Computer Communications 163 (2020) 186–194

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

A deep neural network compression algorithm based on knowledge transfer for edge devices

Yanming Chen [a], Chao Li [b,*], Luqi Gong [b], Xiang Wen [a], Yiwen Zhang [a], Weisong Shi [c]

[a] School of Computer Science, Anhui University, Hefei, China
[b] Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[c] Department of Computer Science, Wayne State University, Detroit, MI, USA

ARTICLE INFO

Keywords:
Edge device
Deep learning
Neural network compression
Knowledge transfer

ABSTRACT

The computation and storage capacity of the edge device are limited, which seriously restrict the application of deep neural network in the device. Toward to the intelligent application of the edge device, we introduce the deep neural network compression algorithm based on knowledge transfer, a three-stage pipeline: lightweight, multi-level knowledge transfer and pruning that reduce the network depth, parameter and operation complexity of the deep learning neural networks. We lighten the neural networks by using a global average pooling layer instead of a fully connected layer and replacing a standard convolution with separable convolutions. Next, the multi-level knowledge transfer minimizes the difference between the output of the "student network" and the "teacher network" in the middle and logits layer, increasing the supervised information when training the "student network". Lastly, we prune the network by cutting off the unimportant convolution kernels with a global iterative pruning strategy. The experiment results show that the proposed method improve the efficiency up to 30% than the knowledge distillation method in reducing the loss of classification performance. Benchmarked on GPU (Graphics Processing Unit) server, Raspberry Pi 3 and Cambricon-1A, the parameters of the compressed network after using our knowledge transfer and pruning method have achieved more than 49.5 times compression and the time efficiency of a single feedforward operation has been improved more than 3.2 times.

1. Introduction

With the advancement of Edge Computing and Internet of Things (IoT) technologies, edge devices are becoming more and more intelligent [1–4]. In order to make these edge devices more intelligent, it is necessary to consider deep learning [5], which has achieved breakthrough achievements in many fields such as computer vision [6], and natural language processing [7]. In recent years, more and more deep neural network models have been proposed, such as AlexNet [8], VGGNet [9], GoogLeNet [10] and ResNet [11], which are becoming more and more accurate in target recognition and getting deeper and deeper. The deep neural network has the characteristics of deep layers and large parameters, which will bring huge storage cost and computational overhead. However, the computing power and storage capacity of the end devices are limited, which seriously restricts the application of deep neural networks to the edge devices. Table 1 shows the depth, parameters and computation amount of some classical neural networks. We can see that the computation and storage cost of deep neural networks are very expensive. There are some approaches to relieve this problem, such as, Zhang et al. proposed a new deep learning-based model, called LDCF, which can effectively learn the high-dimensional and nonlinear relationships between users and services [12], Xu et al. solved the collaborative quantification and placement of edge servers for IoV service offloading with a novel approach combining clustering algorithms and genetic algorithm (GA), which optimized the population initialization of GA and reduced the chance of being trapped in local optimum [3]. In addition, Han et al. [13] show that deep neural networks running on edge devices will increase memory access and consume a lot of battery power. Cloud computing can run deep neural networks at the cloud and provide cloud services [14,15] for the edge devices. For example, Qi et al. utilized Locality-Sensitive Hashing technique to protect the sensitive information of users involved in cross-platform big data integration, which provides an effective and promising way for securing user privacy in various edge-based business applications or systems [4]. But these intelligent applications may encounter high network latency, and high power consumption.

Since the deep neural network cannot run directly on the edge devices and the cloud computing service is not suitable for edge devices,

* Corresponding author.
E-mail addresses: cym@ahu.edu.cn (Y. Chen), lichao@ict.ac.cn (C. Li), zhangyiwen@ahu.edu.cn (Y. Zhang), weisong@wayne.edu (W. Shi).
URLs: http://iacs.ahu.edu.cn (Y. Chen), http://www.ict.ac.cn (C. Li), http://weisongshi.org (W. Shi).

https://doi.org/10.1016/j.comcom.2020.09.016
Received 29 May 2020; Received in revised form 13 August 2020; Accepted 22 September 2020
Available online 28 September 2020

Table 1
Deep neural network parameter.

DNN	Depth	Size (MB)	Computation times (Millions)	Parameters (Millions)
AlexNet	8	200	720	60
VGG-16	16	550	15 300	138
GoogLeNet	22	50	1550	6.8
ResNet	101	170	11 300	42

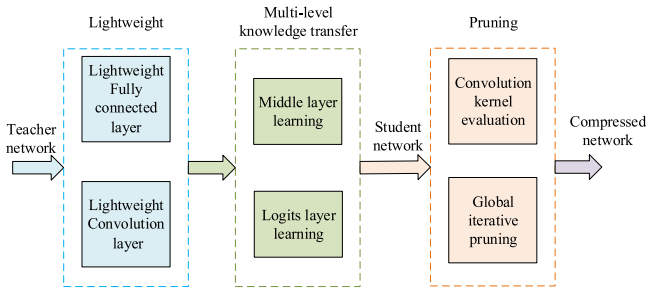

Fig. 1. The three stage compression pipeline of the deep neural network compression algorithm based on knowledge transfer.

we consider compressing the deep neural network and running it on the edge devices. Deep neural network compression can effectively reduce the parameter amount and reduce its computational and storage overhead while the performance of the network model is slightly reduced [16]. The knowledge transfer method is a mainstream approach compressing the deep neural network and it can realize neural network compression without additional runtime library or special hardware support [17]. But there are three problems in existing knowledge transfer methods: the parameters of the designed neural network model are still large; the accuracy of the compressed network model is still far behind of other compression methods; there is still redundant parameters in the compressed network model. In order to solve the above three problems, we propose the deep neural network compression algorithm based on knowledge transfer with three stages: lightweight, multi-level knowledge transfer and pruning. The pipeline of our method is shown in Fig. 1. Lightweight designs a compact "student network" structure that uses a global average pooling layer instead of a fully connected layer and replaces a standard convolution with separable convolutions. Multi-level knowledge transfer learns network changes in the middle layers and adds the soft target with the random factors to the logits layer to increase the supervised information in the process of training and improve the classification accuracy of the "student network" model trained, reduce the loss of classification performance due to compression of neural networks. Through experiments, the proposed method improve the accuracy up to 30% than the knowledge distillation method in reducing the loss of classification performance. In the pruning stage, the "student network" model compressed by the lightweight and multi-level knowledge transfer method is used as a pruning model and the data-independent convolution kernel evaluation method is used to evaluate the importance of each convolution kernel in the pruning model, using a channel-based pruning. The pruning method cuts off the unimportant convolution kernels in the pruning model and uses a global iterative pruning strategy. Experiments show that this pruning method can further reduce 30% of network parameters and shorten the time of a single feed forward execution time of 25%.

The **main contributions** of this paper are proposing the deep neural network compression algorithm based on knowledge transfer with three stages: lightweight, multi-level knowledge transfer and pruning that reduce the network depth, parameter and operation complexity of the deep learning model, taking special care of the issues of the computation and storage capacity of the edge device, and comparing target classification on different hardware with the proposed method. The remainder of this paper is organized as follows. In Section 2, selected related work is reviewed. We present the design of lightweight method, the multi-level knowledge transfer and the pruning approach in Sections 3, 4 and 5, respectfully. In Section 6, we present extensive experiment results. Finally, we conclude this paper in Section 7.

2. Related work

Various deep model compression methods have been developed recently to compress deep learning model. These methods can be broadly categorized into four major categories: parameter sharing methods, network pruning methods, knowledge transfer (teacher–student methods) and low-rank approximation methods. Parameter sharing method is also called parameter quantization method. It refers to selecting a number of representatives in the weight parameter, and using these representatives to represent the specific values of a group of weights. This method only saves the values of these representatives, while the weight matrix only needs to save the index corresponding to each parameter, thus greatly reducing the storage cost. Chen et al. [18] proposed a HashedNets method that uses a low-cost hash function to group weights into hash buckets to share parameters where each hash bucket denotes a single parameter. Gong et al. [19] used k-means clustering to quantize the weights in fully connected layers and achieved up to 24x compression rate for their CNN network with only 1% loss on accuracy on the ImageNet challenge. Courbariaux et al. [20] proposed a binary neutral network to quantify the weight of the network, the value of the restriction weight can only be −1 or 1, which greatly simplified the design of hardware dedicated to deep learning.

Network pruning method refers to deleting redundant parameters in the network and improving network generalization capability [21]. Based on this circular pruning framework, Han et al. [22] put forward a simple and effective strategy. The disadvantage of this method is that the network connection after pruning has no continuity in the distribution, which lead to frequent handover between Central Processing Unit (CPU) cache and memory, which restricted the actual acceleration effect.

The idea of low rank approximation is to reconstruct the dense matrix from a few small matrix approximations. Denil et al. [16] used low rank approximation to compress the weights of different layers. Given the weight matrix, it is expressed as a combination of several low rank matrices, so that the calculation of the product of the matrix can greatly reduce the overall storage and computing overhead. Denton et al. [23] proposed the use of singular value decomposition to reconstruct the weight of all connected layers and decompose the weight matrix by SVD(Singular Value Decomposition). Combined with some other techniques, the convolution layer can be compressed 2–3 times by matrix decomposition, 5–13 times the total connection layer and 2 times the speed, and the loss of precision is controlled within 1%.

Knowledge transfer is also called teacher–student training. The main idea is to adopt a teacher–student strategy and use a pre-trained deep network to train a shallow network on the same task [17]. Bucilu et al. [24] first proposed this idea. They used a comprehensive network of trained deep-level network models to mark some unlabeled simulation data and used this data to train a new network. This new network showed better performance than training with raw data. Ba et al. [25] believed that the input of the softmax layer contained more supervised information than the data label, and should allowed the shallow model to imitate the deep model. However, the total amount of parameters in this method has not been significantly reduced and the effect is limited. Hinton et al. [26] thought that the output of the softmax layer would be a better choice. It contained the prediction probabilities for each category and can be considered as a soft label. They used a hyper parameter T to control the degree of smoothness of the predicted probability.

The above comparison found that the four methods using different compression ideas can reduce the size of the network. Parameter

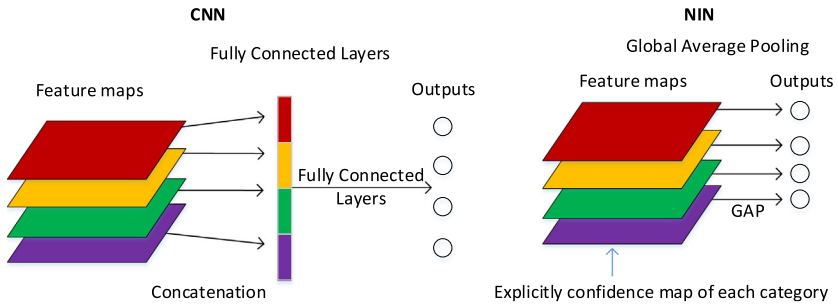

Fig. 2. Fully Connected Layers vs Global average pooling.

sharing method, network pruning method and low-rank approximation method have focused on reducing only the storage complexity of the deep models. The compressed models must be decompressed at runtime, which restricts the deployability of such compressed models on mobile devices. The knowledge transfer method has learned a lightweight network. It does not require special running libraries or hardware to reduce the running time complexity, facilitate deployment at the object device and has better versatility. Considering the complexity of the running time and the ease of deployment, the knowledge transfer method is more suitable for the scene of the object. Therefore, we research the deep neural network compression method based on knowledge transfer and hope to improve the efficiency of method.

3. Lightweight

Lightweight is designed from the perspective of both the fully connected layer and the convolution layer. They can greatly reduce the storage and computational cost of deep neural network without reducing the classification accuracy. Reducing storage cost requires reducing the amount of parameters in fully connected layer, while reducing computational cost requires reducing the amount of computation of convolution layer.

3.1. Lightweight fully connected layer

In order to reduce the storage cost of deep neural network, we need to reduce the parameters in fully connected layer. For reducing the parameter amount of the final compressed network, the most important thing is to reduce the parameter amount in the structure of the student network. It can be seen from Table 2 that the total parameter amount in AlexNet is more than 60 million and the parameter amount of the fully connected layer is more than 96% of the total parameter amount. For most convolutional neural networks, the amount of the fully connected layer parameters is more than 70% of the total parameters. Therefore, in order to reduce the amount of parameters in the "student network", it is necessary to reduce the use of full-connection layers. In this paper, by analyzing the structure of the deep neural network in recent years, we find that using the global average pooling layer instead of the fully connected layer can effectively reduce the parameter amount of the neural network.

The amount of the parameters for the fully connected layer are very large, especially the first fully connected layer directly connected to the convolution layer. On the one hand, it will increase the calculation complexity and the computation complexity in the forward derivation process; on the other hand, too many parameters may cause over-fitting. Replacing the full connection layer with a global average pooling layer can reduce the amount of neural network parameters while maintaining the function of the fully connected layer. In the neural network, the workflow of the fully connected layer is to merge the upper-layer output feature map to obtain the vector, then calculate the inner-product output. As we can see from Fig. 2, the global average pooling layer simplifies this process and calculate feature map averages to get the output. The global average pooling layer greatly reduces the number of parameters in the deep neural network and reduces the risk of overfitting.

3.2. Lightweight convolution layer

In this paper, by analyzing the universal convolutional neural networks in recent years, it is found that most of the computation of the convolutional neural networks is occupied by two parts, which are the convolution layer and the fully connected layer. By the distribution of the AlexNet calculation amount, it can be calculated that in the AlexNet, more than 90% of the computation is in the convolution layer. One of the reasons is that there are many layers of convolution layers in deep neural networks. Another reason is that in the AlexNet, the convolution kernel size is large, which leads to the ratio of the calculation of the convolution layer to the parameter amount larger than the fully connected layer, so that the convolution layer occupies 92% of the calculation cost with 3.77% of the parameter. There are many layers in convolution layers of deep neural network, which is the foundation of deep neural network. Only if the number of convolution layers is large enough, the deep neural network's expression ability can be strong enough, and the deep neural network model's prediction accuracy can be high. Therefore, in order to reduce the computation cost of the deep neural network model, it is necessary to reduce the number of large convolution kernels. To reduce the number of large convolutional kernels, we take two methods. One method is to decompose large convolution kernels into small convolution kernels and the other uses separable convolutions [27] instead of standard convolutions. The standard convolution has two effects, one is to space aggregation and the other is to change dimensions. Space aggregation is the result of convolution, which is calculated by the convolution kernel on a sliding window. The dimension transformation means that the number of channels output by each convolutional layer, which can be changed according to needs. The parameters of the convolution layer are related to the size of the convolution kernel, the number of input channels, and the number of output channels. Instead of replacing convolution kernels, we reduce the number of parameters in the convolution layer by changing the number of channels.

4. Multi-level knowledge transfer

In order to solve the problem of the lack of supervised information in the knowledge transfer method, multi-level knowledge transfer method is used during the training process, which can improve the classification accuracy of "student network". The method of multi-layer knowledge transfer uses a combination of the middle layer learning and logits layer learning. The whole framework is shown in Fig. 3. The middle layer refers to the hidden layer before the logits layer of the deep neural network. The output of these layers is the feature map learned by the neural network. The logits layer is the output of the layer before softmax activation.

4.1. Middle layer learning

In the middle layer, it learned the network variety, which means that we can make the output feature map from the middle layer of

Table 2
Alexnet parameters.

Layer	Parameter shape	Parameter amount	The ratio of parameters
Convolution layer 1	3*11*11*96	34 848	0.01%
Convolution layer 2	48*5*5*256	307 200	0.50%
Convolution layer 3	256*3*3*384	884 736	1.45%
Convolution layer 4	384*3*3*192	663 552	1.09%
Convolution layer 5	192*3*3*256	442 368	0.72%
Fully connected Layer 1	256*6*6*4096	37 748 736	62%
Fully connected Layer 2	4096*4096	16 777 216	28%
Fully connected Layer 3	4096*1000	4 096 000	6.70%

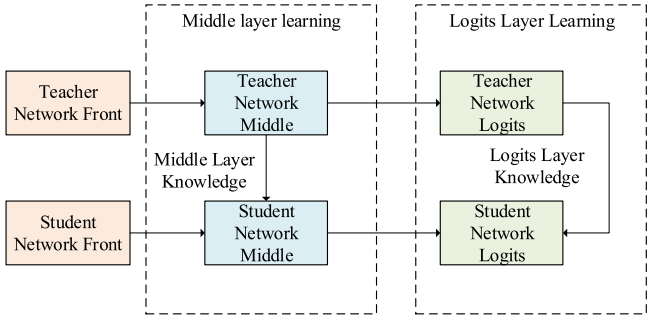

Fig. 3. Knowledge transfer learning framework.

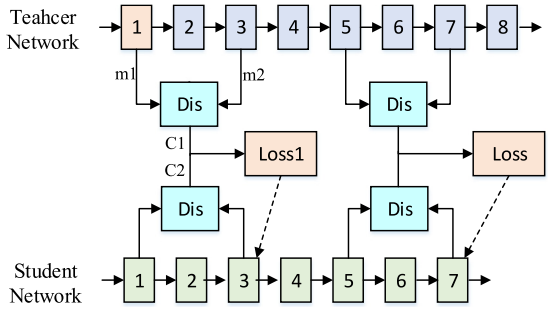

Fig. 4. The middle layer learning.

the "student network" learn the feature map from the middle layer of the "teacher network". The network variety refers to the variety of the feature map extracted when the data passes through the middle layer of the neural network, that is, the difference between the feature map extracted by the convolution layer when the data first enters the middle layer and leaves the middle layer. Deeper networks are more capable of changing the network and are more accurate. The "teacher network" becomes deeper. There are more nonlinear transformation in the network and it has stronger network expression. So the classification accuracy of the "teacher network" trained from the same data is normally higher than that of the student network.

In order to make the classification of "student network" more accurate, we need to improve the ability of sensing network variety, which is equivalent to increasing the depth of network. By learning the network variety, the "student network" can learn the "teacher network" transformation. This method is more effective than simply learning the output of the middle layer of the "teacher network" that can improve the expression ability of "student network". The specific method for learning network variety in the middle layer are shown in Fig. 4. In the upper part of Fig. 4, the deeper network depth represents the "teacher network", while the lower part is the "student network". When the data in the "teacher network" has just entered layer 1 in the middle layer, the feature map m1 extracted from this layer is taken out. When the data reaches the layer 3 through the network variety of the middle layer in the teacher network, the feature map m2 of the layer is extracted. Next, the feature map m1 and the feature map m2 are introduced into a function Dis for network variety calculating. The output C1 is the network variety of the data in the middle layer of the "teacher network". The same operations are also performed in the "student network". The function Dis outputs the network variety C2 through the middle layer of the "student network". Next, the network variety C1 in the middle layer of the "teacher network" and network variety C2 in the middle layer of the "student network" are introduced into the Loss function for difference calculating. This can calculate the difference loss1 of the middle layer network variety ability between the "student network" and the "teacher network". Lastly, the calculated difference loss1 is transmitted back to the layer 3 of the middle layer in the "student network" or back propagation during the "student network" training process. After optimizing the iterative training based on the gradient descent method [28], the difference loss1 will become smaller and smaller. This indicates that the "student network" has gradually learned the ability of network variety of the teacher network, which means that the "student network" has the ability to express the network of the deep network. Then the next middle layer will be optimized. This is the process of learning network variety in the middle layer.

In the middle-layer learning process, there are two functions that play an important role. The first is the "Dis" function that calculates the network variety between the layers in the middle layer of the deep neural network. The second is the Loss function that calculates the network variety difference between the "teacher network" and the "student network" in the same layer. The Dis function is used to measure network variety. In this study, we denote a function with lower computational complexity as Eq. (1). l_j represents the output of the layer whose depth is jf. l_j and k denote the feature map of the kth channel output by the layer whose depth is j. The function $r()$ is a regression function that adjusts the dimensions of the feature map dimension in the ith layer to be the same with the jth layer [29]. After calculating the sum of the network layer difference, the sigmoid function is used to adjust the range of the difference. In this way, the "student network" can learn teacher network's network variety ability better. We can use a universal numerical difference function to calculate the loss function calculating the difference between network variety. In order to simplify the solution process, we choose the Euclidean distance with lower computational complexity, shown in Eq. (2). By these two function modules, we can efficiently calculate the neural network's network variety and the network variety difference between "student network" and teacher network.

$$\mathrm{Dis}_{i,j} = sigmoid(\sum(l_j - r(l_i))) = \frac{1}{1 + e^{-\sum_{k=1}^{m}(l_{j,k} - r(l_{i,k}))}} \tag{1}$$

$$loss(x, y) = d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

4.2. Logits layer learning

In the multi-level knowledge transfer method, the soft target with random factor is added at the logits layer [26]. This paper uses the original optimization objective and soft target optimization objective together to form the final optimization objective, as in Eq. (3). Tea is the logits layer output of the "teacher network", Stu is the logits layer

output of the "student network". V is the label in the input data. The optimization objective L_{kd} consists of two functions. The first function L_{CN} is the cross entropy between the softmax output of the "student network" and the real label of the input data. L_{KL} is the KL divergence between the softmax output of the "student network" and the soft target of the "teacher network" with the temperature parameter T. α is used to adjust the proportion of the two items before and after the final optimization objective. By adjusting the parameter α, we can control the impact of the two items on the optimization target.

$$L_{kd}(Tea, Stu, V) = (1-\alpha) \times L_{CN}(Stu, V) + 2T^2\alpha \times L_{KL}(softmax(\frac{Tea}{T}), softmax(\frac{Stu}{T})) \tag{3}$$

The aim of adding soft target with random factors in the logits layer is to make the "student network" no longer directly learn the soft target output from the logits layer in "teacher network". In turn, a proportion of noise with a certain distribution should be added to the logits layer output in "teacher network". This is the reason why this approach is effective like data enhancement [30] and ensemble learning [31].

The specific method of adding random factors to softening targets is shown in Eq. (4). The "teacher network" is t, the "student network" is s. $Dp()$ is the dropout function. Firstly, some neuron outputs are randomly selected with a given proportion in the "teacher network", and random factor are generated by using Dropout [32]. Then we construct a random numeric matrix that satisfies a certain distribution. Here we use a Gaussian distribution $G(\mu, \sigma)$ with a mean of μ and a variance of σ. This random numeric matrix has the same size as the output of the "teacher network". Then, we take the selected neuron as "mask" and select the random numbers in the same position on the random numeric matrix. These random numbers are added to the output of the neurons. Thus, the soft target is added with random factors. Two hyper parameters will be introduced here. One is the random number proportion ρ of the output, and the other is the variance of the Gaussian distribution. With the increase of ρ, the logits layer output in the "teacher network" are added more random factors and the variation range of output will be larger. With the increase of variance, the range of random factors will be larger. After a series of experiments, it is found that the soft target with random factors can significantly improve the training result of student network.

$$\begin{aligned} &L_{noise}(t, s, V) \\ &= L_{kd}(Dp(I, \rho) \times G(\mu, \sigma) + t, s, V) \\ &= (1-\alpha) \times L_{CN}(s, V) + 2T^2\alpha \times \\ &L_{KL}(softmax(\frac{Dp(I, \rho) \times G(\mu, \sigma) + t}{T}), softmax(\frac{s}{T})) \end{aligned} \tag{4}$$

5. Pruning

In the multi-level knowledge transfer method proposed in this paper, the fully connected layers have been replaced by the global average pooling layer in "student network". Therefore, the redundancy in the "student network" exists in the convolution layer and needs to be pruned. Deep neural network pruning usually involves three steps: evaluating convolution kernels, cutting convolution kernels and fine-tuning.

5.1. Convolution kernel evaluation

Evaluating the convolution kernel is the first step in the pruning method. By evaluating the importance of convolution kernel in the neural network, it is possible to understand its importance to the entire neural network. Unimportant neurons can be cut as redundant. The neuron here refers to an output channel in a convolution layer or a computational unit in a fully connected layer. The evaluation of the importance of a convolution kernel is to score the convolution kernel of each output channel of each convolution layer in a deep neural network, and this score indicates the influence of the convolution kernel on the expression ability of the neural network model. If the score is less than a certain threshold, then it can be considered less important for the neural network model and can be eliminated as redundant.

Data-independent convolution kernel evaluation method is shown in (5). N_i is the number of output channels and N_l is the number of layers. The $Score_{aaws}$ of all the channels is summed to obtain the total $Score_{all}$, and the $Score_{aaws}$ of each channel is divided by the $Score_{all}$ to obtain the normalized $Score_{Norm}$ of each channel.

$$\begin{aligned} Score_{Norm}(l, i) &= \frac{Score_{aaws}(l, i)}{\frac{1}{N_l}\sum_{i=0}^{N_l} Score_{aaws}(l, i)} \\ &= \frac{\frac{1}{n_c \times n_m \times n_n}\sum \left|C_i^l\right|}{\frac{1}{N_i}\sum_{i=0}^{N_l}\frac{1}{n_c \times n_m \times n_n}\sum \left|C_i^l\right|} \end{aligned} \tag{5}$$

$$Score_{aaws}(l, i) = \frac{1}{n_c \times n_m \times n_n}\sum \left|C_i^l\right| \tag{6}$$

In Eq. (6), $n_c \times n_m \times n_n$ is the total number of all parameters in an output channel. C_i^l is the convolution kernel of the ith output channel in layer l. We get the intermediate result m by adding the absolute values of the weight parameters of one convolution kernel, then the $Score_{aaws}$ of the channel is obtained by averaging m. This calculation method uses the statistics of the parameters within the convolution kernel and has nothing to do with the specific data, so it is a data-independent convolution kernel evaluation method. This kind of data-independent evaluation method does not need to obtain the response of each channel of the convolution layer at each time of pruning, so the calculation speed will be faster, which is consistent with the application scenario of this paper. Since the pruning is performed on a global scale, the scores calculated for each channel of each layer need to be normalized so as to avoid deviations caused by the differences between layers, which will result in the occurrence of low-score convolution kernels appear in certain layers.

5.2. Global iterative pruning

Deep neural network pruning involves three steps: evaluating convolution kernels, cutting convolution kernel and fine-tuning. For the specified pruning ratio, these three steps are performed in sequence. The neurons are first evaluated. The convolution kernels of each output channel of all convolution layers are uniformly scored using an improved data-independent convolution kernel evaluation method and ranked according to the score. Since the normalization operation is performed during the evaluation of the convolution kernel, the importance of each convolution kernel in each convolution layer can be compared globally. Next, the unimportant neurons are pruned. Firstly, the convolution kernels are sorted according to the scores calculated in the first step. Secondly, the score threshold is calculated according to the pruning ratio, and the value of weight parameter as well as bias parameter of the convolution kernel whose score are below this threshold are set to zero. Next, a deep neural network is reconstructed, the structure and weight parameter values of the non-zero convolution kernel in the original neural network model are copied. The new neural network thus obtained is the pruned result. Finally, the neural network is fine-tuned because the neural network obtained after pruning has changed, and the network pruning causes the parameters of neural network to deviate from the local optimum obtained by the gradient descent method during training. The neural network classification ability may suffer some loss. Therefore, the neural network after pruning needs to be fine-tuned, and a certain batch of training data is used to retrain the pruned neural network so that the neural network model classification capability can be improved. The entire pruning process is shown in Fig. 5.

In order to improve the effect of network pruning, the paper uses the global iterative pruning method. The pruning of non-essential convolution kernels is performed in accordance with uniform standards in all

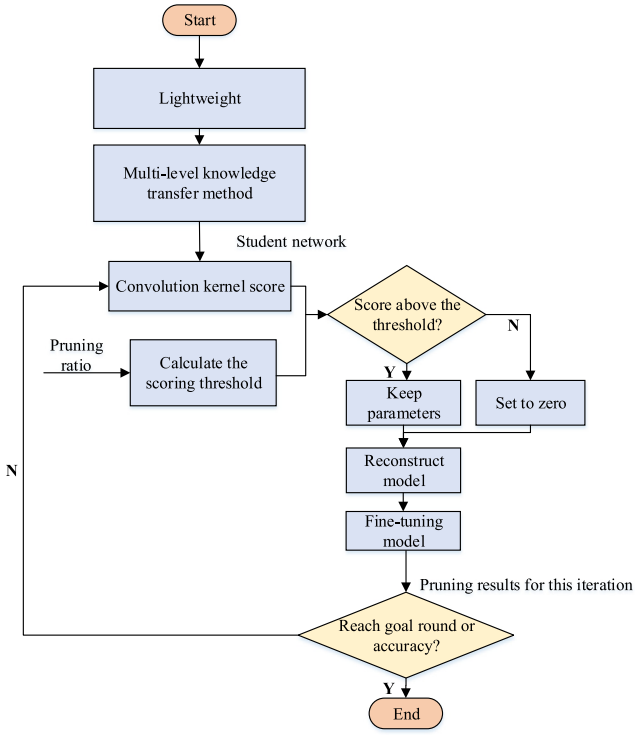

Fig. 5. Global iterative pruning.

output channels of all convolution layers instead of layer by layer. This eliminates the need for fine-tuning every layer and speeds up pruning effectiveness. In addition, in order to ensure that the loss of classification ability caused by neural network pruning can be recovered only after fine-tuning several times after each pruning, an iterative pruning method is used in this paper. Only a small part of non-essential convolution kernels are pruned at each iteration, which make the neural network structure change little and ensures that the parameters of the neural network model as much as possible to maintain the original local optimum. In the global iterative network pruning method, it is not necessary to specify the number of convolution kernels to be pruned for each convolutional layer, but only to specify the same pruning ratio for the entire network at one iteration. Under the given original "student network" classification capability, this method can execute the "evaluating convolution kernels, cutting convolution kernels and fine tuning" step through multiple iterations to continuously approach the optimal parameters of "student network".

6. Experimental evaluation

6.1. Experiment environment

In this section, we evaluate the performance of the proposed algorithm. Our experiments are performed on three different platforms, such as, a GPU server, a Raspberry Pi 3 [33] and a Cambricon-1A platform [34,35]. The experimental platforms' configurations are shown in Table 3. Most of the code in this article was developed using Python based on MXNet [36]. The experimental dataset used in this paper is a small-category dataset of cifar-10 [37], and the dataset has been augmented [30].

6.2. Lightweight and multi-level knowledge transfer experiments

The traditional CNN may be roughly divided into two kinds of structures. One is the stack network structure, such as VGG, Alexnet, and the other is the residual network structure, such as ResNet, DenseNet [38]. In this paper, we mainly use the VGG model, but the compression method in this paper can also be applied to the network structure of the residual structure. Firstly, the experimental verification in this section uses a lightweight approach to design "student network" based on the "teacher network". Next, we use the cifar-10 dataset to train the "teacher network" and "student network". Under the same experimental condition, a comparative training experiment between the method proposed in this paper with the knowledge distillation (KD method) is conducted. The evaluation indicators in the experiment include the classification accuracy of the neural network, the compression ratio of the neural network and the implementation of an inference time of the neural network.

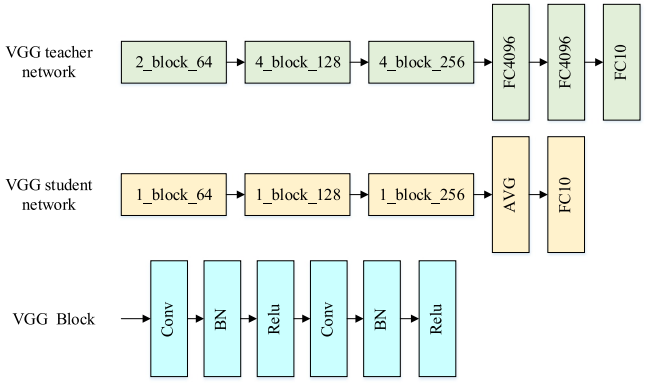

Fig. 6. VGG Experiment Network Structure.

6.2.1. Neural network structure design

As shown in Fig. 6, the VGG based "teacher network" and "student network" structure designed by the lightweight method are detailed.

In "student network", the global average pooling layer is used to replace the unnecessary fully connected layer, and a large number of convolutional layers are reduced to decrease the depth of the neural network. Here, the "teacher network" uses a VGG network with a depth of 23 layers, including 20 convolution layers and 3 fully connected layers. The depth of the "student network" is of 8 layers, including 6 layers of convolution layers and 2 layers of fully connected layers. The output of last fully connected layer is related to the specific dataset being used. If a cifar-10 dataset is used, the output channel of the last fully connected layer is 10 since the dataset has 10 types of targets.

6.2.2. Cifar-10 dataset experiment

The 60 000 32 × 32 color images of the cifar-10 dataset are divided into 10 categories, so the output dimension of the last fully connected layer of "teacher network" and "student network" are both 10. There are 50 000 training images and 10 000 test images. In the VGG experiment, the "teacher network" has 23 layers and the "student network" has 8 layers. We use the following parameters in Table 4 to train the "teacher network" and "student network" directly from the dataset. Then we use a multi-level knowledge transfer method to train the "student network". The parameter settings are shown in Table 5. Lastly, we use the knowledge distillation method to train the "student network". The parameter settings are shown in the Table 6.

The results are shown in Fig. 7. From the experimental results of the above experiments, we can see that the multi-level knowledge transfer method proposed in this paper can improve the classification accuracy of the "student network". Comparing multi-level knowledge transfer with knowledge distillation, it is found that the "student network" trained by the multi-level knowledge transfer method has a higher classification accuracy, which indicates that this method adds more supervised information to the "student network" during the training

Table 3
Platform configuration.

Platform	Processor	Memory	Graphic card	Video memory	OS
GPU server	I7 5930	8 GB*4	TitanX pascal	12 GB	Linux 16.04
Raspberry Pi 3	BCM2837	1 GB	／	／	Raspbian 2018
Cambricon-1A	Cambricon-1A	2 GB	／	／	／

Table 4
Parameter setting for teacher and student network training from scratch.

Neural network	Parameter						
	Batch size	Learning rate			Initialization method	Weight decay	Momentum
		(0~80)	(80~180)	(180~300)			
Teacher network	32	0.1	0.01	0.001	MSRA	0.0001	0.9
Student network	64	0.01	0.001	0.0001	MSRA	0.0001	0.9

Table 5
Parameter setting for multi-level knowledge transfer.

Temperature parameter	Balance parameter	Learning rate multiplication	Middle layer learning			Logits layer learning			
			(0~40)	(40~100)	(100~200)	Decay method	Decay factor	Initial learning rate	Iteration
4	0.7	0.01	0.1	0.01	0.001	Exponential decay	0.97	0.05	300

Table 6
Parameter setting for knowledge distillation.

Temperature parameter	Balance parameter	Learning rate multiplication	Decay method	Decay factor	Initial learning rate	Iteration
4	0.7	0.01	Exponential decay	0.97	0.1	300

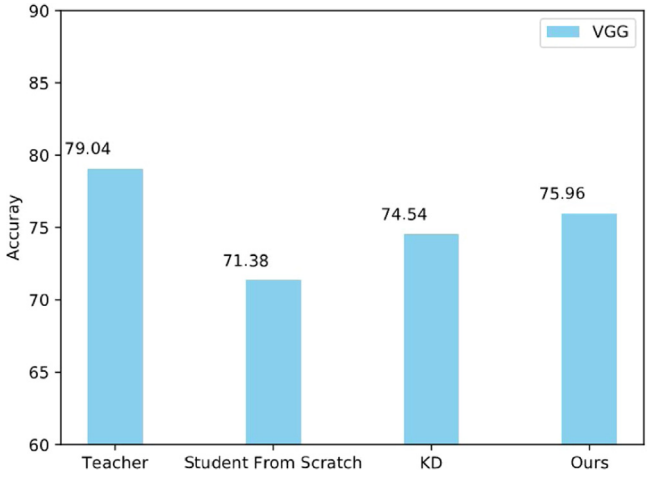

Fig. 7. Comparison of our method and knowledge distillation on cifar-10.

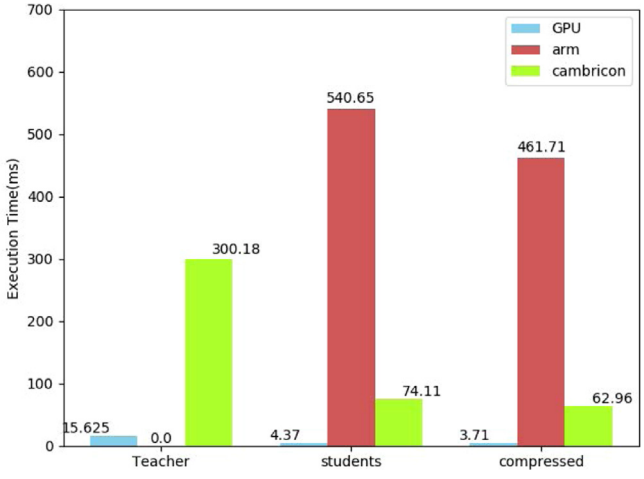

Fig. 8. The execution time comparison on the GPU server, Raspberry Pi 3 and Cambricon-1A.

process. From the experimental data point of view, this method reduces the loss rate of the classification accuracy of the "student network" by more than 15% from the original "student network" training from scratch, and this method is over 30% more efficient than the knowledge distillation method in reducing the loss rate.

According to the calculation of the neural network structure, the total number of parameters of the "teacher network" is 39.24M and most of the parameters exist in the fully connected layer. The total parameter amount of the "student network" after lightweight is 1.15M. Due to the use of a global average pooling layer, the amount of parameters can be greatly reduced. The number of parameters of the "student network" is approximately 2.9% of the "teacher network". As shown in Fig. 8, on the GPU platform, the execution time of a single inference of the "teacher network" is 15.625 ms, which cannot be run on the ARM platform due to memory overflow. It is 300.18 ms on the Cambricon-1A platform. On the GPU platform, the execution time of a single inference of the "student network" is 4.37 ms, 540.65 ms on the ARM platform, and 74.11 ms on the Cambricon-1A platform. The time efficiency of the "student network" has been greatly improved due to the reduction of convolution layers. After multi-level knowledge transfer and lightweight processing, the classification accuracy of "student network" on the cifar-10 dataset was 75.96%, and the network model parameter amount was 1.15M.

6.2.3. *Pruning experiment*

Through the multi-level knowledge transfer, we can get a "student network", which is set as the input model of pruning experiments. In

Table 7
Parameter setting in the pruning experiments for student network.

Batch size	Initial learning rate	Single iteration training times	Single iteration pruning ratio	Iteration times
128	0.001	10	0.02	15

Table 8
Network parameters after global iterative pruning.

	Parameter shape before pruning	Parameter amount before pruning	Parameter shape after pruning	Parameter amount after pruning	Pruning ratio
Block1	3*3*3*64 64*3*3*64	1728 36 864	3*3*3*27 27*3*3*38	729 9234	58.8% 74.9%
Block2	64*3*3*128 128*3*3*128	73 728 147 456	38*3*3*81 81*3*3*103	27 702 75 987	62.4% 40.1%
Block3	128*3*3*256 256*3*3*256	294 912 589 824	103*3*3*215 215*3*3*256	199 305 495 360	32.4% 17.1%
fc	256*10	2560	256*10	2560	0

the experiment, the pruning ratio is set as 0.3, that is, 30% of the total parameters will be cut off. In the iterative pruning process, we set the pruning step as 0.02 and the momentum as 0.9. In this paper, we use batch gradient descent method to adjust the learning rate manually, and the specific parameters are shown in Table 7.

After 15 times pruning with pruning ratios of 2% and fine tuning, we get the compressed network, the classification accuracy of which is 74.04% and the total amount of network parameters is 0.81M detailed in Table 8. From the table, we can see that network pruning can effectively eliminate redundant parameters existing in convolution layer, and the effect of pruning is better in the lower layer network of neural network.

In terms of execution time of compress network on the GPU server, Raspberry Pi 3 and Cambricon-1A shown in Fig. 8, the execution time of a single feedforward operation on GPU server is 3.71 ms, the Raspberry Pi 3 is 461.71 ms and the Cambricon-1A is 62.96 ms. By the analysis of the experimental results, it is found that the "student network" model can still be pruned by the channel pruning method. On the basis of reducing the neural network parameters by 30%, the classification accuracy of the compressed network model is reduced by 1.92%, and the time complexity is improved by at least 15% than the "student network". Through the experiments above, we can draw a conclusion that the knowledge transfer integrating with the pruning method can reduce more parameters and the time complexity of the neural network. In summary, by the experiment based on Cifar-10 dataset and the pruning experiment benchmarked on GPU server, Raspberry Pi 3 and Cambricon-1A, it is found that the parameters in the compressed network have achieved more than 49.5 times compression and the time efficiency of a single feedforward operation has been improved more than 3.2 times than the "teacher network".

7. Conclusions and future works

Toward to intelligent applications of the edge device, we propose the deep neural network compression algorithm based on knowledge transfer with three stages: lightweight, multi-level knowledge transfer and pruning that reduce the network depth, parameter and operation complexity of the deep learning model. Refer to the future work, we will implement the algorithm on Cambricon MLU220 and plan to conduct a series of tests to evaluate and fine-tune the design. In addition, further research is carried out in the direction of the measurement of network variety in the middle layer and the new network structure design.

CRediT authorship contribution statement

Yanming Chen: Conceptualization, Methodology, Writing - reviewing & editing. **Chao Li:** Resources, Visualization, Conceptualization, Methodology, Reviewing & editing. **Luqi Gong:** Formal analysis, Software. **Xiang Wen:** Formal analysis, Writing. **Yiwen Zhang:** Conceptualization, Methodology, Formal analysis, Validation, Reviewing. **Weisong Shi:** Conceptualization, Methodology, Formal analysis, Validation, Reviewing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work is supported in part by the National Natural Science Foundation of China under Grant (No. 61702487, No. 61802001).

References

[1] W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: Vision and challenges, IEEE Internet Things J. 3 (5) (2016) 637–646, http://dx.doi.org/10.1109/jiot.2016.2579198.
[2] W. Shi, S. Dustdar, The promise of edge computing, Computer 49 (5) (2016) 78–81, http://dx.doi.org/10.1109/mc.2016.145.
[3] X. Xu, B. Shen, X. Yin, M.R. Khosravi, H. Wu, L. Qi, S. Wan, Edge server quantification and placement for offloading social media services in industrial cognitive IoV, IEEE Trans. Ind. Inf. (2020) 1, http://dx.doi.org/10.1109/tii.2020.2987994.
[4] L. Qi, X. Wang, X. Xu, W. Dou, S. Li, Privacy-aware cross-platform service recommendation based on enhanced locality-sensitive hashing, IEEE Trans. Netw. Sci. Eng. (2020) 1, http://dx.doi.org/10.1109/tnse.2020.2969489.
[5] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436, http://dx.doi.org/10.1038/nature14539.
[6] L. Liu, W. Ouyang, X. Wang, P. Fieguth, J. Chen, X. Liu, M. Pietikäinen, Deep learning for generic object detection: A survey, Int. J. Comput. Vis. 128 (2) (2020) 261–318, http://dx.doi.org/10.1007/s11263-019-01247-4.
[7] Y. Belinkov, J. Glass, Analysis methods in neural language processing: A survey, Trans. Assoc. Comput. Linguist. 7 (2019) 49–72, http://dx.doi.org/10.1162/tacl_a_00254.
[8] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105, http://dx.doi.org/10.1145/3065386.
[9] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, 2014, arXiv preprint arXiv:1409.1556.
[10] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9, http://dx.doi.org/10.1109/cvpr.2015.7298594.
[11] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778, http://dx.doi.org/10.1109/cvpr.2016.90.
[12] Y. Zhang, C. Yin, Q. Wu, Q. He, H. Zhu, Location-aware deep collaborative filtering for service recommendation, IEEE Trans. Syst. Man Cybern.: Syst. (2019) 1–12, http://dx.doi.org/10.1109/tsmc.2019.2931723.
[13] S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, 2015, arXiv preprint arXiv:1510.00149.

[14] L. Qi, Q. He, F. Chen, X. Zhang, W. Dou, Q. Ni, Data-driven web APIs recommendation for building web applications, IEEE Trans. Big Data (2020) 1, http://dx.doi.org/10.1109/tbdata.2020.2975587.
[15] Y. Zhang, K. Wang, Q. He, F. Chen, S. Deng, Z. Zheng, Y. Yang, Covering-based web service quality prediction via neighborhood-aware matrix factorization, IEEE Trans. Serv. Comput. (2019) 1, http://dx.doi.org/10.1109/tsc.2019.2891517.
[16] M. Denil, B. Shakibi, L. Dinh, N. De Freitas, et al., Predicting parameters in deep learning, in: Advances in Neural Information Processing Systems, 2013, pp. 2148–2156.
[17] B.B. Sau, V.N. Balasubramanian, Deep model compression: Distilling knowledge from noisy teachers, 2016, arXiv preprint arXiv:1610.09650.
[18] W. Chen, J. Wilson, S. Tyree, K. Weinberger, Y. Chen, Compressing neural networks with the hashing trick, in: International Conference on Machine Learning, 2015, pp. 2285–2294.
[19] Y. Gong, L. Liu, M. Yang, L. Bourdev, Compressing deep convolutional networks using vector quantization, 2014, arXiv preprint arXiv:1412.6115.
[20] M. Courbariaux, Y. Bengio, J.-P. David, Binaryconnect: Training deep neural networks with binary weights during propagations, in: Advances in Neural Information Processing Systems, 2015, pp. 3123–3131.
[21] R. Setiono, H. Liu, Neural-network feature selector, IEEE Trans. Neural Netw. 8 (3) (1997) 654–662, http://dx.doi.org/10.1109/72.572104.
[22] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, 2015, pp. 1135–1143.
[23] E.L. Denton, W. Zaremba, J. Bruna, Y. LeCun, R. Fergus, Exploiting linear structure within convolutional networks for efficient evaluation, in: Advances in Neural Information Processing Systems, 2014, pp. 1269–1277.
[24] C. Bucilua, R. Caruana, A. Niculescu-Mizil, Model compression, in: Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, 2006, pp. 535–541, http://dx.doi.org/10.1145/1150402.1150464.
[25] J. Ba, R. Caruana, Do deep nets really need to be deep? in: Advances in Neural Information Processing Systems, 2014, pp. 2654–2662.
[26] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, 2015, arXiv preprint arXiv:1503.02531.
[27] F. Chollet, Xception: Deep learning with depthwise separable convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 1251–1258, http://dx.doi.org/10.1109/cvpr.2017.195.
[28] J. Duchi, E. Hazan, Y. Singer, Adaptive subgradient methods for online learning and stochastic optimization, J. Mach. Learn. Res. 12 (Jul) (2011) 2121–2159.
[29] A. Romero, N. Ballas, S.E. Kahou, A. Chassang, C. Gatta, Y. Bengio, Fitnets: Hints for thin deep nets, 2014, arXiv preprint arXiv:1412.6550.
[30] A. Fawzi, H. Samulowitz, D. Turaga, P. Frossard, Adaptive data augmentation for image classification, in: 2016 IEEE International Conference on Image Processing, ICIP, IEEE, 2016, pp. 3688–3692, http://dx.doi.org/10.1109/icip.2016.7533048.
[31] T.G. Dietterich, Ensemble methods in machine learning, in: International Workshop on Multiple Classifier Systems, Springer, 2000, pp. 1–15, http://dx.doi.org/10.1007/3-540-45014-9_1.
[32] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
[33] U. Learning, Raspberry pi 3: get started with raspberry pi 3 a simple guide to understanding and programming raspberry pi 3 (raspberry pi 3 user guide, python programming, mathematica programming), CreateSpace Independent Publishing Platform, North Charleston, SC, USA, 2016.
[34] Y. Chen, T. Luo, S. Liu, S. Zhang, L. He, J. Wang, L. Li, T. Chen, Z. Xu, N. Sun, et al., Dadiannao: A machine-learning supercomputer, in: Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture, IEEE Computer Society, 2014, pp. 609–622, http://dx.doi.org/10.1109/micro.2014.58.
[35] S. Liu, Z. Du, J. Tao, D. Han, T. Luo, Y. Xie, Y. Chen, T. Chen, Cambricon: An instruction set architecture for neural networks, in: 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture, ISCA, 2016, pp. 393–405, http://dx.doi.org/10.1109/isca.2016.42.
[36] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, Z. Zhang, Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems, 2015, arXiv preprint arXiv:1512.01274.
[37] A. Krizhevsky, G. Hinton, et al., Learning Multiple Layers of Features from Tiny Images, Technical Report, Citeseer, 2009.
[38] G. Huang, Z. Liu, L. Van Der Maaten, K.Q. Weinberger, Densely connected convolutional networks, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 4700–4708, http://dx.doi.org/10.1109/cvpr.2017.243.