking probability and total waiting time incurred during user requests. Bi et al. [25] considered a cloud data center as an $M/M/1/n/\infty$ queueing system. Vakilinia et al. [26] considered that the job arrival rate followed the Poisson process, and the number of jobs in the system could be modeled as an M/G/n/n queueing system. Furthermore, Zhang et al. [27] used an M/G/n queueing model to present the container service process of a Google cluster. Based on the queueing model, the researchers evaluated the average service time. Cao et al. [28] modeled a multicore server processor as a queueing system with multiservers. Based on the model, they proposed an algorithm to optimize the speed of the cores. Feng et al. [29] considered the cloud market as a multi M/M/1 queueing model. Maguluri and Srikant [30] proposed an optimization job-scheduling algorithm to optimize the QoS of a cloud computing service and used a queueing model to present the cloud service process.

Based on these works, the structure of a cloud computing service can be regarded as a queueing model. Although these research works are useful for improving the QoS or resource utilization of cloud computing, there are limitations in their approaches, which ignore the resource overbooking that can impact the performance of services as well as the resource utilization. This creates a large gap between the real execution behavior and the behavior initially expected.

To improve the HCR accuracy, traditional methods including the modified quadratic discriminant function (MQDF) [31] and the graphical lasso quadratic discriminant function (GLQDF) [32] have successfully been used to improve recognition accuracy. Graham used DeepCNet [33] based on DCNNs with good effect at ICDAR 2013 [34], which is the premier competition for document analysis and recognition, and is a Chinese handwritten character recognition competition. Since then, different methods have been proposed to improve character recognition, for example, Murru and Rossini [35] proposed an original algorithm to initialize the weights in a back propagation neural net to improve character recognition training, and Tao et al. [36] proposed a new dimension-reduction method termed sparse discriminative information preservation (SDIP) for Chinese character font recognition. Wang et al. [37] proposed a unified framework to expand short texts based on word embedding clustering and convolutional neural networks (CNNs). Zhong et al. [38] proposed the GoogLeNet models to improve Chinese handwritten character recognition accuracy. The studies [39, 40] also proposed the DCNN-based models to obtain high handwritten character recognition accuracy. These previous studies indicated that DCNN-based models can achieve better recognition accuracy. Based on these works, we also proposed an advanced HCR engine based on DCNN as a demo to present how the advanced HCR cloud service is designed as a real project.

As mentioned above, although the HCR is a traditional online machine learning service, there are some differences. First, because of the requirements of the character input speed, the HCR service is a delay-sensitive application, which requires a simple system architecture. As mentioned above, the HCR service has poor performance under the resource capacity limitation in the mobile devices. How to design an HCR service in cloud computing to provide higher performance is an issue that is worth studying. Second, the HCR service is also a recognition accuracy-sensitive application. Thus, how to design an HCR service in cloud computing to provide better recognition accuracy is a research hotspot. We design a HCR engine based on a DCNN model to achieve better recognition accuracy. Third, owing to the huge number of mobile devices, how to improve the resource utilization in cloud computing also needs to be studied. Based on a queueing model, we design a resource scheduling algorithm under a performance evaluation and the greedy policy. Resource scheduling can guarantee the QoS as well as the improvement of resource utilization. To the best of our knowledge, this is the first work to address these three problems when designing a high-efficiency HCR service in a real cloud computing project.

3. System Design

3.1. Scheme of Handwritten Character Recognition Container as a Service. Cloud computing uses virtualization technology as a resource-sharing method to provide elastic and configurable on-demand resources for the tenants. As mentioned in Section 1, HCR is a common application in mobile devices and is a resource needed to satisfy delay-sensitive requirements. Additionally, the data for handwritten characters are composed by point data, so the size and dimensions of a handwritten character are small [2]. Thus, the delay of data transmission between mobile devices and cloud computing can be overlooked. Therefore, the delay of the recognition process becomes the main element that impacts the performance.

However, VMs based on traditional virtualization technology, for example, KVM, Xen, and Hyper-V, are complete virtualization technologies with a full-guest operating system (OS). They cause such high performance loss that only a few virtual machines (VMs) can be created from one physical machine [41]. According to [42], the time required for creating a VM is 15 s, which makes the resource scheduling lag behind workload changes.

Compared with VMs, containers have different architectures that are useful tools for software deployment and packaging in differently configured environments. The container uses the container engine instead of a Hypervisor layer to isolate the configurable resources environment. Thus, the container can directly run the CPU threading of the physical machine (PM) without a virtualization layer, and it is generally considered that a lightweight virtualization technology is less resource consuming [43, 44]. IBM conducted a performance test of VMs and containers [43], and experiments showed that containers are superior to VMs in terms of CPU, memory, and I/O performance. The startup time of containers is expressed in milliseconds whereas that of VMs is expressed in seconds. Furthermore, containers have been suggested as a solution for more interoperable application packing in the cloud [45]. Hence, containers in cloud computing are more appropriate for HCR service deployment.

Based on the elastic service architecture in cloud computing [46], the HCR container as a service (HCRCaaS) includes these key components:

(i) *The container*: the container is used for resource isolation and lightweight virtualization running environment configuration for the HCR service.

(ii) *The host cluster (resource pool)*: each PM in the cluster can be considered as the container host, which runs daemon threading with the container engine service to provide the container environment. The PM also provides the resources, for example, CPU, memory, and storage. To provide the resource scheduling management, the Python RabbitMQ client library [47] is run to listen to the message from the resource management server for the resource scheduling on demand. According to the message, the PM creates or deletes the containers.

(iii) *Registry container image storage*: the container image storage provides the server storage to store the container image, so that the user can upload and download the container image from the server. It is used to provide the management of the standard HCR container templates for batch elastic expanding. Based on the HCR engine, we design a standard container image based on Ubuntu OS downloaded from the official container hub. Then, to create the HCR container image, the docker-file is used to build the running configuration environment and copy the engine bin file. We set the autoexec of the engine bin file in the final docker-file line.

(iv) *The load balance server*: the load balance server schedules the data from the client devices to the containers to build parallel computing. It is designed to provide the reverse proxy of HCRCaaS by using Nginx 1.9. To provide a transmission control protocol (TCP) load balance, Nginx with the with-stream configuration parameter is set, and the task balance policy is weighted round robin. Since the system containers are the same, the weight values are equal. The server runs daemon threading with the Python RabbitMQ client library to listen to the message from the resource management server for adding or removing the container from the load balance configuration.

(v) *Resource scheduling management*: the resource scheduling manager provides central resource management in HCRCaaS and allocates the container to a container host. Based on the Python framework, three software frameworks are used for resource scheduling management design as follows: (1) The Python Numpy library is used for the resource scheduling algorithm designing, (2) Web Server Gateway Interface (WSGI) provided the tenants with a hypertext transfer protocol (HTTP) interface, and (3) message queueing is designed using the RabbitMQ server software package to send the message for creating or deleting the container from the container hosts.

The architecture of the proposed HCR system based on cloud computing is shown in Figure 1.

3.2. Communication Architecture. As shown in Figure 1, there are two types of communication architecture in the system:

(i) Service communication is responsible for transmitting the handwritten character data to the container and returning the recognition result. The handwritten character data are time-continuous data that are sampled by the client device, for example, a smartphone. The client device stores the character index dictionary, in which the character index is the same as that in the recognition engine. The system receives the data points from the client device and returns the recognition result, which contains the largest probability index of the character classification. The client device uses the largest probability index to provide the candidate character for users.

(ii) Management communication provides the message queueing-RabbitMQ service between the resource scheduling manager and other servers. It is mainly responsible for transmitting the message from the tenants to the servers. Using the creation of a container as an example, the tenant sends a message to the resource management server. Then, the message is transmitted to the message queueing server. The server exchanges the message with the resource scheduling policy to send a command message to the target server for creating the container. After finishing the action, the target server replies with a message to the tenant through the message queueing server. The management communication architecture is shown in Figure 2.

4. Handwritten Character Recognition Engine Design

In recent years, DCNN has achieved excellent results in image classification. It has a better model expression capability. A previous work [48] indicated that each layer in DCNN can be equivalent to a special function component, for example, the convolution layer can be considered as a feature extraction component, the max pooling layer can be considered as a local extremum component, and the activation function can be considered as a nonlinear regression. DCNN can extract high-order features layer by layer, and the final fully connected layer integrates the output features of the final convolution layer or max pooling layer for classification. Thus, DCNN can be considered as a multilayer and nonlinear complex model. Furthermore, owing to the back propagation training method for the entire model, the parameter values of each layer in DCNN can be unitedly adjusted to make the data processing in each layer more coordinated.

To obtain the advanced HCR model, first, we use a comparison experiment to obtain the proper structure of the DCNN-based model. Thus, on the basis the previous work [33, 49], we design three DCNN-based models, which have structures with multiple convolutional layers and fully connected layers, in order to determine the proper structure. Second, for the normalization, we use batch normalization (BN) [50], which can normalize nonlinear inputs and stabilize the distribution by reducing the internal covariate shift to provide the option of using higher learning rates to expedite network convergence. For some deep networks, BN can also effectively solve the problem of vanishing gradients. Third, we use the gradient back propagation training approach. For parameter optimization in the training process, when the training loss decreases, the learning rate should decay to prevent oscillations near the best point. However, setting a small learning rate may cause the low training speed to fall into the local optimal solution. Thus, we set the learning decay rate, which increases with the number of training epochs. The learning rate can be obtained as follows:

$$\mathrm{lr}\left(n_{\mathrm{epoch}}\right) = \mathrm{lr}_{\mathrm{base}} \cdot \mathrm{lr}_{\mathrm{decay}}, \tag{1}$$

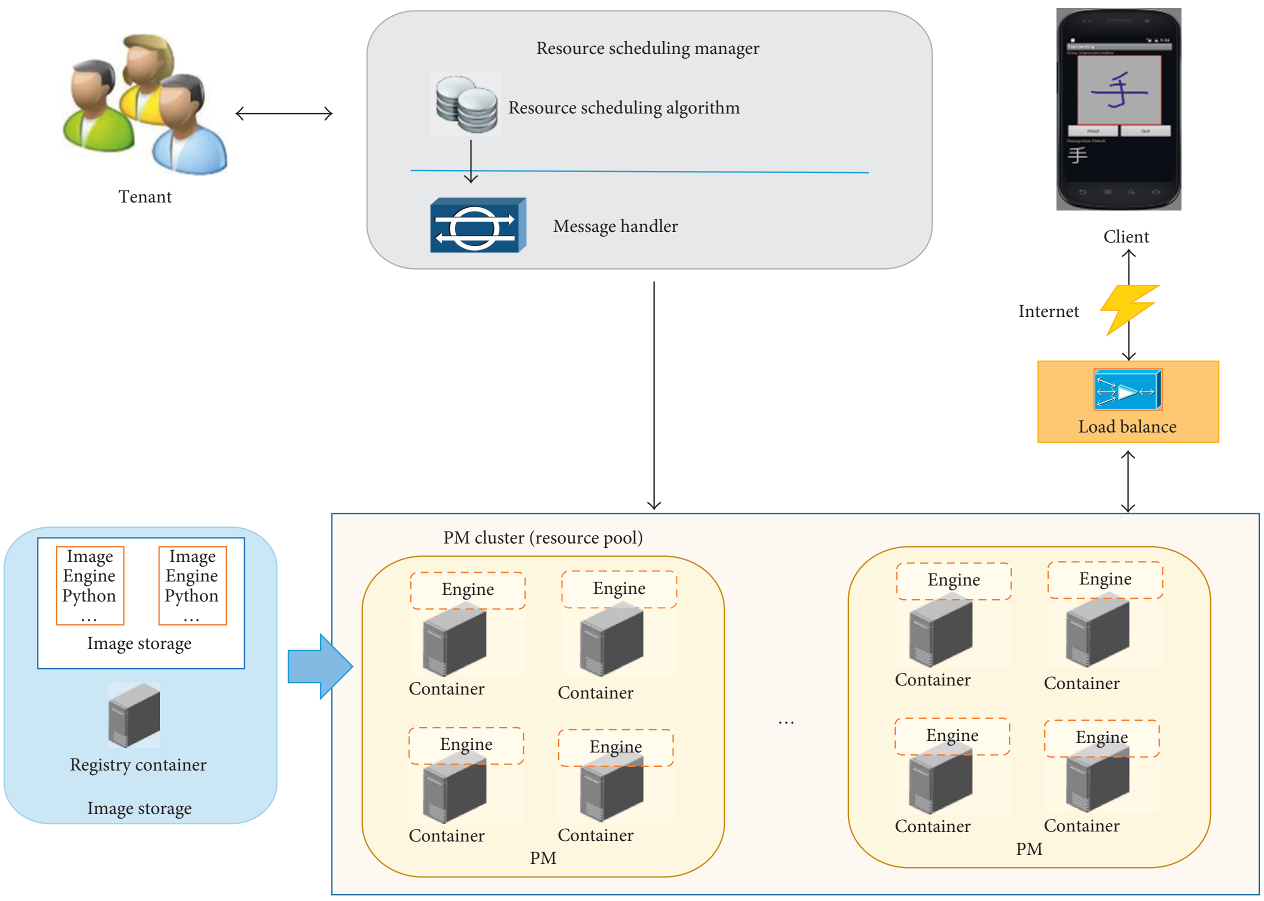

Figure 1: System architecture.

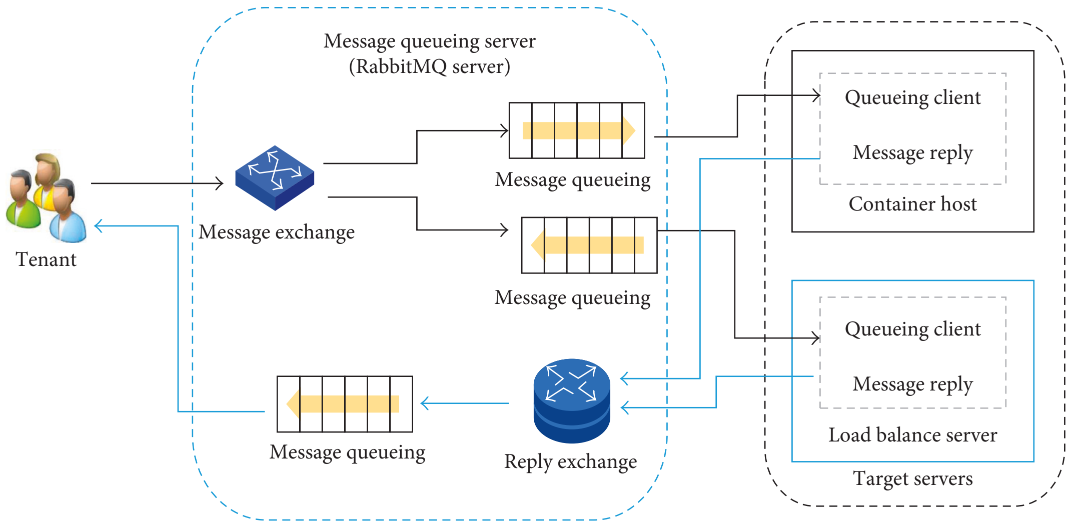

Figure 2: Management communication architecture.

where n_{epoch} is the number of training epochs, lr_{base} is the learning rate basement value, and lr_{decay} is the decay rate.

The three models use the CASIA-HWDB 1.1 dataset [2], which has 300 sets and 1,174,364 handwritten samples, to train on a stand-alone computer. The training dataset includes 240 sets. We use softmax regression as the output. The output is described as follows:

$$h_{(\theta_i)}(x_i) = \frac{\exp\left(\theta_i^T \cdot x_i\right)}{\sum_{j=1}^{k} \exp\left(\theta_j^T \cdot x_j\right)}, \quad j = 1, 2, \ldots, k. \qquad (2)$$

where θ_i^T denotes the weight and bias corresponding to the jth output, and x_i is the ith input feature. The output of the softmax regression layer can be regarded as the confidence probability of the input handwritten characters, which belong to different character classes.

To demonstrate that the DCNN-based methods can achieve better recognition accuracy, we use the ICDAR 2013 competition dataset [34] to conduct the experiment. We not only compared the proposed three models with the traditional methods [2, 51, 52] but also compared them with other DCNN-based methods [34, 38–40]. The comparison results are listed in Table 1. The Number 1–3 models are the proposed models and the others are the comparison models.

From Table 1, the results clearly show that the DCNN-based methods outperform the traditional methods, and the proposed Number 3 model can achieve the best result. Furthermore, to demonstrate that model Number 3 in Table 1 can fit the HCR best, based on model Number 3 in Table 1, we test models with different structures, for example, the structure with one added and removed convolutional layer, structure with one added and removed max pooling layer, and structure with one added and removed fully connected layer.

As shown in Table 2, the recognition accuracies of the different numbers of layers in the DCNN-based models are lower than that of the original DCNN-based model (i.e., Number 3 model in Table 1). This demonstrates that the Number 3 model in Table 1 is the proper structure for 3,755 classes of Chinese handwritten character recognition.

Some previous studies, for example, fisher vector-based method [53, 54], defensive distillation DCNN [55], discriminative spatiality embedded dictionary learning-based representation (DSEDR) [56], data-augmentation [57], robust and sparse fuzzy K-Means with capped l_1-norm (RSFKM) [58], and GA-Bayes [59], have made the prominent achievements in the text classification. In addition, note that the 3,755 categories only contain Chinese handwritten samples. Thus, we compare the three proposed models with these previous methods using the MNIST dataset [60], which contains 10 classes of digit handwritten samples, as shown in Table 3.

In Table 3, the results clearly show that the proposed models can also achieve comparable results that are greater than 99% under the digit handwritten dataset. To test the Latin handwritten recognition, we also use the EMNIST letters dataset [61], which contains 26 balanced classes of Latin handwritten samples, to conduct the experiment. The classified number of output layers of the proposed models is modified to 26. The experiment results are listed in Table 4.

From the results in Table 4, the proposed model also achieves a recognition accuracy of 93% or above in the Latin handwritten dataset. The Number 3 model can achieve the best results.

Although DCNN-based models outperform the traditional methods, compared with the performance of PMs, they have a serious response delay for mobile devices because of the high time complexity. We use the Number 3 model with the best result in Table 1 to compare the average processing time for a single handwritten sample between a mobile device and PM. Loop unrolling is a well-known and efficient strategy to improve speed, especially for large loops. In addition, the BLAS library has been shown to be an efficient way for CPU-based implementation of CNNs. We also use these configuration optimization methods, for example, the BLAS library, loop unrolling, and GPU, to obtain the comparison performance results. The comparison results for a Huawei MATE 7 mobile device and Caffee [63] deep learning framework in PM are listed in Table 5.

From Table 5, HCR service has a serious delay in mobile devices. The motivation of this paper is to design HCR service based on cloud computing. To deploy the most accurate parallel handwritten character recognition service on HCRCaaS, we select the model (Number 3) with the best results in Table 1 as the advance HCR model demo. Based on the model, we used the method with Loop unrolling + BLAS LIB in Table 5 to and combine the model with a TCP interface library to design a feed-forward DCNN-based HCR engine. Since the handwritten data cannot guarantee the same number of data points, they cannot be directly used as input data. Therefore, before the data are inputted to the HCR engine, the system connects the sample point data over time to form a handwritten picture for the input of the DCNN model. The feed-forward DCNN structure is shown in Figure 3.

After compiling the recognition engine using C++ in a Caffe deep learning framework, we deployed it to the container and built the container as a container image for expansion on demand.

5. Resource Scheduling Algorithm

As mentioned above, the containers, which are a lightweight virtualization technology, share their host resources, for example, CPU, in the same host. Owing to the resource capacity limitation of the PM, if the total number of containers is lower than the number of the physical CPU cores, each container can use one isolated core. However, if there are too many containers in a single container host, containers must share these CPU cores, resulting in resource overbooking and degraded performance. However, resource overbooking means that a PM achieves higher resource utilization. Thus, the resource scheduling method needs to achieve a trade-off between resource utilization and performance. Based on the architecture shown in Figure 1, we consider that the HCR service should follow the "first come-first served" (FCFS) principle, which means that each container can be regarded with an M/M/1 queueing model as

$$\lambda_i(t) = \frac{\lambda_{\mathrm{T}}(t)}{k},$$
$$\mu_i(t) = \begin{cases} \mu_{\mathrm{core}}, & \text{if the number of containers} \leq \text{the numer of cores in one PM}, \\ \mu_{\mathrm{share}}, & \text{if the number of containers} > \text{the numer of cores in one PM}, \end{cases} \tag{3}$$

where $\lambda_{\mathrm{T}}(t)$ and $\lambda_i(t)$ are the total arrival intensity and the arrival intensity of the ith container following the Poisson distribution, and the expected number of task arrivals is

Table 1: Structure, recognition accuracy, and hyperparameter settings of three proposed DCNN-based models and other comparison models: each proposed DCNN-based model contains 1 input layer, multiconvolutional layers, multimax pooling layers, and 1 fully connected layer.

Index	Model structure	Accuracy (%)	Hyper parameters
1	96 × 96-80C3 × 3-MP2 × 2-160C2 × 2-MP2 × 2-240C2 × 2-MP2 × 2-320C2 × 2-MP2 × 2-400C2 × 2-MP2 × 2-480C1 × 1-512FC-3755	95.12	minBatch = 128, iterNum = 300,000 dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.1$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 70,000 iterations
2	96 × 96-100C3 × 3-MP2 × 2-200C2 × 2-MP2 × 2-300C2 × 2-MP2 × 2-400C2 × 2-MP2 × 2-500C2 × 2-600C1 × 1-512FC-3755	95.80	minBatch = 128, iterNum = 300,000 dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.1$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 70,000 iterations
3	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-384C3 × 3-MP3 × 3-1024FC-3755	97.30	minBatch = 128, iterNum = 300,000 dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.1$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 70,000 iterations
4	CNN-Fujitsu [34]	94.77	—
5	ART-CNN [34]	95.04	—
6	HCCR-Gradient-GoogLeNet [38]	96.28	—
7	HCCR-Ensemble-GoogLeNet [38]	96.64	—
8	Multi-CNN voting [39]	96.79	—
9	R-CNN-voting [40]	95.55	—
10	ATR-CNN voting [40]	96.06	—
11	MQDF-THU [51]	92.56	—
12	MQDF-HIT [52]	92.61	—
13	DLQDF [2]	92.72	—

Settings for input layer ("Input") are given in rows, which present size of input picture ("size × size"). Settings for each convolutional layer ("Conv") are given in rows, which show number of output feature maps and receptive field size ("numCsize × size"). Settings of max pooling (MP) are specified by type and kernel size ("Mpsize × size"). Settings of fully connected (FC) are specified by dimensionality ("FCdimensionality").

Table 2: Comparison results in different number layers based on Number 3 model.

Index	Model structure	Accuracy (%)
1	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-384C3 × 3-384C3 × 3-MP3 × 3-1024FC-3755 (added one conv layer)	96.88
2	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-384C3 × 3-MP3 × 3-MP3 × 3-1024FC-3755 (added one MP layer)	96.72
3	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-384C3 × 3-MP3 × 3-1024FC-2048FC-3755 (added one FC layer)	96.69
4	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-MP3 × 3-1024FC-3755 (removed one conv layer)	96.74
5	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-384C3 × 3-1024FC-3755 (removed one MP layer)	96.77
6	96 × 96-96C3 × 3-MP3 × 3-128C3 × 3-MP3 × 3-160C3 × 3-MP3 × 3-256C3 × 3-256C3 × 3-MP3 × 3-384C3 × 3-384C3 × 3-MP3 × 3-3755 (removed one FC layer)	96.56

equal to $\lambda_i(t)$during time t. The load balance server uses the round-robin policy to allocate the task to each container; thus, the arrival intensity of each container is $\lambda_i(t) = \lambda_{\mathrm{T}}(t)/k$ when there are k containers in the system. The service rate $\mu_i(t)$, which is the number of tasks be processed in time t, can be divided into two types: (1) when the number of containers is lower than the number of PM CPU cores, $\mu_i(t)$ is equal to the service rate of each CPU core; (2) when the number of containers is higher than the number of CPU cores, more than one container will share the same core service rate, and $\mu_i(t)$ will be lower than that of each core.

The average length of M/M/1 queue can be calculated as

$$L_s(t) = \frac{\lambda_i(t)}{[\mu_i(t) - \lambda_i(t)]}. \tag{4}$$

The expected wait time $W_s(t)$ can easily be found using Little's formula, which is defined as

$$W_s(t) = \frac{L_s(t)}{\lambda_i(t)} = \frac{1}{[\mu_i(t) - \lambda_i(t)]}. \tag{5}$$

Note that $W_s(s)$ is also the average processing time, and we consider $W_s(s)$ as the QoS and performance metric. More important, from Equations (4) and (5), it can be seen that a poor service rate $\mu_i(t)$ will cause a worse service quality according to the M/M/1 queueing model.

According to a different number of containers in the same PM, we test the average processing time of each sample (T_{avg}) to obtain a relative performance function. The system performance for different numbers of containers in the same PM is listed in Table 6.

Table 3: Comparison results in MNIST dataset.

Index	Model structure	Accuracy (%)	Hyper-parameters
1	96×96-80C3×3-MP2×2-160C2×2-MP2×2-240C2×2-MP2×2-320C2×2-MP2×2-400C2×2-MP2×2-480C1×1-512FC-10	99.12	minBatch = 64, iterNum = 9,400 dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.01$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 3 epochs
2	96×96-100C3×3-MP2×2-200C2×2-MP2×2-300C2×2-MP2×2-400C2×2-MP2×2-500C2×2-600C1×1-512FC-10	99.32	minBatch = 64, iterNum = 9,400 dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.01$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 3 epochs
3	96×96-96C3×3-MP3×3-128C3×3-MP3×3-160C3×3-MP3×3-256C3×3-256C3×3-MP3×3-384C3×3-384C3×3-MP3×3-1024FC-10	99.56	minBatch = 64, iterNum = 9,400 dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.01$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 3 epochs
4	Fisher vector-based method	99.66	—
5	Defensive distillation DCNN (transfer temperatures is 20)	99.05	—
6	DSEDR	99.26	—
7	Data-augmentation CNN (5000 training samples per class) (ELASTIC, SMOTE, DBSMOTE)	About (99.60, 99.70, 99.70)	—
8	Data-augmentation CSVM (5000 training samples per class) (ELASTIC, SMOTE, DBSMOTE)	About (89.80, 99.80, 99.80)	—
9	Data-augmentation CLEM (5000 training samples per class) (ELASTIC, SMOTE, DBSMOTE)	About (99.60, 99.80, 99.80)	—
10	RSFKM	59.48	—
11	GA-bayes (2K2K MNIST)	56.83	—

Table 4: Comparison results in EMNIST dataset.

Index	Model structure	Accuracy (%)	Hyper parameters
1	96×96-80C3×3-MP2×2-160C2×2-MP2×2-240C2×2-MP2×2-320C2×2-MP2×2-400C2×2-MP2×2-480C1×1-512FC-26	93.00	minBatch = 64, iterNum = 109,060 Dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.01$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 3 epochs
2	96×96-100C3×3-MP2×2-200C2×2-MP2×2-300C2×2-MP2×2-400C2×2-MP2×2-500C2×2-600C1×1-512FC-26	93.00	minBatch = 64 iterNum = 109,060 Dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.01$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 3 epochs
3	96×96-96C3×3-MP3×3-128C3×3-MP3×3-160C3×3-MP3×3-256C3×3-256C3×3-MP3×3-384C3×3-384C3×3-MP3×3-1024FC-26	95.40	minBatch = 64, iterNum = 109,060 Dropout (fc) = 0.5, $\mathrm{lr}_{\mathrm{base}} = 0.01$ $\mathrm{lr}_{\mathrm{decay}}$: reducing 0.1 every 3 epochs
4	OPIUM-based method [61, 62]	85.27	—

Table 5: Delay comparison results for mobile devices and PMs.

Index	Platform	Method	Average processing time (s)
1	Huawei MATE 7	Direct calculation	8.2
2	Huawei MATE 7	BLAS LIB + GPU	0.1
3	PM	Direct calculation	1.368
4	PM	Loop unrolling + BLAS LIB	0.0211

When the PM is overbooked, the containers in the same PM will share CPU resources. We consider that there is a linear relationship between the service rate of each container μ_{share} and the number of containers k as

$$\mu_{\mathrm{share}} = \frac{1}{\left[T_{\mathrm{avg}} = f(k)\right]}. \tag{6}$$

Based on Table 6, we obtain a linear performance degradation relationship function using order-1 linear differential equations and the least squares method. The fitting effect and the R-squared metric are shown in Figure 4.

The linear performance degradation relationship function is calculated as

$$T_{\mathrm{avg}} = 0.0022\cdot k - 0.001. \tag{7}$$

The R-squared value is 0.998, meaning that Equation (7) describes the relationship between the performance and the number of the containers well. On the basis of this

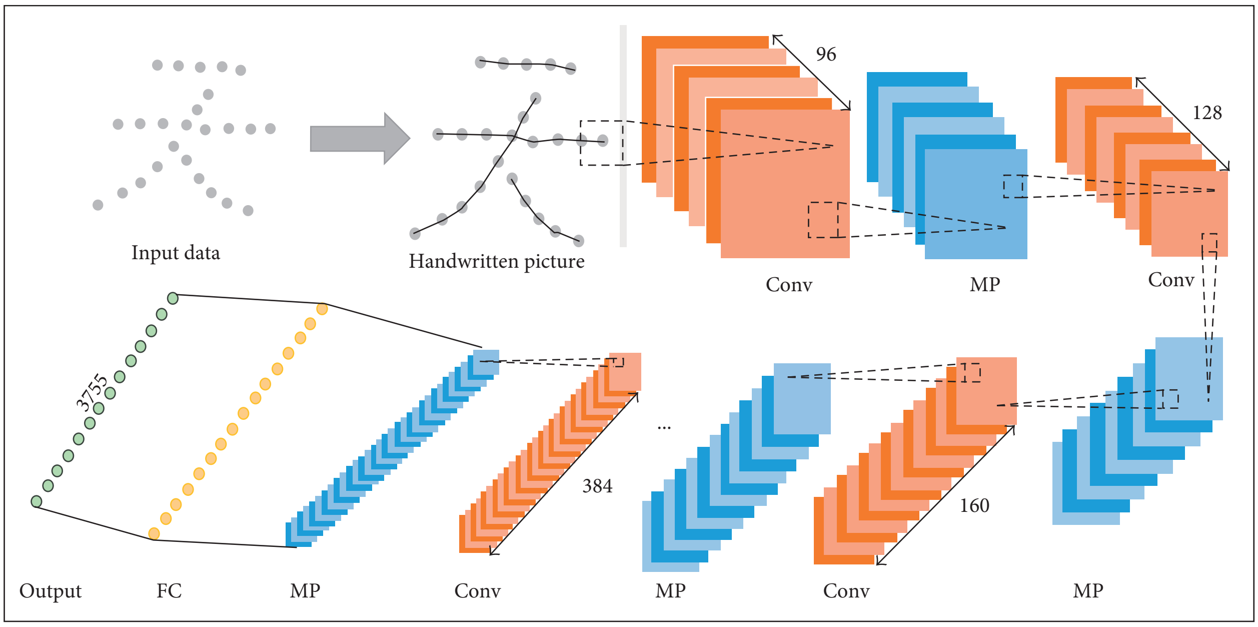

Figure 3: Feed-forward DCNN structure of Number 3 model in Table 1.

knowledge, we propose a greedy performance evaluation (GPE) resource scheduling algorithm to evaluate the performance of each container in the same PM. Taking the resource overbooking into consideration, the algorithm finds the proper PM to place the containers to guarantee the QoS and improve the resource utilization under the greedy policy. The resource scheduling is trigged by the tenant or load balance server under QoS monitoring. The pseudocode for the GPE algorithm is shown in Algorithm 1.

6. Experiments and Analysis

To demonstrate the efficiency, we design a HCRCaaS prototype system. The experimental system is composed of seven nodes including one controller server, one load balance server, one image storage server, and four container hosts. The software configurations of each node are listed in Table 7, and the hardware configurations of the nodes are listed in Table 8.

Furthermore, a stand-alone server is also built with the same hardware and software specifications as the container host for a comparison experiment. We perform a series of experiments to evaluate system performance and resource utilization. We consider that the processing time is the key metric of the HCR service. The longer the processing time, the lower the performance.

6.1. Performance Comparison between Stand-Alone Server, Container, and KVM. Similar to the previous work [64], to demonstrate the efficiency of the HCR delay-sensitive service under the container, the objective of this experiment is to obtain a performance loss (i.e., processing time delay) comparison between KVM and the container. We test the stand-alone server without virtualization technology as the standard of the highest performance. The experiment is conducted using 32,768 samples from CASIA-HWDB [2] for testing the performance of the HCR service in different environments. The processing times of the HCR engine in the container, KVM, and stand-alone server are shown in Figure 5.

It can be seen from Figure 5 that the processing time of the stand-alone server is the shortest, which shows that the stand-alone server performs best. The processing time delay ratio between the container and the stand-alone server, that is, $(718.628 - 673.520)/673.520 \approx 6.69\%$, is smaller than that between KVM and the stand-alone server, that is, $(749.864 - 673.520)/673.520 \approx 11.33\%$. This is because KVM is a large and complex software process. Each KVM has its own virtualization hardware resources including CPU, memory, and NIC. Furthermore, KVM must run its own guest operating system to provide a software environment; therefore, the architecture itself results in performance loss. However, the container is a lightweight virtualization based on a Linux container (LXC) that can directly exploit the container host's hardware resources including its CPU and memory. Furthermore, it does not run the guest operating system to provide the running environment. It can be considered a useful tool to provide different isolated configurations in the stand-alone server. Thus, the container can outperform KVM. Owing to the delay sensitivity of the HCR service, the container can reduce the response delay for providing high resilience and agile computation service quality. This demonstrates that the container can achieve higher performance than KVM in cloud computing.

Table 6: Performance for different numbers of containers.

Number of containers	1	2	4	8	10	12	14	16	32	64	128
T_{avg} (s)	0.021	0.021	0.022	0.022	0.023	0.026	0.027	0.032	0.069	0.136	0.276

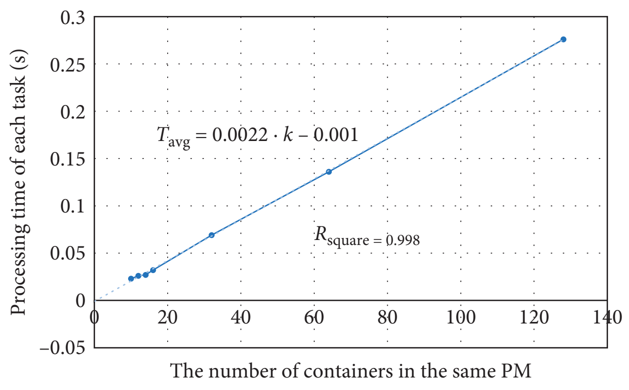

Figure 4: Fitting effect and R-squared metric.

6.2. Performance Comparison between HCRCaaS and Stand-Alone Server. The performance of a stand-alone server and HCRCaaS (i.e., 16 containers in HCRCaaS) is compared under the same hardware specification. The processing time is tested for different numbers of samples from 128 to 131,072. The processing time of HCR is measured from the data arriving at the load balance server to when the test client receives the result. The results of the performance testing are shown in Figure 6 and Table 9.

In Figure 6 and Table 9, the red line denotes the processing time of HCRCaaS, and the blue line denotes the processing time using a stand-alone server. When the number of samples is lower than 128 (a light workload), the HCR engine can completely process these data in a short period. Owing to the load balance server in HCRCaaS, the processing time of HCRCaaS is shorter than that of the stand-alone server. However, when the number of samples increases and the workload becomes large, the processing time increases dramatically in the stand-alone server but not when using HCRCaaS. For example, when the number of samples increases from 128 to 512, the processing time in the stand-alone server increases by 9.589 s, while it increases by 0.5331 s in HCRCaaS. Furthermore, the processing time comparison ration between the stand-alone server and HCRCaaS is $2920.2390/193.1579 \approx 15.12$ for 131,072 samples. The efficiency of HCR can be significantly improved using HCRCaaS.

We also compare the processing time of 32,768 samples in different numbers of containers in one host. The result is shown in Figure 7. The performance increases linearly with increasing numbers of containers when the number of containers is lower than 8. This is because each host contains eight cores, and each container completely occupies one core. By contrast, when the number of containers is higher than that of the cores, overbooking results in more than one container sharing the same core, degraded container performance, and, thus, only a slow increase in cloud performance.

Furthermore, when the number of containers is much higher than that of the cores, there is serious resource overbooking and degradation of overall system performance to guarantee the QoS. Therefore, the processing time increases when the number of containers is greater than 32. Overall, performance can be improved by increasing the number of containers when resource overbooking is not serious, but when the system is heavily overbooked, there is a significant degradation in system performance and hence the need for the GPE algorithm.

6.3. Comparison of Performance with Load Balance Server and without Load Balance Server. The load balance server is a key component of a parallel computing system and is responsible for scheduling the HCR data to different containers. Its performance may also have an impact on the QoS. We test the processing time of six groups of samples from 128 to 131,072 with and without use of the load balance server, as shown in Figure 8 and Table 10. When the number of samples is lower than 512, the processing times with and without the load balance server are similar. Moreover, when the number of samples increases, the processing time with the load balance server also increases slightly. This is because the data size of the HCR is small, as mentioned above, and we consider that the load balance can be regarded as an M/M/∞ queueing model. Thus, the performance loss of the load balance service is minor. The results show that the largest loss of performance is still very small at $(3012.1560 - 2990.2390)/2990.2390 \approx 0.73\%$.

6.4. Performance Comparison between GPE and Greedy Algorithms. To evaluate how the QoS is impacted by the resource scheduling algorithm, we test a large workload using the GPE algorithm and the greedy scheduling algorithm, that is, each host can be in heavy resource overbooking to achieve the highest resource utilization. We set the average maximum waiting time to 0.032 s, the number of users to 16, and the number of samples to 32,768. The results are shown in Figure 9 and Table 11.

It can be seen from Figure 9 and Table 11 that the greedy algorithm does not guarantee the QoS as the number of containers increases. This is because the greedy algorithm only considers the resource utilization, resulting in heavy resource overbooking to place as many containers as

```
The pseudocode of the GPE algorithm
Input:
(1) Set the current arrival number for each container $\lambda_t(t)$
(2) The number of containers in each container host $[k_1, k_2, \ldots, k_n]$, and $n$ is the number of hosts in the system
(3) The number of CPU cores of the container hosts $[C_1, C_2, \ldots, C_n]$
(4) The maximum waiting time $W_{\max}$
(5) The container scheduling trigger command Trigger, where Trigger > 0 if launching the new containers, Trigger = 0 if keeping the
    containers, and Trigger < 0 if deleting the containers
(6) The performance relationship function $T_{\text{avg}} = f(k)$
(7) Output: the updating expected waiting time $W_s(t)$
Step 1:
If trigger < 0
  Sorting the index jth of the container hosts with the number of the containers in on-descender order
  Find the jth host which has the minimum containers
  Set $k_j = k_j - 1$
  Re-calculate $\lambda_i(t) = \lambda_t(t)/(\sum_{j=1}^{n} k_j)$
  Calculate the $W_s$ of the containers in jth container host from (3)–(7)
  If $W_s(t) < W_{\max}$
    Deleting a container in jth container host
    Go to Step 3
  Else
    Set $j = j + 1$
    Repeat Step 1
End If
Step 2:
If trigger > 0
  Sorting the index jth of the container host with the number of the containers in on-ascender order
  Find the jth host which has the maximum containers
  Set $k_j = k_j + 1$
  Re-calculate $\lambda_i(t) = \lambda_t(t)/(\sum_{j=1}^{n} k_j)$
  Calculate the $W_s(t)$ in jth container host from (3)–(7)
  If $W_s(t) < W_{\max}$
    Creating a container in jth container host
    Go to Step 3
  Else
    Set $j = j + 1$
    Repeat Step 2
  End If
End If
Step 3: Updating $[k_1, k_2, \ldots, k_n]$ and adding or deleting the container in the load balance server.
```

Algorithm 1: GPE algorithm.

Table 7: Software configurations.

Node index	Operating system	Software tools
Management server	Ubuntu 14.04	Python 2.7 + RabbitMQ + WSGI 2.7
Load balance server	Ubuntu 14.04	Nginx 1.9.5 + TCP Plug-In
Image storage server	Ubuntu 14.04	Docker 1.9.1
Container host 1	Ubuntu 14.04	Python 2.7 + Docker 1.9.1
Container host 2	Ubuntu 14.04	Python 2.7 + Docker 1.9.1
Container host 3	Ubuntu 14.04	Python 2.7 + Docker 1.9.1
Container host 4	Ubuntu 14.04	Python 2.7 + Docker 1.9.1

possible in the same container host. With the GPE algorithm, when the number of containers is lower than 32, the containers will be created in the same container host if the performance can guarantee the QoS. However, when the number of containers is higher than 32 and the maximum number of containers in the container host is 15 to guarantee the QoS (based on the average maximum waiting time), the GPE algorithm tries to find a proper container host to reduce resource overbooking. The resource scheduling strategy in these situations is equivalent to the average resource scheduling strategy. Therefore, the GPE algorithm can achieve the proper trade-off between the resource allocation balance and utilization.

6.5. Resource Utilization Evaluation and Analysis. To highlight the resource utilization improvement, we compare and analyze the resource utilization of the traditional PM cluster and HCRCaaS. We define the resource utilization comparison ratio between HCRCaaS and the PM cluster as follows:

Table 8: Hardware configurations.

Node index	CPU	Memory (GB)	Hard disk	Network
Control server	Intel i7 2670	16	64 GB SSD	3 Gigabit-NICs bonding
Load balance server	Intel E5 2609	64	64 GB SSD	3 Gigabit-NICs bonding
Image storage server	Intel E5 2609	32	2 TB	1 Gigabit-NICs
Container host 1	Intel E5 2609	96	2 TB	3 Gigabit-NICs bonding
Container host 2	Intel E5 2609	96	2 TB	3 Gigabit-NICs bonding
Container host 3	Intel E5 2609	96	2 TB	3 Gigabit-NICs bonding
Container host 4	Intel E5 2609	96	2 TB	3 Gigabit-NICs bonding

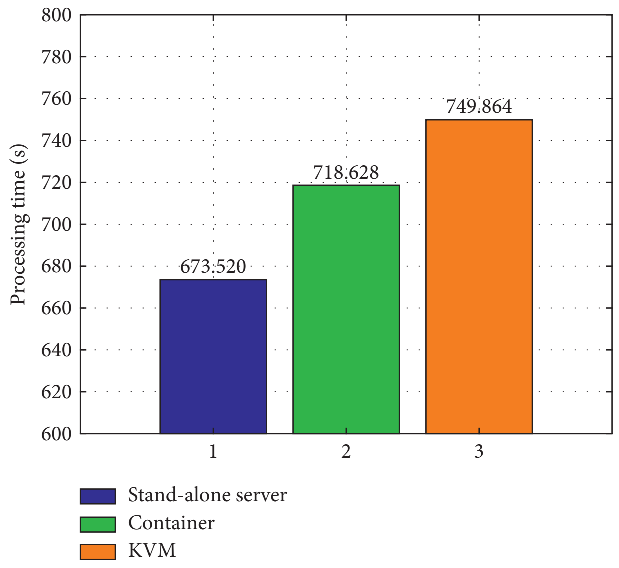

Figure 5: Processing time under different environments.

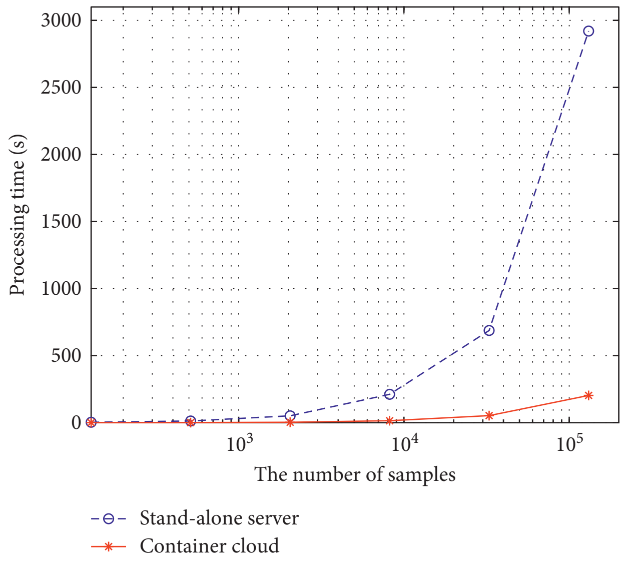

Figure 6: Processing times of stand-alone server and HCRCaaS.

$$C = \frac{\left(\mu_c \cdot \sum_{i=1}^{k} n_i\right)}{\left(k \cdot \mu_{\mathrm{p}}\right)}, \quad (8)$$

where n_i is the number of containers in the jth container host, μ_c is the container resource utilization, μ_{p} is each core utilization of the container host, and k is the number of hosts. When the result of Equation (8) increases, more containers are created in the cluster. If the ratio is higher than 1, the number of containers is higher than that of the PMs. Thus, the larger the result of Equation (8), the larger the resource utilization. According to Section 5, we suppose that $\mu_{\mathrm{p}} \approx \mu_c$. Based on the hardware configuration, we compare the resource utilization comparison ratio between HCRCaaS and the PM cluster under different numbers of containers when the number of PMs is 4 (i.e., $k = 4$). The results are listed in Table 12. It is obvious that the resource utilization improves as the number of system containers increases.

7. Conclusion

In this paper, we designed a handwritten character recognition system based on a container cloud to better utilize handwritten character recognition technology. Using parallel computing and lightweight virtualization technology, we successfully improved the system performance. To overcome problems caused by resource overbooking, we proposed a performance evaluation approach to evaluate the performance of each container as the resource size changed. Using a greedy policy, we designed a GPE algorithm to guarantee the QoS and improve the resource utilization. Our experiments showed that the system efficiency increased significantly with container expansion. This system can easily be extended to other applications, for example, text line recognition, formula recognition, image pattern recognition, and video pattern recognition. The system can also easily be deployed via Amazon, Rackspace, or Windows Azure and private cloud computing platforms.

Future work will aim to improve the system in three respects. First, we will improve the model using more features such as HOG or SIFT to obtain more accurate recognition.

Table 9: Processing times of stand-alone server and HCRCaaS.

	Processing times (s)					
Number of samples	128	512	2048	8192	32768	131072
Stand-alone server	3.1132	12.7022	51.2632	195.5276	687.5201	2920.2390
HCRCaaS	0.1827	0.7158	3.1828	14.7109	52.6269	193.1579

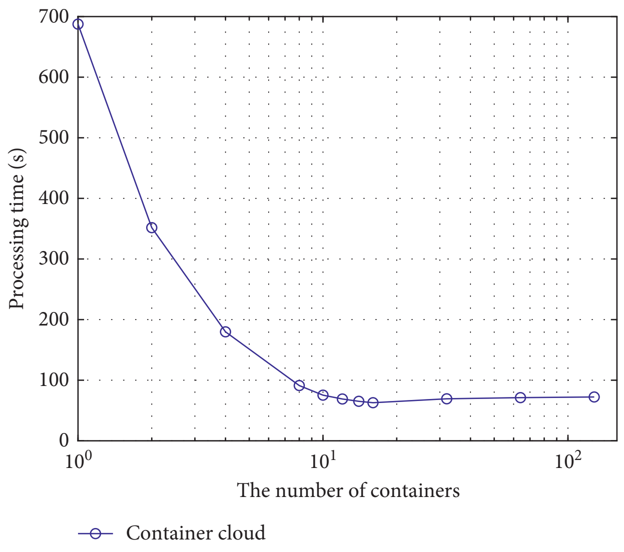

Figure 7: Processing time of HCRCaaS for different numbers of containers.

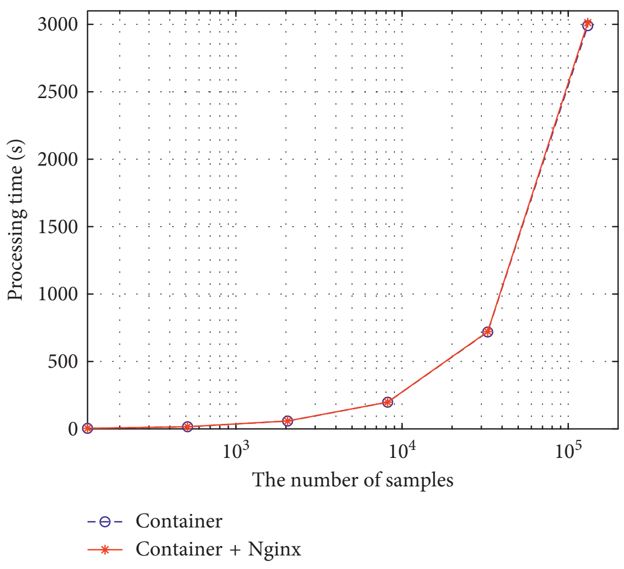

Figure 8: Processing times with and without load balance server.

Table 10: Processing times with and without load balance server.

	Processing times (s)					
Number of samples	128	512	2048	8192	32768	131072
Container	3.9156	15.8902	58.1356	197.8992	718.628	2990.2390
Container + Nginx	4.0723	16.1205	58.6780	198.3204	721.231	3012.1560

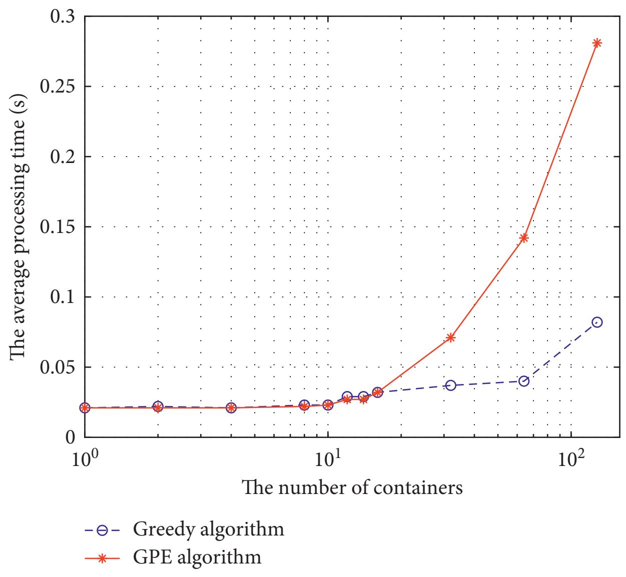

Figure 9: Average processing time under greedy algorithm and GPE algorithm.

Table 11: Average processing time under greedy algorithm and GPE algorithm.

	The average processing times (s)										
Number of containers	1	2	4	8	10	12	14	16	32	64	128
GPE algorithm	0.021	0.022	0.021	0.023	0.023	0.029	0.029	0.032	0.037	0.040	0.082
Greedy algorithm	0.021	0.021	0.021	0.022	0.023	0.027	0.027	0.032	0.071	0.142	0.281

Table 12: Resource utilization comparison ratio for different numbers of containers.

	Resource utilization comparison ratio										
Number of containers	1	2	4	8	10	12	14	16	32	64	128
C	0.25	0.5	1	2	2.5	3	3.5	4	8	16	32

Second, we will evaluate more models such as the long-short term memory model in order to provide more efficient and accurate recognition. Third, we will improve the resource scheduling method to provide an adaptive scalable method in which the number of containers can be automatically adjusted according to the workload. In addition, we will design a workload prediction model for a proactive scheduling resource policy. This will avoid frequent tenant monitoring of the workload and optimize the resource utilization.

Data Availability

Previously reported CASIA datasets are used to support this study and are available at http://www.nlpr.ia.ac.cn/databases/handwriting/home.html. MNIST datasets are used to support this study and are available at http://yann.lecun.com/exdb/mnist/. EMNIST datasets are used to support this study and are available at https://www.nist.gov/itl/iad/image-group/emnist-dataset.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research was supported in part by Guangdong Science Technology Project (Grant nos. 2017A010101027, 2015B010101004, and 2015B010131004), National Nature Science Foundation of China (Grant nos. 61472144, 61673182), GD-NSF (no. 2017A030312006), Science and Technology Program of Guangzhou, China (Grant no. 201707010160), and the National Key Research and Development Program of China (no. 2016YFB1001405).

References

[1] H. Fujisawa, "Forty years of research in character and document recognition—an industrial perspective," *Pattern Recognition*, vol. 41, no. 8, pp. 2435–244, 2008.

[2] C. Liu, F. Yin, D. Wang, and Q.-F. Wang, "Online and offline handwritten Chinese character recognition: Benchmarking on new databases," *Pattern Recognition*, vol. 46, no. 1, pp. 155–162, 2013.

[3] G. Wang and T. E. Ng, "The impact of virtualization on network performance of Amazon EC2 data center," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pp. 1–9, San Diego, CA, USA, March 2010.

[4] R Ranjan, L Zhao, X Wu et al., *Peer-to-Peer Cloud Provisioning: Service Discovery and Load-Balancing, Cloud Computing*, Springer, London, UK, 2010.

[5] S. Abolfazli, Z. Sanaei, A. Gani, F. Xia, and W.-M. Lin, "RMCC: restful mobile cloud computing framework for exploiting adjacent service-based mobile cloudlets," in *Proceedings of IEEE Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 15–18, Singapore, December 2014.

[6] Y. Lee, Y. Kim, H. Han, and S. Kang, "Fine-grained, adaptive resource sharing for real pay-per-use pricing in clouds," in *Proceedings of IEEE Conference on Cloud and Autonomic (ICCAC)*, pp. 236–243, Boston, MA, USA, September 2015.

[7] J. Veen, E. Lazovik, M. Makkes, and R. J. Meijer, "Deployment strategies for distributed applications on cloud computing infrastructures," in *Proceedings of IEEE Conference on Cloud Computing Technology and Science (CloudCom)*, pp. 228–233, Bristol, UK, December 2013.

[8] F. Caglar and A. Gokhale, "iOverbook: intelligent resource-overbooking to support soft real-time applications in the cloud," in *Proceedings of IEEE Conference on Cloud Computing*, pp. 538–545, Anchorage, AK, USA, June 2014.

[9] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[10] I. Triguero, D. Peralta, J. Bacardit, S. García, and F. Herrera, "MRPR: a MapReduce solution for prototype reduction in big data classification," *Neurocomputing*, vol. 150, pp. 331–345, 2015.

[11] J. Wettinger, U. Breitenbücher, O. Kopp, and F. Leymann, "Streamlining DevOps automation for Cloud applications using TOSCA as standardized metamodel," *Future Generation Computer Systems*, vol. 56, pp. 317–332, 2015.

[12] A. Kaceniauskas, R. Pacevic, V. Starikovicius, A. Maknickas, M. Staškūnienė, and G. Davidavičius, "Development of cloud services for patient-specific simulations of blood flows through aortic valves," *Advances in Engineering Software*, vol. 103, pp. 57–64, 2017.

[13] A. Anjum, T. Abdullah, M. Tariq, Y. Baltaci, and N. Antonopoulos, "Video stream analysis in clouds: an object detection and classification framework for high performance video analytics," *IEEE Transactions on Cloud Computing*, 2016.

[14] T. Verbelen, T. Stevens, F. Turck, and B. Dhoedt, "Graph partitioning algorithms for optimizing software deployment in mobile cloud computing," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 451–459, 2013.

[15] D. Tao, L. Jin, W. Liu, and X. Li, "Hessian regularized support vector machines for mobile image annotation on the Cloud," *IEEE Transactions on Multimedia*, vol. 15, no. 4, pp. 833–844, 2013.

[16] B. Tripathy and D. Mittal, "Hadoop based uncertain possibilistic kernelized c-means algorithms for image segmentation and a comparative analysis," *Applied Soft Computing*, vol. 46, pp. 886–923, 2016.

[17] D. Xia, B. Wang, H. Li, Y. Li, and Z. Zhang, "A distributed spatial-temporal weighted model on MapReduce for short-term traffic flow forecasting," *Neurocomputing*, vol. 179, pp. 246–263, 2016.

[18] J. Xin, Z. Wang, L. Qu, G. Yu, and Y. Kang, "A-ELM: adaptive distributed extreme learning machine with MapReduce," *Neurocomputing*, vol. 173, pp. 368–374, 2016.

[19] C. Zhang, C. Chen, D. Chen, and K. T. Ng, "MapReduce based distributed learning algorithm for restricted Boltzmann machine," *Neurocomputing*, vol. 198, pp. 4–11, 2016.

[20] Y. Xia, M. Zhou, X. Luo, Q. Zhu, J. Li, and Y. Huang, "Stochastic modeling and quality evaluation of infrastructure-as-a-service Clouds," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 1, pp. 162–170, 2015.

[21] Y. Chiang, Y. Ouyang, and C. Hsu, "An efficient green control algorithm in cloud computing for cost optimization," *IEEE Transactions on Cloud Computing*, vol. 3, no. 2, pp. 145–156, 2015.

[22] J. Du, C. Jiang, Y. Qian, Z. Han, and Y. Ren, "Resource allocation with video traffic prediction in cloud-based space systems," *IEEE Transactions on Multimedia*, vol. 18, no. 5, pp. 820–830, 2016.

[23] S. Li, Y. Zhou, L. Jiao et al., "Towards Operational Cost Minimization in Hybrid Clouds for Dynamic Resource Provisioning with Delay-aware Optimization," *IEEE Transactions on Services Computing*, vol. 8, no. 3, pp. 398–409, 2015.

[24] H. Khazaei, J. Misic, and V. Misic, "Performance analysis of cloud computing centers using M/G/m/m+r queuing systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 23, no. 5, pp. 936–943, 2012.

[25] J. Bi, H. Yuan, W. Tan, and B. Li, "TRS: temporal request scheduling with bounded delay assurance in a green cloud data center," *Information Sciences*, vol. 360, pp. 57–72, 2016.

[26] S. Vakilinia, M. Ali, and D. Qiu, "Modeling of the resource allocation in cloud computing centers," *Computer Networks*, vol. 91, pp. 453–470, 2015.

[27] Q. Zhang, M. Zhani, R. Boutaba, and J. L. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the Cloud," *IEEE Transactions on Cloud Computing*, vol. 2, no. 1, pp. 14–28, 2014.

[28] J. Cao, K. Li, and I. Stojmenovic, "Optimal power allocation and load distribution for multiple heterogeneous multicore server processors across Clouds and data centers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 45–58, 2014.

[29] Y. Feng, B. Li, and B. Li, "Price competition in an oligopoly market with multiple IaaS Cloud providers," *IEEE Transactions on Computers*, vol. 63, no. 1, pp. 59–73, 2014.

[30] ST. Maguluri and R. Srikant, "Scheduling jobs with unknown duration in clouds," *IEEE Transactions on Networking*, vol. 22, no. 6, pp. 1938–1950, 2014.

[31] Y. Wang, X. Ding, and C. Liu, "MQDF discriminative learning based offline handwritten Chinese character recognition," in *Proceedings of IEEE Conference on Document Analysis and Recognition (ICDAR)*, pp. 18–21, Beijing, China, September 2011.

[32] M. Zhou, X. Zhang, F. Yin, and C.-L. Liu, "Discriminative quadratic feature learning for handwritten Chinese character recognition," *Pattern Recognition*, vol. 49, pp. 7–18, 2016.

[33] B. Graham, "Sparse arrays of signatures for online character recognition," 2013, http://arxiv.org/abs/1308.0371v2.

[34] F. Yin, Q. Wang, X. Zhang, and C.-L. Liu, "ICDAR 2013 Chinese handwriting recognition competition," in *Proceedings of IEEE Conference on Document Analysis and Recognition (ICDAR)*, pp. 1464–1470, Washington, DC, USA, August 2013.

[35] N. Murru and R. Rossini, "A Bayesian approach for initialization of weights in backpropagation neural net with application to character recognition," *Neurocomputing*, vol. 193, no. 12, pp. 92–95, 2016.

[36] D. Tao, L. Jin, S. Zhang, Z. Yang, and Y. Wang, "Sparse discriminative information preservation for Chinese character font categorization," *Neurocomputing*, vol. 129, pp. 159–167, 2014.

[37] P. Wang, B. Xu, J. Xu, G. Tian, C.-L. Liu, and H. Hao, "Semantic expansion using word embedding clustering and

convolutional neural network for improving short text classification," *Neurocomputing*, vol. 174, pp. 806–814, 2016.

[38] Z. Zhong, L. Jin, and Z. Xie, "High performance offline handwritten Chinese character recognition using GoogLeNet and directional feature maps," in *Proceedings of IEEE Conference on Document Analysis and Recognition (ICDAR)*, pp. 846–850, Tunis, Tunisia, 2015.

[39] L. Chen, S. Wang, W. Fan, J. Sun, and S. Naoi, "Beyond human recognition: a CNN-based framework for handwritten character recognition," in *Proceedings of 3rd IAPR Asian Conference on Pattern Recognition*, pp. 695–699, Kuala Lumpur, Malaysia, November 2015.

[40] C. Wu, W. Fan W, Y. He, J. Sun, and S. Naoi, "Handwritten character recognition by alternately trained relaxation convolutional neural network," in *Proceedings of the 14th International Conference on Frontiers in Handwriting Recognition*, pp. 291–296, Crete, Greece, Sept 2014.

[41] W. Lloyd, S. Pallickara, O. David, J. Lyon, M. Arabi, and K. Rojas, "Performance implications of multi-tier application deployments on infrastructure-as-a-service clouds: towards performance modeling," *Future Generation Computer Systems*, vol. 29, no. 5, pp. 1254–1264, 2015.

[42] F. Hermenier, X Lorca, J. Menaud, G. Muller, and J. Lawall, "Entropy: a consolidation manager for clusters," in *Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments ACM*, pp. 41–50, Washington, DC, USA, 2009.

[43] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers," in *Proceedings of IEEE International Symposium on performance analysis of systems and software (ISPASS)*, pp. 171-172, Philadelphia, PA, USA, March 2015.

[44] K. Kaur, T. Dhand, N. Kumar, and S. Zeadally, "Container-as-a-service at the edge: trade-off between energy efficiency and service availability at fog nano data centers," *IEEE Wireless Communications*, vol. 24, no. 3, pp. 48–56, 2017.

[45] L. Affetti, G. Bresciani, and S. Guinea, "aDock: a cloud infrastructure experimentation environment based on open stack and docker," in *Proceedings of IEEE Conference on Cloud Computing*, pp. 203–210, New York, NY, USA, June 2015.

[46] R. Calheiros, E. Masoumi, R. Ranjan, and R. Buyya, "Workload prediction using ARIMA model and its impact on cloud applications' QoS," *IEEE Transactions on Cloud Computing*, vol. 3, no. 4, pp. 449–458, 2015.

[47] A. Videla and J. Williams, *RabbitMQ in Action: Distributed Messaging for Everyone*, Manning Publications Company, Greenwich, London, UK, 2011.

[48] B. Cai, X. Xu, K. Jia, C. Qing, and D. Tao, "DehazeNet: an end-to-end system for single image haze removal," *IEEE Transactions on Image Processing*, vol. 25, no. 11, pp. 5187–5198, 2016.

[49] X. Xiao, L. Jin, Y. Yang, W. Yang, J. Sun, and T. Chang, "Building fast and compact convolutional neural networks for offline handwritten Chinese character recognition," *Pattern Recognition*, vol. 72, pp. 72–81, 2017.

[50] S. Ioffe and C. Szegedy, "Batch normalization: accelerating deep network training by reducing internal covariate shift," in *Proceedings of International Conference on Machine Learning (ICML)*, pp. 448–456, Lille, France, July 2015.

[51] Q. Fu, Q. Ding, T. Liu, Y. Jiang, and Z. Ren, "A novel segmentation and recognition algorithm for Chinese handwritten address character strings," in *Proceedings of Conference on Pattern Recognition*, pp. 974–977, Hong Kong, China, August 2006.

[52] T. Su, "Chinese handwriting recognition: an algorithmic perspective," in *Springer Briefs in Electrical and Computer Engineering*, Springer, Berlin, Germany, 2013.

[53] C. Shi, Y. Wang, F. Jia, K. He, C. Wang, and B. Xiao, "Fisher vector for scene character recognition: a comprehensive evaluation," *Pattern Recognition*, vol. 72, pp. 1–14, 2017.

[54] Y. Wang, C. Shi, C. Wang, B. Xiao, and C. Qi, "Multi-order co-occurrence activations encoded with fisher vector for scene character recognition," *Pattern Recognition Letters*, vol. 97, pp. 69–76, 2017.

[55] N. Papernot, P. McDaniel, X. Wu, S. Jha, and A. Swami, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of IEEE Symposium on Security and Privacy (SP)*, pp. 582–597, San Jose, CA, USA, May 2016.

[56] C. Shi, S. Gao, M. Liu et al., "Stroke detector and structure based models for character recognition: a comparative study," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 4952–4964, 2015.

[57] S. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, "Understanding data augmentation for classification: when to warp?," in *Proceedings of International Conference on Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1–6, Gold Coast, QLD, Australia, November 2016.

[58] J. Xu, J. Han, K. Xiong et al., "Robust and sparse fuzzy k-means clustering," in *Proceedings of International Joint Conference on Artificial Intelligence(IJCAI)*, pp. 2224–2230, New York, NY, USA, July 2016.

[59] T. Vo-Van, H. Che-Ngoc, and T. Nguyen-Trang, "Textural features selection for image classification by Bayesian method," in *Proceedings of International Conference on Natural Computation, Fuzzy Systems and Knowledge Discovery (ICNC-FSKD)*, pp. 731–737, Guilin, China, July 2017.

[60] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[61] G. Cohen, S. Afshar, J. Tapson et al., "EMNIST: an extension of MNIST to handwritten letters," http://arxiv.org/abs/1702.05373.

[62] A. Schaik and J. Tapson, "Online and adaptive pseudoinverse solutions for ELM weights," *Neurocomputing*, vol. 149, pp. 233–238, 2015.

[63] Y. Jia, E. Shelhamer, J. Donahue et al., "Caffe: convolutional architecture for fast feature embedding," in *Proceedings of International Conference on Multimedia (ICM)*, pp. 675–678, Orlando, FL, USA, August 2014.

[64] L. Yin, J. Luo, and H. Luo, "Tasks scheduling and resource allocation in fog computing based on containers for smart manufacture," *IEEE Transactions on Industrial Informatics*, 2018.