An Incremental Deep Convolutional Computation Model for Feature Learning on Industrial Big Data

Peng Li, Zhikui Chen, Laurence Tianruo Yang, Jing Gao, Qingchen Zhang, and M. Jamal Deen

***Abstract*—The deep convolutional computation model (DCCM) enabled remarkable progress in feature learning of industrial big data in Internet of Things. However, as a typical static deep learning model, it is difficult to learn features for incremental industrial big data. To solve this problem, we propose an incremental DCCM by developing two incremental algorithms, i.e., parameter-incremental algorithm and structure-incremental algorithm. The parameter-incremental algorithm aims to incrementally train the fully connected layers together with fine tuning for incorporating the new knowledge into the prior one. Then, the structure-incremental algorithm is used to transfer the previous knowledge by introducing an updating rule of the tensor convolutional, pooling, and fully connected layers. Furthermore, the dropout strategy is extended into the tensor fully connected layer to improve the robustness of the proposed model. Finally, extensive experiments are carried out on the representative datasets including CIFRA and CUAVE to justify the proposed model in terms of adaption, preservation, and convergence efficiency.**

***Index Terms*—Deep convolutional computation model (DCCM), incremental learning, industrial big data, tensor computation.**

I. Introduction

RECENTLY, many deep learning computational, predictive, and management architectures were proposed for industrial big data of Internet of Things [1]–[4]. These architectures can be classified into two schemes. One scheme uses tendency used advanced computational techniques to construct sophisticated models [5]. The other scheme extracts important information from big data to train robust models [6]. This second scheme has resulted in a more robust model with the same accuracy as the first scheme, but with a lower training cost. Also, for heterogeneous data feature learning, a robust and fast deep convolutional computation model (DCCM) was obtained by extending the traditional convolutional neural network (CNN) into the tensor space [7]. This CNN model achieved a higher classification accuracy than other deep learning models, such as deep computation model [8] and multimodal deep learning models [9] for heterogeneous data. However, because the DCCM is a static deep learning model, it is difficult for it to learn the features for industrial big data.

The velocity at which industrial big data is now generated is high and is steadily increasing [10]–[13]. This rapidly increasing volume of big data must be collected and processed in real time. However, the distribution of industrial big data is often very diverse, and so, it cannot be handled by static learning models. For example, if a deep model is trained for dataset X, then when Z is collected, the well-trained static model may not produce desired results for Z if Z has different distribution characteristics from X. To tackle this challenge, we can train a completely new architecture for X and Z in the batch method [14]. However, this way is time consuming and can even be impossible, with the result that it cannot satisfy the requirement of real-time processing. An effective solution to address this problem is to use incremental learning [15] that is well suited for learning the features of industrial big data.

The incremental learning method aims to accommodate new patterns without compromising the loss of historical knowledge [16], [17]. Furthermore, it is very efficient because it does not need to train the model on the entire historical data. Many incremental algorithms were proposed, and among them, two representative kinds of the incremental learning are the parameter-incremental algorithm and the structure incremental algorithm. For example, the incremental backpropagation [18] and online learning [19] are the typical parameter-incremental algorithms. These methods adapt the new input by designing some new constraint rules to modify the weights and biases of the trained model. Also, the new input is fed into the model in a sequential fashion. In addition, because it is not necessary to keep

Manuscript received February 5, 2018; revised April 1, 2018 and May 24, 2018; accepted August 2, 2018. Date of publication September 19, 2018; date of current version March 1, 2019. This work was supported in part by the National Natural Science Foundation of China under Grant 61672123 and Grant 61602083, in part by the Major Science and Technology Planning Project of Guangdong Province under Grant 2015B010110006, in part by the Doctoral Scientific Research Foundation of Liaoning Province 20170520425, and in part by Dalian University of Technology Fundamental Research Fund under Grant DUT15RC(3)100. The work of M. J. Deen was supported by the Canada Research Chair program. Paper TII-18-0355. *(Corresponding author: Laurence Tianruo Yang.)*

P. Li is with the School of Software Technology, Dalian University of Technology, Dalian 116600, China, and also with the Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON L8S 4L8, Canada (e-mail: lipeng2015@mail.dlut.edu.cn).

Z. Chen and J. Gao are with the School of Software Technology, Dalian University of Technology, Dalian 116600, China (e-mail: zkchen@dlut.edu.cn; gaojing@dlut.edu.cn).

L. T. Yang and Q. Zhang are with the Department of Computer Science, St. Francis Xavier University, Antigonish, NS B2G 2W5, Canada (e-mail: ltyang@ieee.org; qzhang@stfx.ca).

M. J. Deen is with the Department of Electrical and Computer Engineering, McMaster University, Hamilton, ON L8S 4L8, Canada (e-mail: jamal@mcmaster.ca).

Digital Object Identifier 10.1109/TII.2018.2871084

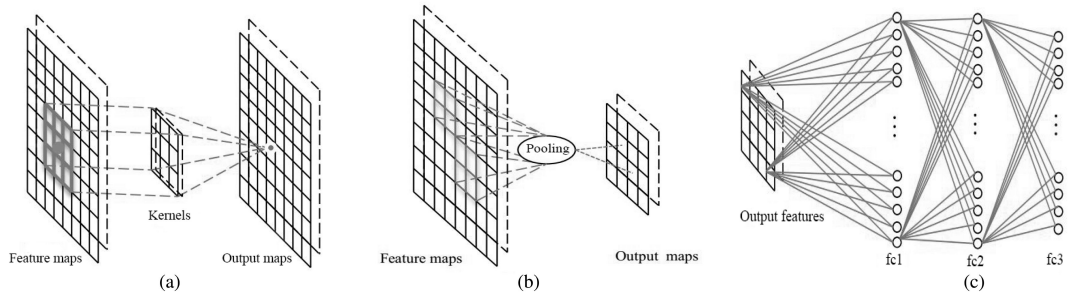


Fig. 1. Tensor computation of the deep convolutional computation model. (a) Tensor convolution. (b) Tensor pooling. (c) Fully connected.

the whole dataset in memory, these algorithms work well for real-time massive data applications.

A popular structure-incremental algorithm is the Learn++ [20]. In this algorithm, a new multilayer perceptron that captures the features of the new input is added to the ensemble of pretrained classifiers in updating the network structure. This kind algorithm is especially suitable for feature learning in dynamic applications. However, these incremental algorithms used traditional classification methods and are not suitable for the DCCM. Therefore, in this paper, we focus on improving the DCCM for diverse industrial big data.

Our incremental DCCM is designed to capture the heterogeneous features of the industrial big data over the tensor space, and it can incorporate the new features without forgetting the historical knowledge. It is developed by the parameter-incremental learning algorithm (PIL) and structure-incremental learning algorithm (SIL). In PIL, to speed up the incremental learning, the initial increments are computed by a novel loss function that considers the performance of adaptation and preservation. Simultaneously, a novel dropout strategy is designed to further guarantee the performance of adaptation and preservation. In this novel dropout strategy, each neuron of the tensor fullyconnected layers is grouped into three subsets and each kind of neuron is allocated with a different switch probability to make full use of the idle neurons learning the new patterns. Finally, a fine-tuning training is adapted to merge the historical knowledge and current knowledge.

In SIL, the updating rules of tensor convolutional, pooling, and fully connected layers are designed to transfer the historical knowledge. Furthermore, the virtual neurons with weights that are always assigned to be zero are used in the updating process to ensure the computing of the tensor multidot product, convolution, and pooling. Simultaneously, the dropout strategy is extended into the tensor space, which further improves the robustness of the structure-incremental DCCM. Finally, massive experiments are conducted on the typical datasets: CUAVE and CIFAR to evaluate the performance of the incremental DCCM. The results demonstrate the efficiency of our proposed model in adaptation of new knowledge and preservation of historical knowledge.

The three major contributions of this paper can be summarized as follows.

1) To effectively capture hidden incremental features of big data, an incremental DCCM is proposed, in which the parameter-incremental algorithm and the structure-incremental algorithm are designed (Section III). Extensive experiments are conducted to evaluate the performance of the proposed model (Section IV).
2) To learn features of the new input with the similar distribution, a parameter-incremental DCCM is designed without losing the historical knowledge (Section III-A). In the model, a novel loss function speeds up the learning process, an improved dropout scheme is designed to reinforce the training of idle subarchitectures of tensor fully connected layers, and a fine-tuning training is used to incorporate the historical knowledge.
3) To learn features of the new input from the dynamic environment, a structure-incremental DCCM is designed (Section III-B), which transfers the historical knowledge of the historical data by devising the updating rule of tensor convolutional, pooling, and fully connected layers without retraining from the whole data. Moreover, the dropout strategy is also extended into tensor space to further improve the performance of the DCCM.

II. Preliminaries

A. Deep Convolutional Computation Model

Based on the tensor data representation model, a DCCM is devised to learn features for the heterogeneous data [21], [22]. In DCCM, the tensor representation is used to model each heterogeneous object, capturing the complexly nonlinear relationship. Based on this tensor structure, the tensor convolution is defined, achieving a property of the parameter sharing. Then, a high-order backpropagation algorithm (HBP) is proposed to train parameters of DCCM. And the three components of DCCM are demonstrated in Fig. 1.

In Fig. 1(a), the N-order input tensor I is mapped by the N-order kernel tensor K in the following form:

$$FT = f\left(I * K + b\right) \tag{1}$$

where $*$ represents the tensor convolution, f denotes the nonlinear function, and b is a bias tensor.

As shown in Fig. 2(b), the N-order input tensor T of the tensor pooling layer is mapped to a lower dimension space as follows:

$$FT = f\left(\beta \cdot \text{pool}\left(T\right) + b\right) \tag{2}$$

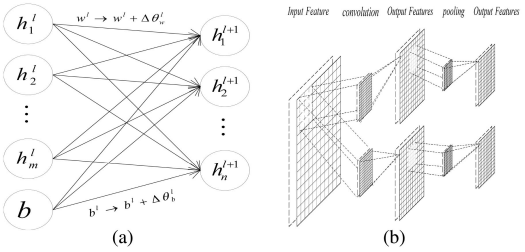


Fig. 2. Incremental DCCM. (a) Parameter computation method. (b) Basic constrained network.

Algorithm 1: High-order Back-Propagation Algorithm.

Input: Tensor example $\{(X^{(i)}, Y^{(i)})\}$, $step_{\max}$, Learning rate η, $threshold$

Output: Updated parameters $\theta = W, b, K, \beta$

Forward Pass:

Extract features

Tensor convolution $Map_c = f(X * K + b)$;

Tensor pooling $Map_s = f(\beta \cdot pool(Map_c) + b)$;

Map features and compute outputs

$H = f(W \odot Map_s + b)$;

Backprogation Process:

Differentiate loss $\Delta\delta_\theta$ with respect to $\theta = W, b, K, \beta$;

Update tensor fully connected layer

$W_f = W_f - \eta\nabla\delta_{\theta W_f}$;

Update tensor pooling layer $W_s = W_s - \eta\nabla\delta_{\theta W_s}$;

Update tensor pooling layer $W_c = W_c - \eta\nabla\delta_{\theta W_c}$;

where $\mathrm{pool}(T)$ produces a compact tensor, and f and b denote the nonlinear function and the bias tensor, respectively.

Fig. 1(c) shows the tensor fullyconnected layer as follows:

$$H = f(W \odot I + b) \tag{3}$$

where $\odot$ represents the multidot product and $N+1$-order tensor $W = R^{\beta \times i_1 \times i_2 \cdots i_N}$ denotes the weight tensor with the $\beta = j_1 \times j_2 \cdots j_N$.

Finally, the brief of the high-order learning algorithm is in Algorithm 1.

B. Dropout: A Regularization Strategy

The dropout is an effective method preventing the supervised neural network from overfitting [23], [24]. In dropout, each neuron is dropped randomly, sampling an ensemble of thinned deep architectures. For a dropout network, $z^{(h)}$, $W^{(h)}$, $W^{(h)}$, and $b^{(h)}$ are the input, output, weight, and bias of the hidden layer h, respectively. The feedforward is described as follows:

$$\begin{aligned} s_i^{(\mathrm{h})} &\sim \mathrm{Bernoulli}(p) \\ \widetilde{\mathbf{a}}^{(l)} &= \mathbf{s}^{(\mathrm{h})} * \mathbf{a}^{(l)} \\ z_j^{(l+1)} &= \mathbf{w}_j^{(l+1)} \widetilde{\mathbf{a}}^{(l)} + b_j^{(l+1)} \\ a_j^{(l+1)} &= f\left(z_i^{(l+1)}\right) \end{aligned} \tag{4}$$

where $\mathbf{s}^{(\mathrm{h})}$ is the mask vector with each element subjected to the Bernoulli distribution. $\widetilde{\mathbf{a}}^{(l)}$ is the masked output vector. In the backpropagation, the loss only passes through the selected architecture. Finally, the whole architecture without dropout computes the prediction value with the parameter scaled by a factor. And the dropout is adopted to reuse the idle subnetworks to capture the new knowledge in this paper.

III. Incremental DCCM

The DCCM can effectively extract heterogeneous features from the static big data by designing an elaborate architecture. However, it is still a challenge for DCCM trained by the HBP to capture the dynamic features of the fast-growing big data. To solve this challenge, two incremental algorithms, i.e., PIL and SIL are proposed. Especially, PIL learns the features of the new data of the similar distribution by updating parameters of the fully connected layer, whereas SIL updates the architecture of model to capture features of new data collected from dynamic environment.

A. PIL Algorithm

The parameter-incremental algorithm includes two stages: the initial increment computation and the incremental training. In the initial increment computation, the parameter increment is computed to speed up the incremental training based on a new loss function that adopts new examples without losing the historical knowledge in the tensor space. In the incremental training, the idle nodes of tensor fully connected layers are trained to capture the new knowledge based on the are trained to capture the new knowledge. Then, a fine-tuning training is incorporated to merge the new knowledge with the historical knowledge.

1) Initial Increment Computation: In Fig. 2(a), $\theta = \{W^l, b^l | l = 1, 2, \ldots, L\}$ denotes the historical knowledge, whereas $\Delta\theta = \{\Delta W^l, \Delta b^l | l = 1, 2, \ldots, L\}$ is the initial parameter increment computed from the new data. To improve the efficiency, the $\theta + \Delta\theta$ should be close to the final value, reducing the steps in the incremental training. Since the final θ considers the performance of adaption and preservation, $\Delta\theta$ cannot learn the new data, but preserve the knowledge from the historical data.

To compute the parameter increment $\Delta\theta$, we adopt the similar strategy of the computation of the parameter increment [25]. Differently, a simplified loss function is used, because the goal of the computation of the parameter increment is only to speed up the training process. In details, the weight and bias tensors are first unfolded into the vector form effectively by the following formula:

$$x_l = X_{i_1 i_2 \cdots i_N}, l = i_1 + \sum_{j=2}^{N} \prod_{k=1}^{j-1} I_k \tag{5}$$

where x_l and $X_{i_1 i_2 \ldots i_N}$ represent the element of the vector x and the tensor X, respectively. And the novel loss function $J_{\mathrm{Increment}}$ is adopted to data vectors unfolded from the tensor, as follows:

$$J_{\mathrm{Increment}} = J_{\mathrm{Adaption}} + J_{\mathrm{Preservation}} \tag{6}$$

where J_{Adaption} and $J_{\mathrm{Preservation}}$ are the postprediction loss and the deformation loss, respectively. Specifically, given a new pair of

data sample (x, y), J_{Adaption} is the metric of the loss between the target label y and the output of the network with the updated parameter $\theta + \Delta\theta$. It is defined as form:

$$J_{\text{Adaption}} = \frac{1}{2}\Delta y^T \Theta \Delta y, \Delta y = H_{\theta+\Delta\theta}(x) - y \tag{7}$$

where $H_{\theta+\Delta\theta}(x)$ denotes the reconstructured output and the Θ is the weight matrix, $\Theta = \left(I - u\left(\frac{\partial H}{\partial \theta}\right)^T \frac{\partial H}{\partial \theta}\right)^{-1}$. Simultaneously, to evaluate the performance of preservation, the deformation caused by the parameter adaption to the new data is measured by the L^2-norm between the updated output and the primitive output of the computation model. The deformation loss $J_{\text{Preservation}}$ is expressed as

$$J_{\text{Preservation}} = \frac{1}{2}\left\|H_{\theta+\Delta\theta}(x) - H_\theta(x)\right\|_2^2. \tag{8}$$

Thus, the $J_{\text{Increment}}$ is expressed in the following form:

$$J_{\text{Increment}} = \frac{1}{2}\Delta y^T \Theta \Delta y + \frac{1}{2}\left\|H_{\theta+\Delta\theta}(x) - H_\theta(x)\right\|_2^2. \tag{9}$$

The increment computation problem is stated as: for a new data pair (x, y), compute the parameter increment $\Delta\theta$ modifying the weight and bias vector θ to minimize the $J_{\text{Increment}}$, as follows:

$$\frac{\partial J}{\partial \Delta\theta} = \frac{\partial}{\partial \Delta\theta}\left(\frac{1}{2}\Delta y^T \Theta \Delta y + \frac{1}{2}\left\|H_{\theta+\Delta\theta}(x) - H_\theta(x)\right\|_2^2\right) = 0. \tag{10}$$

Since the nonlinear function is used to capture the hidden correlations over the heterogeneous high-order space, it is difficult to compute an analytical $\Delta\theta$ by directly solving the Formula (10). Thus, Taylor's Formula and the deformation of the parameter from the global level to the neuron level are used to find a suboptimal solution to the parameter increment. And the suboptimal solution $\Delta\theta$ is expressed as

$$\Delta\theta = -\mu\frac{\partial H_\theta(x)}{\partial \theta}\Delta y. \tag{11}$$

Once the parameter increment is obtained, (5) is used to form the tensor parameter increment.

2) Incremental Training: To make DCCM capture the intrinsic features of the whole data, an incremental training algorithm is designed, composed of the dropout training and the final fine-tuning training.

In the dropout training, to discover the free subarchitecture in the tensor fully connected layers, we first explore the weight space and classify the neurons of tensor fully connected layers into three subsets by two surfaces of the ball with radii R_1 and R_2 ($R_1 \leq 0.4$ and $R_2 \leq 3$, when $N = 3$). Each neuron of each subset is assigned with a probability p_i $\{i = 1, 2, 3\}$. p_i is subjected to the Bernoulli(p_i) distribution, indicating that this unit is dropped out with the probability of p_i in the training process. The idle neurons are assigned with a large probability, whereas the weights preserving the historical knowledge are assigned with a small probability. Thus, the idle neurons have a large chance to be trained and capture the features of incremental data in the dropout training phase. At the same time, the weights that preserve the historical knowledge have small chances to be updated.

In detail, the norm of the $(N-1)$-order weight tensor of each neuron is computed as follows:

$$\|\mathrm{w}\| = w \otimes w = \sum_{i_1=1}^{I_1} \cdots \sum_{i_{n-1}}^{I_{N-1}} w_{i_1 i_2 \ldots i_{N-1}}^2. \tag{12}$$

Then, the double surfaces of two concentric balls segment the weight space into three subspaces in the following form:

$$\begin{aligned} W_A &: 0 < \|\mathrm{w}\| < R_1 \\ W_B &: r_1 \leq \|\mathrm{w}\| < R_2 \\ W_C &: \|\mathrm{w}\| \geq R_2. \end{aligned} \tag{13}$$

The neurons of each subspace play a different role in the learning features. Specifically, the neuron of the subspace W_A has little effect on the feature learning, which can be ignored [26]. The neuron of the subspace W_B preserves most of the knowledge. And the neuron of subspace W_C processes the information on special examples, which often causes overfitting [23], [24]. Thus, the knowledge of the historical data is mainly stored in the neurons of the subspace W_B. In other words, the network ability to learn new data depends on the weight of subspace W_A. To adapt the new data and preserve much the historical knowledge as much, three types of neurons should be trained in a concerted way: the neurons of W_A should be trained by the new data with a higher probability, the neurons of the subspace W_B should have a low chance to be updated, and the weight of subspace W_C be mapped to the subspace W_B by the max-norm.

To achieve this concerted method, a dropout training scheme is designed as follows.

Step 1: A probability p_i $\{\mathrm{i} = 1, 2, 3\}$ $(p_1 > p_3 > p_2 \geq 0)$ is assigned to each neuron of the tensor fully connected layer. The neuron of the subspace W_A, W_B, and W_C is given probabilities p_1, p_2, and p_3 , respectively. And a mask tensor M full of 1's and 0's is produced.

Step 2: The parameter is updated by the initial parameter increment in the following form:

$$W_{\text{post}} = W + \Delta W * M. \tag{14}$$

The initial weight increment ΔW is used one time in the whole training process.

Step 3: The output of the tensor fully connected layer is expressed as follows:

$$O = f\left(W_{\text{post}} \odot X \otimes \mathrm{M} + b\right). \tag{15}$$

The backpropagation of the incremental training is computed as the following three steps.

Step 1: For each neuron of the output layer l, compute the $\Delta\delta^{\mathrm{l}}$ in the following form:

$$\begin{aligned} \Delta\delta^{(l)} &= \frac{\partial J_{\text{DCCM}}}{\partial z^{(l)}} = \frac{\partial}{\partial z^{(l)}}\frac{1}{2}\left(a^{(l)} - y^{(l)}\right)^2 \\ &= \left(a^{(l)} - y^{(l)}\right) \otimes f'\left(z^{(l)}\right) \end{aligned} \tag{16}$$

where $\otimes$ is the elementwise product.

Step 2: For the neuron of other tensor fully connected layers, compute the $\Delta\delta^{\mathrm{l}}$ as follows:

$$\Delta\delta^l = \left(W^l\right)^T \odot \delta^{l+1} \otimes f'\left(z^{(l)}\right) \otimes M. \tag{17}$$

Step 3: Compute the ΔW and Δb of each tensor fully connected layer as follows:

$$\Delta W^l = M^l \otimes A \odot \delta_E^{(l+1)}, \Delta b^l = \delta^{(l+1)} \tag{18}$$

where $\delta_E^{(l+1)}$ is the extension of the $\delta^{(l+1)}$ with each element extended into a tensor that has the same order and dimension with the activation tensor A.

In the fine tuning, DCCM without the dropout is retrained on the randomly sampled data from the whole data using Algorithm 1. The complexity of DCCM is $O\left(h^N + s^N + J^N\right)$, whereas that of PIL is $O\left(J^N\right)$, since PIL only trains the tensor fully connected layer [7].

B. SIL Algorithm

The SIL algorithm is devised to incrementally learn features from the dynamic big data. In SIL, the rules for the model topology increment, i.e., the fully connected network and the constrained network are designed to transfer the historical knowledge from the previous model. Furthermore, the dropout is extended into the tensor fully connected to improve the robustness of the incremental model.

Algorithm 2: Parameter-Incremental Learning Algorithm.

Input: Tensor example $\{(X^{(i)}, Y^{(i)})\}$, $step_{\max}$, Learning rate η, $threshold$
Output: Updated parameters $\theta = W, b$
for $step = 1, 2 \cdots step_{\max}$ **do**
Produce the mask tensor $M \sim Bernoulli\,(p)$;
Extract features and compute activations:
Tensor convolution $Map_c = f\,(X * K + b)$;
Tensor pooling $Map_s = f\,(\beta \cdot pool\,(Map_c) + b)$;
Map features and compute outputs
$H = f\,(W \odot Map_s + b)$;
Initialize the parameter increment:
Compute the errors of outputs $\Delta Y = H_\theta\,(X) - Y$;
Compute differentiate loss $\Delta\delta_\theta$ with respect to $\theta = W_f, b_f$;
Compute parameter increment $\Delta\theta = -\mu\frac{\partial H_\theta(x)}{\partial\theta}\Delta y$;
Incremental training:
Update the parameter $\theta = \theta + \Delta\theta \otimes M$
Map features and compute outputs with dropout
$H = f\,(W \odot (Map_s \otimes M) + b)$;
Backprogation Process:
Differentiate loss $\Delta\delta_\theta$ with respect to $\theta = W, b$;
Update tensor fully connected layer
$W_f = W_f - \eta\nabla\delta_{\theta W_f}$;
$b_f = b_f - \eta\nabla\delta_{\theta b_f}$;
end for
Fine-tuning training:
Sample a subset S from the all data;
Use S to re-train the model without dropout on Algorithm 1;

1) Increment of the Fully Connected Network: As shown in Fig. 1(c), a tensor fully connected tensor network is stacked by several tensor fully connected layers. Due to the definition of the multidot product, the structure of the tensor fully connected network limits the number of neural nodes added into layers at a time. To keep the structure integrality of the tensor fully connected network, the virtual nodes are introduced by setting the weight of virtual nodes to zeros, making virtual nodes absent from the computation processes.

Taking a two-layer tensor fully connected network for example, it is the simplest form which only includes the input and output layers. Specifically, the N-order tensor $X \in R^{I_1 \times I_2 \cdots I_N}$ represents the input feature, the N-order tensor $H \in R^{J_1 \times J_2 \cdots J_N}$ denotes the output feature, and the parameters (w, b) are expressed in the following form:

$$\begin{aligned} &w \in R^{\alpha \times I_1 \times I_2 \cdots I_N}, (\alpha = J_1 \times J_2 \cdots J_N) \\ &b \in R^{J_1 \times J_2 \cdots J_N}. \end{aligned} \tag{19}$$

When M new neural nodes are to be added in the input layer of this network at a time, N virtual nodes are incorporated into the input feature layer, simultaneously, to keep the structure of the network. And N is computed by the following form:

$$\begin{aligned} N &= L \times I_2 \times I_3 \cdots I_N - M \\ L &= \lceil M \div (I_2 \times I_3 \cdots I_N) \rceil \end{aligned} \tag{20}$$

where $\lceil\rceil$ denotes the top integral function. Thus, the parameters of the current network are modified as

$$\begin{aligned} &w \in R^{\alpha \times (I_1 + L) \times I_2 \cdots I_N}, (\alpha = J_1 \times J_2 \cdots J_N) \\ &b \in R^{J_1 \times J_2 \cdots J_N}. \end{aligned} \tag{21}$$

Similarly, M nodes join into the output layer, thus the parameters are updated as follows:

$$\begin{aligned} &w \in R^{\alpha \times I_1 \times I_2 \cdots I_N}, (\alpha = (J_1 + L) \times J_2 \cdots J_N) \\ &b \in R^{(J_1 + L) \times J_2 \cdots J_N}. \end{aligned} \tag{22}$$

More generally, M_1 nodes and M_2 nodes are added the input and output layer at a time, respectively. The parameters of the updated network are expressed in the following form:

$$\begin{aligned} &w \in R^{\alpha \times (I_1 + L_1) \times I_2 \cdots I_N}, (\alpha = (J_1 + L_2) \times J_2 \cdots J_N) \\ &b \in R^{(J_1 + L_2) \times J_2 \cdots J_N}. \end{aligned} \tag{23}$$

After the topology increment, each element of the prior weight tensor is copied to the new one with the same indexes, the weights of virtual nodes are always set to zeros, and the weights of remaining nodes are randomly initialized by the normal distribution, which are close to the zero.

2) Increment of the Constrained Network: The incremental topology of the constrained network is mainly expressed by the increment of the tensor convolutional and pooling kernels, since the existing kernel cannot capture the unknown features contained in the new data by the enlarging the kernel dimension without losing obtained knowledge. For example, a basic constrained network is shown in Fig. 2(b), in which $X \in R^{H_1 \times H_2 \cdots H_N}$ represents the input feature, $k_{ci} \in R^{I_1 \times I_2 \cdots I_N}$ $(i = 1, 2, \ldots, s)$ denotes the ith tensor convolutional subkernel, and $k_{pi} \in R^{J_1 \times J_2 \cdots J_N}$ is the ith tensor pooling subkernel. The

parameters of the current topology are expressed as:

$$K_c \in R^{s\times I_1\times I_2\cdots I_N}$$
$$K_p \in R^{s\times J_1\times J_2\cdots J_N}$$
$$O_c \in R^{s\times(H_1-I_1+1)\times(H_1-I_1+1)\cdots(H_N-I_N+1)}$$
$$O_p \in R^{s\times\frac{H_1-I_1+1}{J_1}\times\frac{H_2-I_2+1}{J_2}\cdots\frac{H_N-I_N+1}{J_N}} \quad (24)$$

where the $N+1$-order tensor K_c is the total convolutional kernel, and K_p is the $N+1$-order tensor pooling kernel.

Keeping the input feature layer of the constrained network unchanged, a new N-order tensor k_{ci} is incorporated into the tensor convolutional layer as a subkernel, which results in a new output feature map with an element added in the bias. Simultaneously, a new tensor pooling kernel is added to map the new output feature map to the output of the tensor pooling layer. And the parameters of the current topology are in the following form:

$$K_c \in R^{(s+1)\times I_1\times I_2\cdots I_N}$$
$$K_p \in R^{(s+1)\times J_1\times J_2\cdots J_N}$$
$$O_c \in R^{(s+1)\times(H_1-I_1+1)\times(H_1-I_1+1)\cdots(H_N-I_N+1)}$$
$$O_p \in R^{(s+1)\times\frac{H_1-I_1+1}{J_1}\times\frac{H_2-I_2+1}{J_2}\cdots\frac{H_N-I_N+1}{J_N}}. \quad (25)$$

More generally, M new tensor convolutional subkernels join into the convolutional layer with M elements added in the bias. The parameters of the constrained network have to be modified as follow:

$$K_c \in R^{(s+M)\times I_1\times I_2\cdots I_N}$$
$$K_p \in R^{(s+M)\times J_1\times J_2\cdots J_N}$$
$$O_c \in R^{(s+M)\times(H_1-I_1+1)\times(H_1-I_1+1)\cdots(H_N-I_N+1)}$$
$$O_p \in R^{(s+M)\times\frac{H_1-I_1+1}{J_1}\times\frac{H_2-I_2+1}{J_2}\cdots\frac{H_N-I_N+1}{J_N}}. \quad (26)$$

After the increment of the topology, the new tensor convolutional and pooling kernels are initialized randomly as the small numbers that are also subjected to the normal distribution and close to zero.

Finally, the dropout HBP is designed to efficiently train the structure-incremental DCCM, which can further improve the robustness of the model. The details are outlined in Algorithm 3. The total complexity of SIL is $O\left(h^N + s^N + J^N\right)$ with the constant $N \le 4$, since it trains the whole parameters of the model in the same way with Algorithm 1 [5].

Algorithm 3: Structure-Incremental Learning Algorithm.

Input: Tensor example $\{(X^{(i)}, Y^{(i)})\}$, $step_{\max}$, Learning rate η, $threshold$

Output: Updated parameters $\theta = W, b, K, \beta$

Update the topology of parameter:

$K_c \in R^{(s+M)\times I_1\times I_2\cdots I_N}$
$K_p \in R^{(s+M)\times J_1\times J_2\cdots J_N}$
$O_c \in R^{(s+M)\times(H_1-I_1+1)\times(H_2-I_2+1)\cdots\ (H_2-I_2+1)}$;
$O_p \in R^{(s+M)\times\frac{J_1}{n}\times\frac{J_2}{n}\cdots\frac{J_N}{n}}$
$w \in R^{\alpha\times(I_1+L_1)\times I_2\cdots I_N}$
$b \in R^{(J_1+L_2)\times J_2\cdots J_N}$;

Produce the mask tensor $M_s \sim Bernoulli\,(p)$;

for $step = 1, 2\cdots step_{\max}$ **do**

Extract features and compute activations:

Tensor convolution $Map_c = f\,(X * K + b)$;
Tensor pooling $Map_s = f\,(\beta \cdot pool\,(Map_c) + b)$;
Map features and compute outputs
$H = f\,(W \odot (Map_s \otimes M_s) + b)$;

Backprogation Process:

Differentiate loss $\Delta\delta_\theta$ with respect to $\theta = W, b, K, \beta$;
Update tensor fully connected layer
$W_f = W_f - \eta\nabla\delta_{\theta W_f}$;
Update tensor pooling layer $W_s = W_s - \eta\nabla\delta_{\theta W_s}$;
Update tensor pooling layer $W_c = W_c - \eta\nabla\delta_{\theta W_c}$;

end for

Fine-tuning training:

Sample a subset S from the all data;
Use S to re-train the model without updating structure;

IV. Experiments

In this section, extensive experiments are conducted on two typical datasets: CIFAR [27] and CUAVE [7] to verify the performance of the incremental DCCM by comparing with the parameter-incremental computation algorithm (PIC) [25] and the HBP [7] in classification accuracy and the training time. In the experiment, PIC is adopted in the fully connected layers of DCCM to construct the compared incremental models for assessing the effectiveness of the parameter-incremental deep convolutional computation model (PIDCCM). HBP is proposed to train DCCM to capture the heterogeneous data, to validate the effectiveness of the structure-incremental deep convolutional model (SIDCCM). All the experiments are carried out on the server with 10-core, 20-thread, 1.9-GHz Intel Xeon E7-4800 CPU, 64-GB memory, and 1-TB driver. To fully evaluate the performance of the proposed incremental algorithms, the test dataset is equally divided into five subsets, labeled by the experiment number. Each model is conducted on five subset test datasets ten times. The average results with the corresponding experiment number are demonstrated in figures.

A. Experiments on CIFAR Dataset

We first verify the performance of PIDCCM on the CIFAR that is a representative image dataset used to evaluate the algorithms of the image recognition and deep learning. CIFAR contains 60 000 images fallen into 100 classes equally. Furthermore, the 100 classes can be grouped into 20 super classes. In our experiments, each image is denoted by a three-order tensor $R^{32\times32\times3}$. The first two orders represent the height and weight, whereas the third order stores the color channels. Since the efficiency of PIDCCM is verified in terms of adaption, preservation, and convergence speed, CIFAR is divided into four subsets as follows.

1) Subset-1 is the training dataset including four subclasses of each superclass.

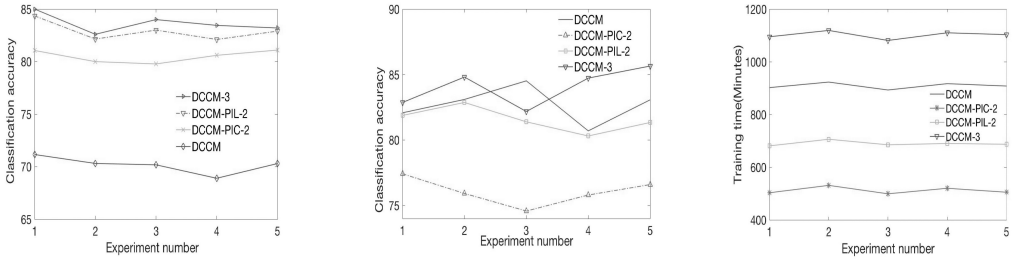


Fig. 3. Results of the parameter-incremental deep convolutional computation model on the CIFAR dataset. (a) Adaption results. (b) Preservation results. (c) Training time.

TABLE I
SHORT NAME OF THE MODELS USED IN CIFAR DATASET

Model name	Description
DCCM	DCCM trained by HBP on the subset-1
DCCM-PIC-2	DCCM trained by PIC on the subset-2
DCCM-PIL-2	DCCM trained by PIL on the subset-2 and subset-1
DCCM-3	DCCM trained by HBP on the whole training data

2) Subset-2 is the incremental training dataset including the remaining subclass of each superclass.
3) Subset-3 is the preservation test dataset including the same subclasses with subset-1.
4) Subset-4 is the adaption test dataset including the same subclass with subset-2.

To evaluate the performance of PIDCCM, DCCM with five constrained layers and three fully connected layers is trained for the above-mentioned datasets. Four parameter sets can be obtained, which are listed in Table I.

To evaluate the adaption of PIDCCM, the trained models are carried out the subset-4. And the results are demonstrated in Fig. 3(a).

In Fig. 3(a), the results produced by DCCM is significantly lower than those of other models, because this model is only trained on the subset-1 and it does not well learn the new patterns in the subset-2, meaning that the traditional static model with the history knowledge cannot distinguish the new input. Different from the traditional static model, the incremental models update its parameters in an incremental way on the subset-2. Thus, the incremental models achieve the high classification accuracy on the subset-4, indicating that PIL and PIC methods perform effectively in terms of adaption. Furthermore, the results produced by the PIL model is higher than those produced by the PIC model, since the approximation of the loss function between the neuron-level and global-level preservation with the Taylor approximation produces some deviations to the real features of the new input. Moreover, the improved dropout of PIL can make the free subarchitecture of the model capture the new knowledge. In addition, DCCM-PIL-2 produces similar results with the results of DCCM-3 that is trained on the whole dataset by HBP. Such observations demonstrate that our PIL method can perform well for the incremental learning in terms of the adaption.

Fig. 3(b) shows the preservation results produced by those four models on subset-3. There are three observations. First, DCCM-3 yields the slightly higher accuracy than DCCM, since the parameters of DCCM-3 are more robust than those of DCCM. In detail, DCCM-3 is trained on the whole dataset, whereas DCCM is trained only on subset-1, resulting that the parameters of DCCM-3 are generalized better than those of DCCM. Second, the results produced by DCCM-PIL-2 are higher than those of DCCM-PIC-2, because the PIL method can protect the weights contained the historical knowledge by a low updating probability. And the fine-tuning training merges the new and historical knowledge effectively by using a small number of samples. Moreover, the results of DCCM-PIL-2 are slightly lower than DCCM-3, which demonstrates the effectiveness of DCCM-PIL-2 in terms of preservation.

Finally, the convergence time of those models are shown in Fig. 3(c). DCCM represents the average training time of DCCM on subset-1. DCCM-PIC-2 and DCCM-PIL-2 are the average incremental training time, whereas DCCM-3 is the average training time on the whole dataset. From the results, two observations are obtained. First, DCCM-PIC-2 and DCCM-PIL-2 spend less time than DCCM-3 indicating that the incremental methods are more efficient than the completely retrained method on the whole dataset in terms of convergence, because DCCM-3 requires the old data and the new data to train the model. Those incremental methods can preserve the historical knowledge without retraining on the whole data. Second, the PIL method spends a bit more time than the PIC method, since PIL uses the fine-turning training to merge the new and historical knowledge, which needs more training time.

Overall, PIDCCM yields the best performance in terms of adaption and preservation, although it takes a bit more time than other incremental model. Specifically, it cannot achieve the similar adaption classification accuracy with DCCM-3 on the subset-4, but the similar preservation classification accuracy with DCCM on the subset-3, indicating the effectiveness of PIDCCM.

B. Experiments on the CUAVE Dataset

To verify the performance of SIDCCM, we carry out some experiments on the CUAVE dataset that is composed of the digits from 0 to 9 and the frontal face recorded from 36 speakers. In our experiment, the data objects produced by the odd-numbered speakers are used as the training dataset, and the rest of CUAVE are used to evaluate the models. Each audio-visual object is represented as a tensor $R^{128\times128\times128}$. The first two orders contain the information of the lip shape, whereas the audio spectrogram with temporal derivatives comprises the third order. A total of 10% of the initial total number of each hidden layer are added

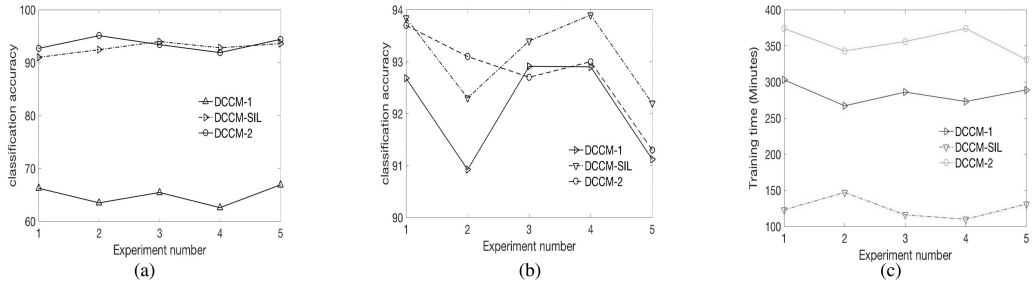


Fig. 4. Results of the structure-incremental deep convolutional computation model on the CUAVE dataset. (a) Adaption results. (b) Preservation results. (c) Training time.

TABLE II
SHORT NAME OF THE MODELS USED IN CUAVE DATASET

Model name	Description
DCCM-1	DCCM trained by HBP on the subset-1
DCCM-SIL	DCCM trained by SIL on the subset-1 subset-2
DCCM-2	DCCM trained by HBP on the subset-1 and subset-2

into the corresponding layer at one time. The classification accuracy is the index adopted to measure the models. Since we also evaluate SIDCCM in terms of adaption, preservation, and convergence time, the CUAVE dataset is divided into four subsets as follows.

1) Subset-1 is the initial training dataset including objects labeled with the digit from 0 to 7.
2) Subset-2 is the incremental dataset including the remaining objects.
3) Subset-3 is the preservation test dataset including objects of the same label subset-1.
4) Subset-4 is the adaption test dataset including objects of the same label subset-2.

To assess the performance of SIDCCM, DCCM with the architecture of seven constrained layers and three fully connected layers is trained, which was validated in [7]. Thus, three parameter sets can be obtained, which are listed in Table II.

The results of SIDCCM are demonstrated in Fig. 4(a), which shows the adaption results. The results of DCCM-SIL are greatly higher than those of DCCM-1, because DCCM-SIL trained by SIL on the incremental training dataset can capture the new features of the new inputs, whereas DCCM-1 cannot deal with features of unseen samples effectively. In addition, DCCM-SIL can produce the similar results with DCCM-2. Those observations indicate the effectiveness of SIDCCM in terms of adaption.

Fig. 4(b) illustrates the preservation results of those models. These three models produce the similar results that are very effective in terms of classification accuracy. And the results of DCCM-3 and DCCM-SIL are slightly higher than DCCM-1 in most cases, since DCCM-3 and DCCM-SIL are trained on the whole dataset, which contributes to a good parameter generalization of the model. Moreover, the DCCM-SIL produces the best classification result at the fourth experiment, since the dropout strategy extended into the tensor space can further generalize the model parameters. Those observations prove the effectiveness of the SIDCCM model in terms of preservation.

Finally, SIDCCM is evaluated in terms of convergence time. And the results are in Fig. 4(c). The training time of DCCM-SIL is greatly less than that of DCCM-2, although both models are trained on the whole data. The DCCM-SIL model can incorporate the historical knowledge by using the sophisticated structure updating rules. In DCCM-SIL, the historical data are used to merge the new and historical knowledge, whereas the whole data are used to train a new ensemble of new parameters in DCCM-2, which costs more time.

The results show that SIDCCM achieves the best performance in terms of adaption, preservation, and convergence time.

V. CONCLUSION

In this paper, an incremental DCCM is designed to learn features of the industrial big data over the tensor space. An important advantage of the proposed model is to implement the incremental learning effectively and efficiently for big data by presenting two algorithms—PIL algorithm and SIL algorithm. The PIL is used to update the parameters for accommodating new arriving objects with the similar distribution. The SIL algorithm transfers the historical knowledge to the new architecture. The dropout technique is adopted to make full use of the idle architectures of the current network in PIL and improve the robustness of the model in SIL. The experimental results demonstrate that the proposed model performs better than the DCCM and the parameter computation models in terms of the adaptation, preservation, and convergence time, proving its potential for the feature learning of the incremental big data. Although the incremental learning method can learn features on big data of high velocity in an incremental fashion, it is still not efficient enough for big data real-time processing due to the large number of parameters needed. To improve the efficiency of the incremental DCCM, the tensor network and decomposition techniques will be adopted to condense the proposed model in our future work.

REFERENCES

[1] C. Chen, H. Xiang, T. Qiu, C. Wang, Y. Zhou, and V. Chang, "A rear-end collision prediction scheme based on deep learning in the Internet of Vehicles," *J. Parallel Distrib. Comput.*, vol. 117, pp. 192–204, 2018, doi: 10.1016/j.jpdc.2017.08.014.

[2] A. A. Moghaddam, A. Seifi, and T. Niknam, "Multi-operation management of a typical micro-grids using Particle Swarm Optimization: A comparative study," *Renew. Sustain. Energy Rev.*, vol. 16, no. 2, pp. 1268–1281, 2012.

[3] M. Z. A. Bhuiyan, G. Wang, and A. Vasilakos, "Local area prediction-based mobile target tracking in wireless sensor networks," *IEEE Trans. Comput.*, vol. 64, no. 7, pp. 1968–1982, Jul. 2015.

[4] X. Liu, R. Deng, K. R. Choo, and Y. Yang, "Privacy-preserving outsourced support vector machine design for secure drug discovery," *IEEE Trans. Cloud Comput.*, 2018. [Online]. Available: https://ieeexplore.ieee.org/document/8281028

[5] G. S. Mahalakshmi, G. Muthuselvi, S. Sendhilkumar, P. Vijayakumar, Y. Zhu, and V. Chang, "Sustainable computing based deep learning framework for writing research manuscripts," *IEEE Trans. Sustain. Comput.*, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8344521

[6] S. Kalsi, H. KaurEmail, and V. Chang, "DNA cryptography and deep learning using genetic algorithm with NW algorithm for key generation," *J. Med. Syst.*, vol. 42, no. 17, pp. 1–12.

[7] P. Li, Z. Chen, L. T. Yang, Q. Zhang, and M. J. Deen, "Deep convolutional computation model for feature learning on big data in Internet of Things," *IEEE Trans. Ind. Inform.*, vol. 14, no. 2, pp. 790–798, Feb. 2018.

[8] Q. Zhang , L. T. Yang, and Z. Chen, "Privacy preserving deep computation model on cloud for big data feature learning," *IEEE Trans. Comput.*, vol. 65, no. 5, pp. 1351–1362, May 2016.

[9] N. Srivastava and R. Salakhutdinov, "Multimodal learning with deep Boltzmann machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 2012, pp. 2222–2230.

[10] X. Wu, X. Zhu and G. Wu, "Data mining with big data," *IEEE Trans. Knowl. Data Eng.*, vol. 26, no. 1, pp. 97–107, Jan. 2014.

[11] M. Z. A. Bhuiyan, G. Wang, J. Cao, and J. Wu, "Deploying wireless sensor networks with fault-tolerance for structural health monitoring," *IEEE Trans. Comput.*, vol. 64, no. 2, pp. 382–395, Feb. 2015.

[12] A. A. Moghaddam, H. Monsef, and A. R. Kian, "Optimal smart home energy management considering energy saving and a comfortable lifestyle," *IEEE Trans. Smart Grid*, vol. 6, no. 1, pp. 324–332, Jan. 2015.

[13] X. Liu, R. Deng, K. R. Choo, Y. Yang, and H. Pang, "Privacy-preserving outsourced calculation toolkit in the cloud," *IEEE Trans. Depend. Secure Comput.*, 2018. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/8318625

[14] B. Krawczyk, L. L. Minku, J. Gama, J. Stefanowski and M. Wozniak, "Ensemble learning for data stream analysis: A survey," *Inf. Fusion*, vol. 47, pp. 132–156, 2017.

[15] R. Elwell and R. Polikar, "Incremental learning of concept drift in nonstationary environments," *IEEE Trans. Neural Netw.*, vol. 22, no. 10, pp. 1517–1531, Oct. 2011.

[16] Y. Sun, K. Tang, L. L. Minku, S. Wang and X. Yao, "Online ensemble learning of data streams with gradually evolved classes," *IEEE Trans. Knowledge Data Eng.*, vol. 28, no. 6, pp. 1532–1545, Jun. 2016.

[17] G. Ditzler, M. Roveri, C. Alippi, and R. Polikar, "Learning in nonstationary environments: A survey," *IEEE Comput. Intell. Mag.*, vol. 10, no. 4, pp. 12–25, Nov. 2015.

[18] L. Fu, H. Hsu, and J. C. Principe, "Incremental backpropagation learning networks," *IEEE Trans. Neural Netw.*, vol. 7, no. 3, pp. 757–761, May 1996.

[19] A. H. L. West and D. Saad, "On-line learning with adaptive backpropagation in two-layer networks," *Phys. Rev. E*, vol. 56, no. 3, pp. 3426–3445, 1997.

[20] R. Polikar, L. Udpa, S. S. Udpa, and V. Honavar, "Learn++: An incremental learning algorithm for supervised neural networks," *IEEE Trans. Syst., Man Cybernet.*, vol. 31, no. 4, pp. 497–508, Nov. 2001.

[21] Q. Zhang, L. T. Yang, Z. Chen, and P. Li, " A survey on deep learning for big data," *Inf. Fusion*, vol. 42, pp. 146–157, 2018.

[22] H. Ning, H. Liu, and L. T. Yang, "Aggregated-proof based hierarchical authentication scheme for the Internet of Things," *IEEE Trans. Parallel Distrib. Syst.*, vol. 26, no. 3, pp. 657–667, Mar. 2015.

[23] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[24] X. Bouthillier, K. Konda, P. Vincent, and R. Memisevic, "Dropout as data augmentation," 2016. [Online]. Available: https://arxiv.org/pdf/1506.08700.pdf

[25] S. Wan and L. E. Banta, "Parameter incremental learning algorithm for neural networks," *IEEE Trans. Neural Netw.*, vol. 17, no. 6, pp. 1424–1438, Nov. 2006.

[26] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," 2016. [Online]. Available: https://arxiv.org/pdf/1602.07360.pdf

[27] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Training very deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2377–2385.

Peng Li received the B.E. degree in electronic and information engineering from Dezhou University, Dezhou, China, in 2012. He is currently working toward the Ph.D. degree in software engineering at Dalian University of Technology, Dalian, China.

His research interests include deep learning and big data.

Zhikui Chen received the B.E. degree in mathematics from Chongqing Normal University, Chongqing, China, in 1990, and the Ph.D. degree in solid mechanics from Chongqing University, Chongqing, China, in 1998.

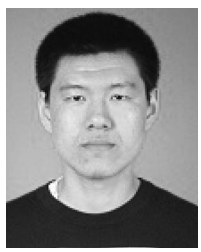
He is currently a Professor with Dalian University of Technology, Dalian, China. His research interests include Internet of Things and big data.

Laurence Tianruo Yang received the B.E. degree in computer science and technology and the B.S. degree in applied physics both from Tsinghua University, Beijing, China, in 1992, and the Ph.D. degree in computer science from the University of Victoria, Victoria, BC, Canada, in 2006.

He is currently a Professor with St. Francis Xavier University, Antigonish, NS, Canada. His research interests include parallel and distributed computing, embedded and ubiquitous/pervasive computing, and big data. His research has been supported by the National Sciences and Engineering Research Council, and the Canada Foundation for Innovation.

Jing Gao received the B.E. degree in computer science and technology and the Ph.D. degree in computer software and theory from Harbin Institute of Technology, Harbin, China, in 2008 and 2015, respectively.

She is currently an Assistant Professor with the School of Software Technology, Dalian University of Technology, Dalian, China. Her current research interests include multimodal data mining and deep learning.



Qingchen Zhang received the Ph.D. degree in software engineering from Dalian University of Technology, Dalian, China, in 2015.

He is currently a Postdoctoral Fellow with St. Francis Xavier University, Antigonish, NS, Canada. His research interests include cloud computing, deep learning, and big data.

M. Jamal Deen received the Ph.D. degree in electrical engineering and applied physics from Case Western Reserve University, Cleveland, OH, USA, in 1985.

His research interests include nano/optoelectronics, nanotechnology, and their emerging applications in health and environment. He is a Distinguished University Professor and Senior Canada Research Chair of Information Technology with McMaster University, Hamilton, ON, Canada.