Recomputation of the dense layers for performance improvement of DCNN

Yimin Yang, *Member, IEEE,* Q. M. Jonathan Wu, *Senior Member, IEEE*, Xiexing Feng, *Graduate Student Member, IEEE,* and Thangarajah Akilan, *Graduate Student Member, IEEE,*

Abstract—radient descent optimization of learning has become a paradigm for training deep convolutional neural networks (DCNN). However, utilizing other learning strategies in the training process of the DCNN has rarely been explored by the deep learning (DL) community. This serves as the motivation to introduce a non-iterative learning strategy to retrain neurons at the top dense or fully connected (FC) layers of DCNN, resulting in, higher performance.radient descent optimization of learning has become a paradigm for training deep convolutional neural networks (DCNN). However, utilizing other learning strategies in the training process of the DCNN has rarely been explored by the deep learning (DL) community. This serves as the motivation to introduce a non-iterative learning strategy to retrain neurons at the top dense or fully connected (FC) layers of DCNN, resulting in, higher performance.G The proposed method exploits the Moore-Penrose Inverse to pull back the current residual error to each FC layer, generating well-generalized features. Further, the weights of each FC layers are recomputed according to the Moore-Penrose Inverse. We evaluate the proposed approach on six most widely accepted object recognition benchmark datasets: Scene-15, CIFAR-10, CIFAR-100, SUN-397, Places365, and ImageNet. The experimental results show that the proposed method obtains improvements over 30 state-of-the-art methods. Interestingly, it also indicates that any DCNN with the proposed method can provide better performance than the same network with its original Backpropagation (BP)-based training.

1 Introduction

The past few years have witnessed the development of deep learning (DL) including auto-encoders, deep convolutional neural networks (DCNN), etc [1], [2], [3], [4], [5], [6], [7], [8]. DL has been around for a long time, since the early works in the 1980s [1], [2], [9], [10], [11], [3]. The *Neocognitron* [9] was probably the first network that deserved a deep structure that incorporated neurophysiological insights. Hinton *et al.* [5], [6], [7] initiated a breakthrough that was used to reduce the dimensionality of data by multilayer neural networks (NN) with the back-propagation (BP). Over many benchmark datasets, recent DL methods including GoogLeNet [12], AlexNet [13], very deep convolutional network [14], 96/160-layer ResNet [15], Network in Networks [16], Google-Inception model [17], and DenseNet [18], have substantially advanced the state-of-the-art accuracies of objection recognition and have become very good at discovering intricate structures in real data. With the increase in NN depth, the richness of the data representation is enhanced, and the generalization performance of the final classifier improves as well. Recent evidence reveals that network depth is crucial, as the classification/recognition results of deeper NN are better than the shallow ones. This can be observed from the DCNNs with a depth of 8-layer of AlexNet [13], 16-layer of VGG [14], 152-layer of ResNet [19], and 264-layer of DenseNet [18] from 2012 to 2017. With the increase in network depth or network architecture optimization, the performance of DCNN methods has been boosted significantly and is therefore applicable to many real-world applications, such as image recognition, semantic, segmentation, etc.

However, the performance improvement through network architecture optimization is approaching its limitation according to the recent results on the ILSVRC competition. For example, compared to the 8-layer AlexNet which won the ILSVRC in 2012, the top-5 accuracy of 19-layer VGG mode, the winner of ILSVRC in 2014, surged from 84.6 % to 92.7 %. However, after 2015, the top-5 accuracy of ILSVRC increased marginally. For example, 152-layer ResNet release in 2016 and 264-layer DenseNet released in 2017 provide 94.3% and 93.8% top-5 error rates respectively on the ImageNet validation set. But more significant was the increase in number of network depth, which added 240 extra layers compared to VGG-19 network. In other words, the number of network depth spiked dramatically around 13 times over the period. Thus naturally leads to the following motivation: ***Can we further improve the performance of the DCNN models by other learning methods?***

Although a lot of research efforts have accomplished architectural improvements in the DCNN, all the present-day DCNN models use the BP as a cornerstone of their end-to-end training. Such iterative training process of the BP suffers from slow convergence, getting trapped in a local minimum and being sensitive to the learning rate configurations. Unlike iterative learning strategy, the non-iterative methods, such as alternating minimization, Moore-Penrose Inverse, QuickNet, random forest, etc., have emerged into the single-layer-based classifiers since a long time [20]. Likewise, the Moore-Penrose Inverse utilized in this paper can be referred to the work of Schmid [20] back from 1992. Authors in [20] mentioned that neuron weights could sometimes be called the Fisher vector and found by solving the linear equations through standard numerical methods, such as BP or the generalized inverse method. Later, in 2004, Huang *et al.* [21] proved that with

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

Y. M. Yang is with the Computer Science Department, Lakehead University P7B 5E1, Thunder Bay, Ontario, Canada.
Q. M. J. Wu, X. Feng, and T. Akilan are with the Department of Electrical and Computer Engineering, University of Windsor N9B 3P4, Ontario, Canada.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2019.2917685, IEEE Transactions on Pattern Analysis and Machine Intelligence

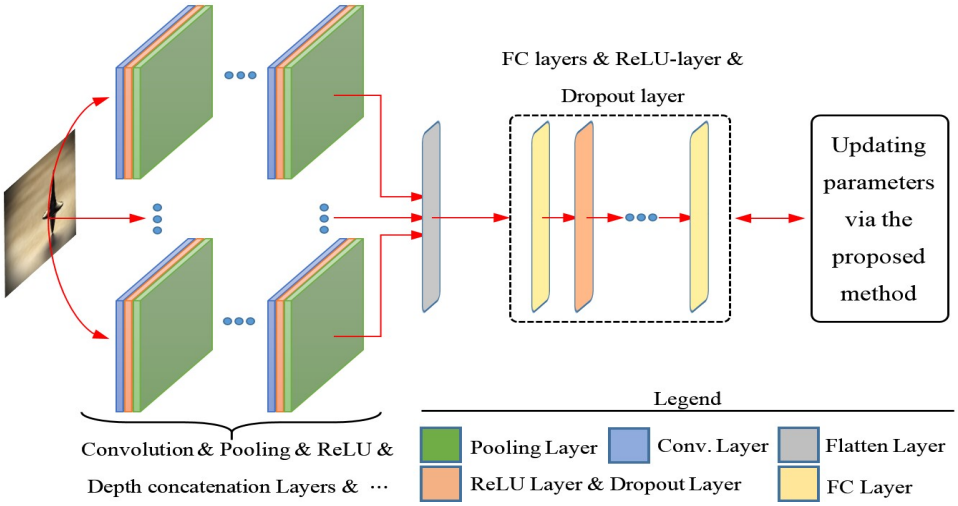


Fig. 1. Schematic Diagram of the proposed method

Moore-Penrose Inverse, single-layer networks are universal approximators even when some neurons in the network are generated randomly. Following this, many researchers propose non-iterative learning-based classifiers for regression and classification problem [22], [23], [24], [25], [26], [27], [28].

However, the non-iterative learning algorithms for training a DCNN model are rarely found. Driven by the confliction, a more detailed motivation arises: ***if the DCNN network structure is maintained constant, could we use a non-iterative learning algorithm to obtain a better performance?*** To address this question, in this paper we try to propose a non-iterative learning strategy to retrain the weights in fully connected layers in DCNN to further boost the generalization performance further. Particularly, this paper contributes the following:

1) Adaptability of DCNN models. In the proposed method, we utilize the Moore-Penrose Inverse strategy to pull back the current residual error **e** of the network to each fully-connected layer one by one, generating a desired output **P** for each fully connected (FC) layer. Then according to the obtained desired output and input features, we use the same strategy to recalculate weights in each FC layer. However, our method only recalculates the parameters in the fully-connected layers but never involves any network structure modification, which makes the proposed method fit for all existing DCNN models.

2) Performance improvements. Experimental results show that a DCNN model using the proposed method always provides better performance than the same DCNN model with its original BP method. For instance, our method achieves categorization accuracy of 94.8% on the Scene15 dataset, which is almost close to **human-level performance**. Furthermore, as Moore-Penrose Inverse method itself does not need any iterative operation; as compared to other DCNN models with iterative methods, the recomputation operation only requires a reasonably extra computational workload (See Fig.12).

2 The Proposed Method

Training a DCNN with BP takes thousands of iterations to adjust the network parameters such as weights and biases of each layer, which would take several hours even in advanced GPUs. Here we show how a traditional DCNN architecture training process can be recalculated with the help of multi-layer neurons that are trained by the Moore-Penrose Inverse strategy. The detailed schematic diagram of the proposed method is shown in Fig 1.

2.1 DCNN with BP-based optimizer

The convolutional (Conv) layer is the cornerstone of the modern deep neural networks (DNN). A $Conv$ operation w.r.t. a filter $\mathbf{a}$, bias b, and an input patch $\mathbf{x}$ is computed as follows:

$$Cov(m,n) = b + \sum_{p=1}^{F-1}\sum_{q=0}^{F-1} \mathbf{a}(p,q) * \mathbf{x}(m+p, n+q), \tag{1}$$

where $*$, F, (m,n), and (p,q) represent the Conv operation, size of the convolutional filter or kernel, the first coordinate of the input image patch, and element indexes of the filter respectively. Hence, the filter weights are updated during training using BP.

The DCNN network is trained using Stochastic Gradient Descent with Momentum (SGDM) optimizer that minimizes the cost function, i.e., difference between the actual output and the predicted output of the network. For instance, if a single layer network with sigmoid classifier is used for a two-class problem (binary classification task), then the log probability of a target class t w.r.t. weights $\mathbf{a}$, an input x, and the prediction y, which can be given by following:

$$\ln p(t|x,\mathbf{a}) = t \ln y + (1-t)\ln(1-y). \tag{2}$$

Eventually, the SGD optimises the parameter $\mathbf{a}$ resulting in maximising the log probability. In other words, it minimises the negative log probability or the cost function E^n of a layer n defined by (3).

$$E^n = -\left(t^n \ln y^n + (1-t^n)\ln(1-y^n)\right). \tag{3}$$

Then, the gradient descent algorithm computes the derivative of the loss with respect the weight $\frac{\partial E}{\partial a}$ noted as $\nabla E(\mathbf{a})$ to update the weight as

$$\mathbf{a}_{l+1} = \mathbf{a}_l - \mu \nabla E(\mathbf{a}_l) \tag{4}$$

where l stands for the iteration step and μ is the learning rate. The SGD will repeat the process until it converges to a minimum solution ($\nabla E(\mathbf{a}_l) \approx 0$).

The proposed pipeline also utilizes the SGDM similar to the traditional CNN methods to update the parameters. However, the weights of the FC layers are recomputed as described in Section 2.2 to improve the training process.

2.2 DCNN with the proposed method

2.2.1 Notations

All the notations used in the paper are shown in Table I.

TABLE 1
Notations used in the paper

Notation	Definition
$\mathbf{a}^n$	the parameters/weights in nth FC layer
μ	learning rate
F	dropout rate
$\mathbf{H}^n$	input features of nth FC layer
$\mathbf{e}^n$	current output error of nth FC layer
$\mathbf{P}^n$	desired output adjustment of nth FC layer
$\mathbf{o}$	output of the last FC layer
$\mathbf{x}$	input data
$\mathbf{y}$	desired output data of the network
C	coefficient parameter
$\mathbf{I}$	identity matrix

2.2.2 Retraining FC layers

Consider a problem of mapping inputs $\mathbf{x}$ to output $\mathbf{y}$ with a function $\mathbf{f}(\mathbf{x})$ given a dataset ($\mathbf{x}$,$\mathbf{y}$). Similar to [26], we define the nested objective function to learn the function $f(\mathbf{x})$ with n hidden layers, as shown that in Fig.2 (to simplify notation, we ignore bias parameters).

$$\begin{aligned} \mathbf{E} &= \frac{1}{2}\sum \|\mathbf{y}_n - f(\mathbf{x},\mathbf{a})\|^2 \\ f(\mathbf{x},\mathbf{a}) &= f_n(\cdots f_2(f_1(\mathbf{x},\mathbf{a}^1),\mathbf{a}^2)\cdots,\mathbf{a}^n) \end{aligned} \tag{5}$$

where each layer function has the form $f_n(\mathbf{x},\mathbf{a}) = g(\mathbf{x},\mathbf{a})$, i.e., a linear mapping followed by a squashing nonlinearity ($g(\cdot)$ applies a scalar function, such as the sigmoid function, sine/cosine function, etc.) But because FC layers in DCNN has no activation function, and the inputs come from the flatten layer, we can rewrite the above equation as follows:

$$\begin{aligned} \mathbf{E} &= \frac{1}{2}\|\mathbf{a}\|^2 + \frac{1}{2}\sum \|\mathbf{y} - f(\mathbf{H},\mathbf{a})\|^2 \\ f(\mathbf{H},\mathbf{a}) &= f_n(\mathbf{a}^n \cdot (\cdots f_2(\mathbf{a}^2 \cdot f_1(\mathbf{a}^1 \cdot \mathbf{H}^0)))) \end{aligned} \tag{6}$$

The basic issue is the n FC layers in the DCNN. The traditional way to minimize equation (6) is by computing the gradient over all the weights of the layers using BP. But we use Moore-Penrose Inverse strategy to pull back the residual error of the network to each FC layer, generating a desired output $\mathbf{P}$ for each FC layer. The detailed retraining method has the following two elements:

1) A step where we fix other input weights and biases of FC layers $(\cdots,\mathbf{a}^{n+1},\mathbf{a}^{n-1},\cdots,\mathbf{a}^1)$, and only update input weights ($\mathbf{a}^n$) and biases with the desired output $\mathbf{P}^n$.
2) A step where we fix all the input weights, and calculate a desired output $\mathbf{P}^{n-1}$ for the nth layer using Moore-Penrose Inverse strategy.

2.2.3 Step 1: Update parameters in each FC layer

For the nth layer of the FC layers in any DCNN model, given N samples that the desired outputs of the nth layer $\mathbf{y}$, current network output $\mathbf{o}$, the weights of the nth layer $\mathbf{a}^n$, the inputs $\mathbf{H}^n$, which is the outputs of nth FC layer, the problem for the proposed method can be formulated as follows:

$$\min_{\mathbf{a}^n} \mathbf{E} = C\frac{1}{2}\sum_{i=1}^{N} \|\mathbf{y}_i - f(\mathbf{H}_i^n,\mathbf{a}^n)\|^2 \tag{7}$$

Because no activation function used in FC layers in DCNN, f is linear function. With alternating minimization, we try to calculate the error-based weight η^n, satisfying $\sum_{i=1}^{N}(\eta^n \cdot \mathbf{H}_i^n) = \mathbf{o} - \mathbf{y}$. Therefore in other words, such error-based weights could satisfy $\sum_{i=1}^{N}(\eta^n + \mathbf{a}^n) \cdot \mathbf{H}_i^n) = \mathbf{y}$. In this study, we try to minimize the training error as well as the norm of error-based weights η^n to avoid the far changes made from the previous weights $\mathbf{a}^n$.

$$\begin{aligned} &\min_{\eta} \mathbf{E} = C\frac{1}{2}\sum_{i=1}^{N} \|\mathbf{e}_i\|^2 + \frac{1}{2}\|\eta\|^2 \\ &\text{Subject to } \sum_{i=1}^{N} \mathbf{H}_i\eta^n = \mathbf{P}_i - \mathbf{e}_i \end{aligned} \tag{8}$$

where $\mathbf{P}$ is the desired adjustments of the outputs, $\mathbf{e}$ is the training error with respect to the input $\mathbf{H}_i^n$. Based on the KKT theorem, training the aforementioned equation is equivalent to solving the following optimization problem.

$$\mathbf{E} = \frac{1}{2}\|\eta^n\|^2 + C\frac{1}{2}\sum_{i=1}^{N}\mathbf{e}_i^2 - \sum_{i=1}^{N}\sum_{j=1}^{m}\alpha_{i,j} f(\mathbf{H}_i^n,\eta^n) - \mathbf{P}_i + \mathbf{e}_i) \tag{9}$$

where $\alpha_{i,j}$ is the Lagrange multiplier. We can have the KKT corresponding optimality conditions as follows:

$$\frac{\partial E}{\partial \mathbf{a}} = 0 \longrightarrow \eta^n = (\mathbf{H}_i^n)^T \cdot \alpha \tag{10a}$$

$$\frac{\partial E}{\partial \mathbf{e}} = 0 \longrightarrow \alpha_i = C\mathbf{e}_i \tag{10b}$$

$$\frac{\partial E}{\partial \alpha} = 0 \longrightarrow \mathbf{H}_i^n\eta^n - \mathbf{P} + \mathbf{e} = 0 \tag{10c}$$

From equation (10a) and equation (10b), we have

$$\eta^n = C(\mathbf{H}_i^n)^T\mathbf{e} \tag{11a}$$

$$\mathbf{e} = \frac{I}{C}((\mathbf{H}_i^n)^T)^{-1}\eta^n \tag{11b}$$

where $\mathbf{H}^{-1}$ is Moore-Penrose inverse of $\mathbf{H}$. From (10c), we have

$$\begin{aligned} \mathbf{H}_i^n\eta^n - \mathbf{P} + \frac{I}{C}((\mathbf{H}_i^n)^T)^{-1}\eta^n &= 0 \\ \mathbf{H}_i^n\eta^n + \frac{I}{C}((\mathbf{H}_i^n)^T)^{-1}\eta^n &= \mathbf{P} \\ (\mathbf{H}_i^n)^T(\mathbf{H}_i^n + \frac{I}{C}((\mathbf{H}_i^n)^T)^{-1})\eta^n &= \mathbf{H}^T\mathbf{P} \\ \left((\mathbf{H}_i^n)^T(\mathbf{H}_i^n) + \frac{I}{C}\right)\eta^n &= \mathbf{H}^T\mathbf{P} \\ \eta^n = \left((\mathbf{H}_i^n)^T(\mathbf{H}_i^n) + \frac{I}{C}\right)^{-1} &\cdot \mathbf{H}^T\mathbf{P} \end{aligned} \tag{12}$$

Thus the last FC layer (softmax layer) can be updated by

$$\begin{aligned} \mathbf{a}^n &= \mathbf{a}^{n+1} + \mu \cdot \eta^n \\ &= \mathbf{a}^n + \mu \cdot \left(((\mathbf{H}_i^n)^T(\mathbf{H}_i^n) + \frac{I}{C})^{-1} \cdot \mathbf{H}^T\mathbf{P}\right) \end{aligned} \tag{13}$$

where $\mu \in (0,1]$ represent the learning rate to determines how fast weights of the fully connected layers change.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2019.2917685, IEEE Transactions on Pattern Analysis and Machine Intelligence

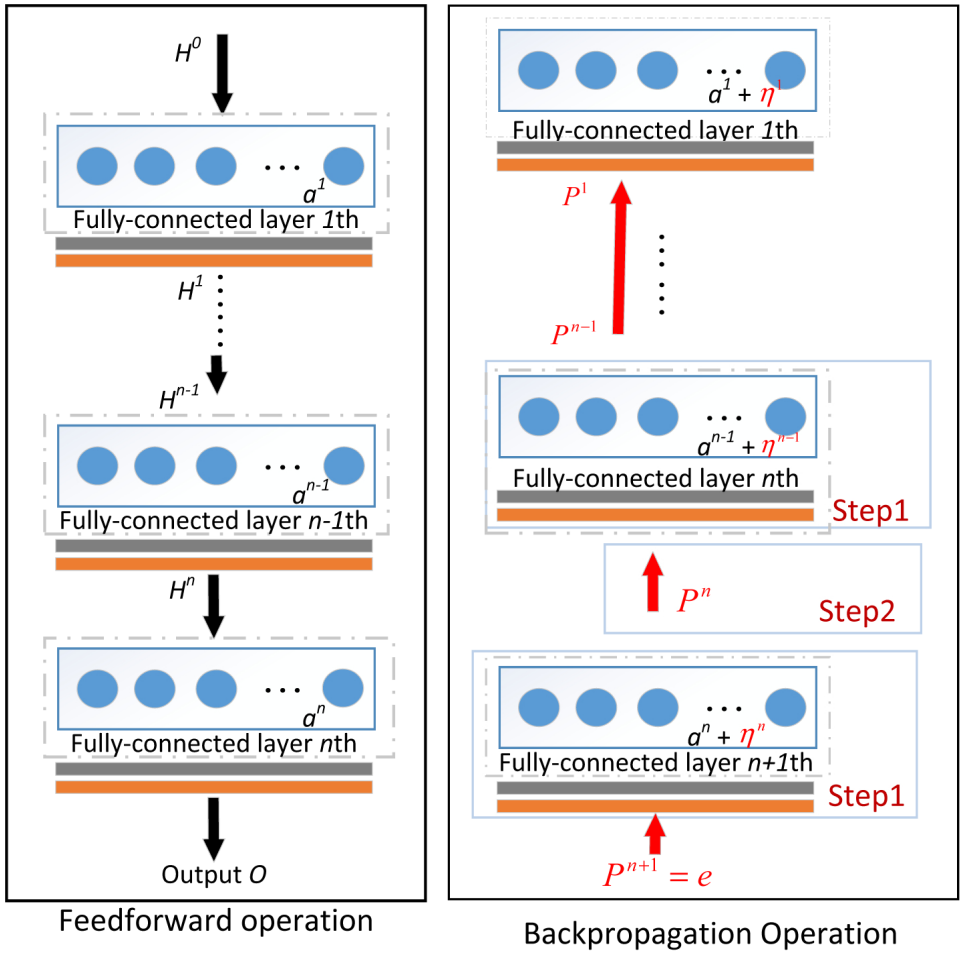


Fig. 2. Learning strategy of the proposed method. ⟵ represents the feedforward operations of the proposed method, while ⟵ represents the error-inverse operations of the proposed method. (a) Step 1: update parameters $(\mathbf{a}^1, \cdots, \mathbf{a}^n)$ in each fully connected layer. (b) Step 2: update the adjustment of the outputs $(\mathbf{P}^1, \cdots, \mathbf{P}^n)$ in each fully connected layer.

As the proposed method is going to retrain the parameters in the FC layer, the learning rate should be introduced for convergence of the entire network. Similar to the learning rate used in the BP part, here, the learning rate μ should be involved to match the weights changes occurring simultaneously in the Conv layers. At each iteration after the BP-based training, we use the Moore-Penrose Inverse to calculate $\boldsymbol{\eta}^n$ and add it to the weights. Without the learning rate μ in previous equation, the weights will change significantly with each iteration, leading to the over-correct, increasing the testing loss. As shown in Fig.3-4, without learning rate μ, the top-1 testing accuracy will actually decrease.

2.2.4 Step 2: Update desired adjustment of the output in each FC layer

As shown in Fig.5, we already have the updated $\mathbf{a}^n$ in the nth FC layer; then, we need to update the $\mathbf{a}^{n-1}$ in the upper layer $(n-1$th) FC layer. According to Subsection 2.3.2, first, the calculate of the desired adjustment of the upper layer $\mathbf{P}^{n-1}$ is necessary. Based on the updated $\mathbf{a}^n$, we can update the current output error of nth FC layer as shown in Fig.5 step 2.1:

$$\mathbf{e}^n = \mathbf{P}^n - \sum_{i=1}^{N} \mathbf{H}^n \cdot \eta^n \tag{14}$$

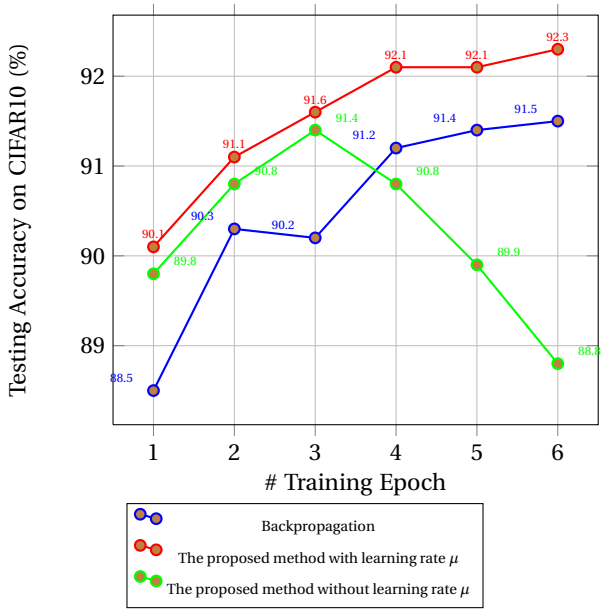


Fig. 3. Testing accuracy performance comparison of AlexNet with or without learning rate used in the proposed method.

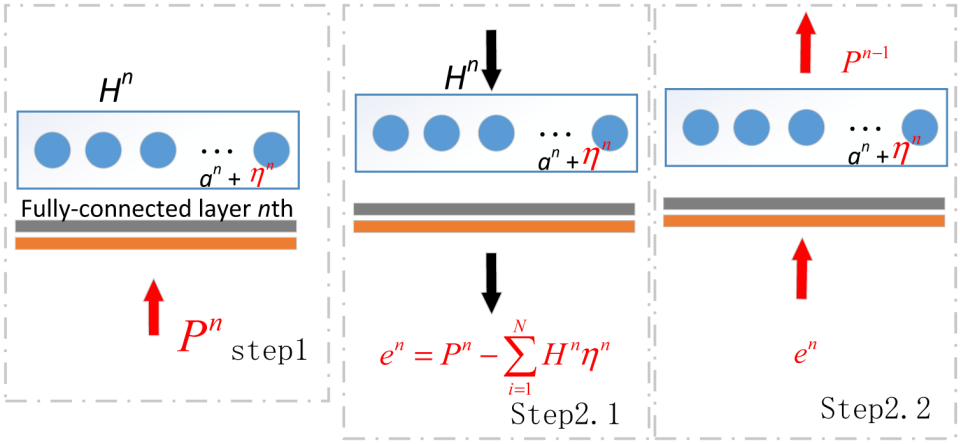


Fig. 5. Illustration of the key re-calculation operations with the proposed method in fully-connected layers. ⟵ represents the feedforward operations of the proposed method, while ⟵ represents the error-inverse operations of the proposed method. (a) Step 1: update parameters ($\mathbf{a}^n$) in the nth fully connected layer. (b) Step 2.1: update the current output error of the nth layer with the updated parameters ($\mathbf{a}^n$). (c) Step 2.2: update the desired adjustment of $n-1$th layer $\mathbf{P}^{n-1}$ with updated error $\mathbf{e}^n$.

Algorithm 1 The proposed method

Initialization: Given a designed DCNN network architecture, input images dataset with labels $\mathbf{x}, \mathbf{y}$, number of FC layers in DCNN n, positive coefficient C, learning rate μ, dropout rate F, batch size value, and maximum training epoch number L_1 and L_2.
for $(j_1 = 1, j_1 <= L_1, j_1 ++)$ **do**
  BP-based training:
  Use SGDM-optimizer to train the DCNN network with only one training epoch.
  while $j_2 < L_2$ **do**
    retraining FC layers:
    1) Extract deep features from the flatten layer ($\mathbf{H}^0$), extract current weights from the DCNN model trained by BP.
    2) Obtain the current output error ($\mathbf{e}^n$) of the last FC layer.
    for $(i = n, i <= 1, i--)$ **do**
      Step1) update the weights $\mathbf{a}^i$ in ith FC layer using equation (18).
      Step2.1) calculate the current error $\mathbf{e}^i$ of the ith FC layer using equation (14).
      Step2.2) calculate the desired adjustment $\mathbf{P}^{i-1}$ of $i-1$th FC layer using equation (16).
    end for
  end while
end for
Obtain feature data $\mathbf{H}_f$.

Because no activation function is used in FC layer, for the output error of the nth layer $\mathbf{e}^n$, we have the following equation:

$$\mathbf{e}^n = \sum_{i=1}^{N} \mathbf{P}^{n-1} \cdot \mathbf{a}^n \tag{15}$$

Based on Moore-Penrose inverse strategy, we can pull the error back across the nth FC layer and the desired adjustment output for $n-1$th layer is as follow:

$$\mathbf{P}^{n-1} = \mathbf{e}^n \cdot (\mathbf{a}^n)^T (\frac{C}{I} + \mathbf{a}^n (\mathbf{a}^n)^T)^{-1} \tag{16}$$

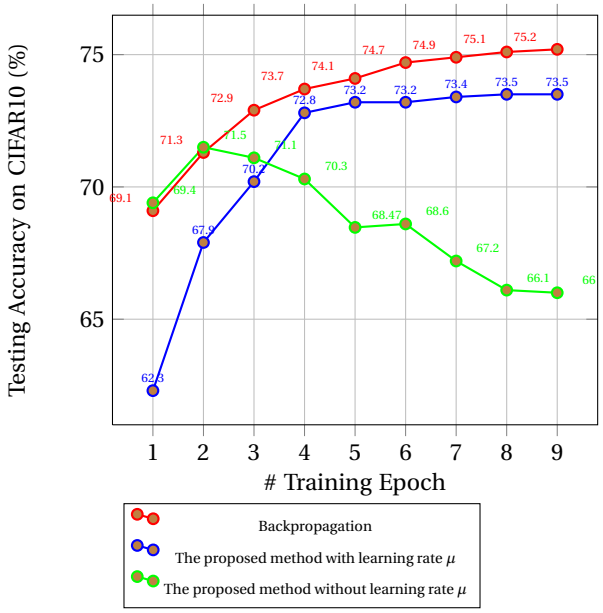


Fig. 4. Testing accuracy performance comparison of AlexNet with or without learning rate used in the proposed method.

where $(\cdot)^{-1}$ is Moore-Penrose inverse. Due to the relu-layer existing, here we give two models for users to select

$$\mathbf{P}^{n-1} = \begin{cases} \mathbf{e}^n \cdot (\mathbf{a}^n)^T (\frac{C}{I} + \mathbf{a}^n (\mathbf{a}^n)^T)^{-1} & \text{model 1} \\ \max(0, \mathbf{e}^n \cdot (\mathbf{a}^n)^T (\frac{C}{I} + \mathbf{a}^n (\mathbf{a}^n)^T)^{-1} & \text{model 2} \end{cases} \tag{17}$$

2.2.5 *Update Parameters through a dropout operation*

Recent studies show dropout layer in DCNN also plays a vital role to counteract over-fitting issue. With a dropout operation, the parameters in each fully-connected layer could be updated by

$$\mathbf{a}^n = F(\mathbf{a}^n + \mu \cdot (\mathbf{P}^n ((\mathbf{H}^n)^T (\frac{C}{I} + \mathbf{H}^n (\mathbf{H}^n)^T)^{-1}))) \tag{18}$$

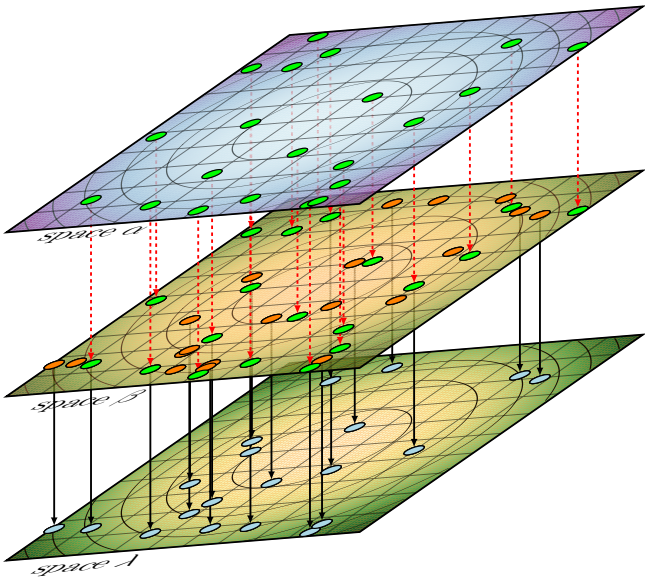


Fig. 6. Dropout strategy for updating parameters. For instance, the initial parameters of 20 neurons are distributed in space α. After updating the neurons using the proposed method, the parameters of 20 neurons are distributed in space β. Finally, if the dropout rate equals 50 %, we only randomly select ten neurons from space α, and the other ten neurons from space β to finalize the 20 recalculated neurons in space λ.

$$\begin{aligned}\mathbf{a}^n &= F(\mathbf{a}^{n+1} + \cdot\mu\cdot\eta^n) \\ &= F\left(\mathbf{a}^n + \mu\cdot\left(((\mathbf{H}_i^n)^T(\mathbf{H}_i^n) + \frac{I}{C})^{-1}\cdot\mathbf{H}^T\mathbf{P}\right)\right)\end{aligned} \tag{19}$$

where F represents a dropout operation to partially update neurons with a random selection way. For example, if we randomly choose a 50% dropout rate, the detailed operation steps can be indicated as the following Fig.6.

2.2.6 The learning steps of the proposed method

Our method can be summarized in Algorithm 1. In order to fully indicate the proposed method, the following is a specific example of using VGG16 for training CIFAR10 as given in Fig.7, which includes three major parts.

- Part I. Loading VGG16 model, and training the VGG16 with SGDM-optimizer. Then extracting weights of the three FC layers as initial weights in retraining part.
- Part II. Extracting deep features from the flatten layer ($\mathbf{H}^0$), and calculating outputs of the three FC layers ($\mathbf{H}^1, \mathbf{H}^2, \mathbf{H}^3$)
- Part III. Calculating desired output adjustment of the three layers ($\mathbf{P}^1, \mathbf{P}^2, \mathbf{P}^3$), as well as the updated weights ($\mathbf{a}^1+\eta^1, \mathbf{a}^2+\eta^2, \mathbf{a}^3+\eta^3$).

In this paper, running Part II and Part III simultaneously indicates that the training epoch of the retraining process equals one. As mentioned in Table 3 and Table 4, we only use one or two training iterations in the experiments. We will release our source codes including the Matlab version and Keras version, after the work has been published.

1. [44] mentioned "This dataset (Scene15) contains only 15 scene categories with a few hundreds images per class, where current classifiers are saturating this dataset nearing human performance at 95 percent".

TABLE 2
Scene-15 classification accuracy for our method against leading alternate approaches without data argumentation

Method	Scene15
Improved classifiers based on NN/SVM/Kernel/KNN	
Kernel codebook [29]	76.6
Object-to-class kernels [30]	88.8
KNN with localized multiple kernel [31]	89.1
Label Consistent K-SVD, Spatial pyramid [32]	92.9
Sparse representation-based methods	
Linear spatial pyramid, sparse coding [33]	80.3
Laplacian sparse coding, feature combination [34]	88.9
Recent feature coding methods	
Feature fusion [35]	71.6
Visual word ambiguity [36]	76.7
Hard assignment [37]	81.4
Soft assignment [38]	82.2
Centrist, Spatial PACT [39]	83.9
Hierarchical networks	
Feature pooling [40]	80.6
Multilayer ELM, SIFT features [25]	82.4
Sparse coding, Max-pooling[41]	84.3
Six-layer deep network, Macro Feature [42]	85.4
Five-layer manifold deep network [43]	86.9
CNN networks with pre-trained features	
Hybrid-CNN, pretrained by Places205 dataset [44]	91.6
AlexNet, pretrained by Places365 dataset [45]	90.0
AlexNet, pretrained by ImageNet dataset	82.4
16-layer VGG, pretrained by ImageNet dataset	88.0
GoogLeNet, pretrained by Places365 dataset [45]	91.2
16-layer VGG, pretrained by Places365 dataset [45]	92.0
16-layer VGG, pretrained by Places365+ImageNet dataset [45]	92.2
Our architecture	
Our method with ImageNet pretrained Alexnet	86.2
Our method with ImageNet pretrained 16-layer VGG	89.8
Our method with Places205 pretrained Alexnet	91.8
Our method with Places205 pretrained 16-layer VGG	**94.8**
Human-level Performance[1] [44]	**95.0**

3 Experimental Verification

3.1 Rival methods

This section aims at examining the performance of our proposed learning method, we test the proposed method on several image datasets. The experiments are conducted in two environments (Matlab 2017b or Keras). For efficient comparisons, we evaluate the **29** state-of-the-art methods arising from the following **three families**:

(1) Recent single layer classifiers include centrist [39], hard/soft assignment [37], sparse coding [33], feature fusion [35], object-to-class kernels [30], multilayer ELM [25], visual word ambiguity [36], K-SVD [32], soft assignment [38], KNN kernel [31], kernel codebook [29], and Laplacian sparse coding [34].

(2) Recent DCNN models include AlexNet [13], VGG-16/19 model [46], Network in Networks [16], Google-Inception model [17], 96/160-layer recurrent convolutional network [15], ResNet50/100, DenseNet [18], ImageNet-pretrained CNN models, Places365-pretrained models [45], hybrid-CNN features [44], hierarchical manifold deep network [43], Multi-column deep network [47], All convolutional net [48], and Deep-supervised Nets [49].

(3) Recent DCNN with feature fusion technologies include multi-layer deep network [42], max-pooling with spatial pyra-

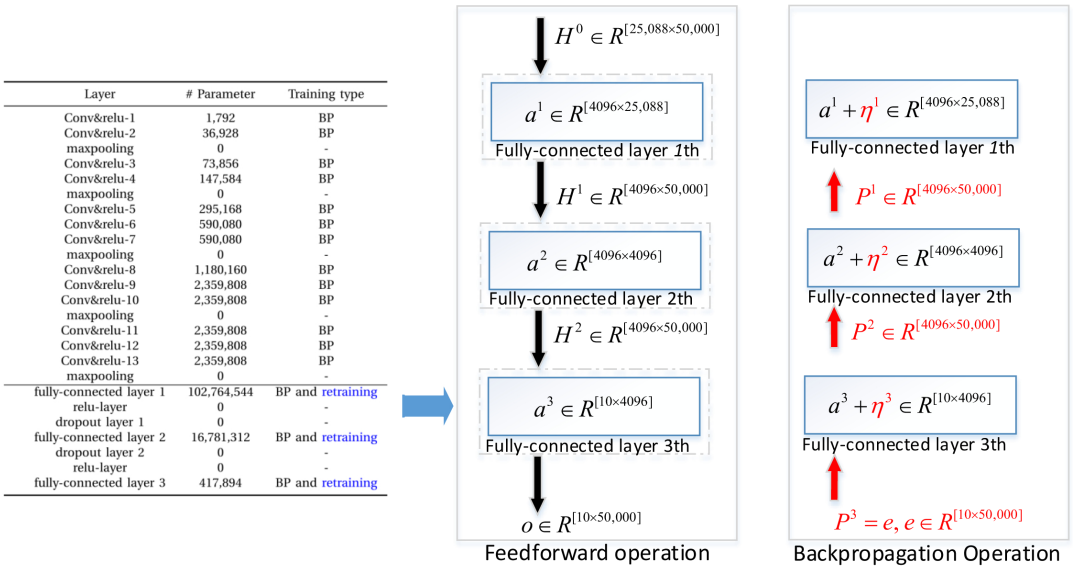


Fig. 7. Retraining fully connected layers of VGG16 with our proposed method (CIFAR10). ⟵ represents the feedforward operations of the proposed method, while ⟵ represents the error-inverse operations of the proposed method.

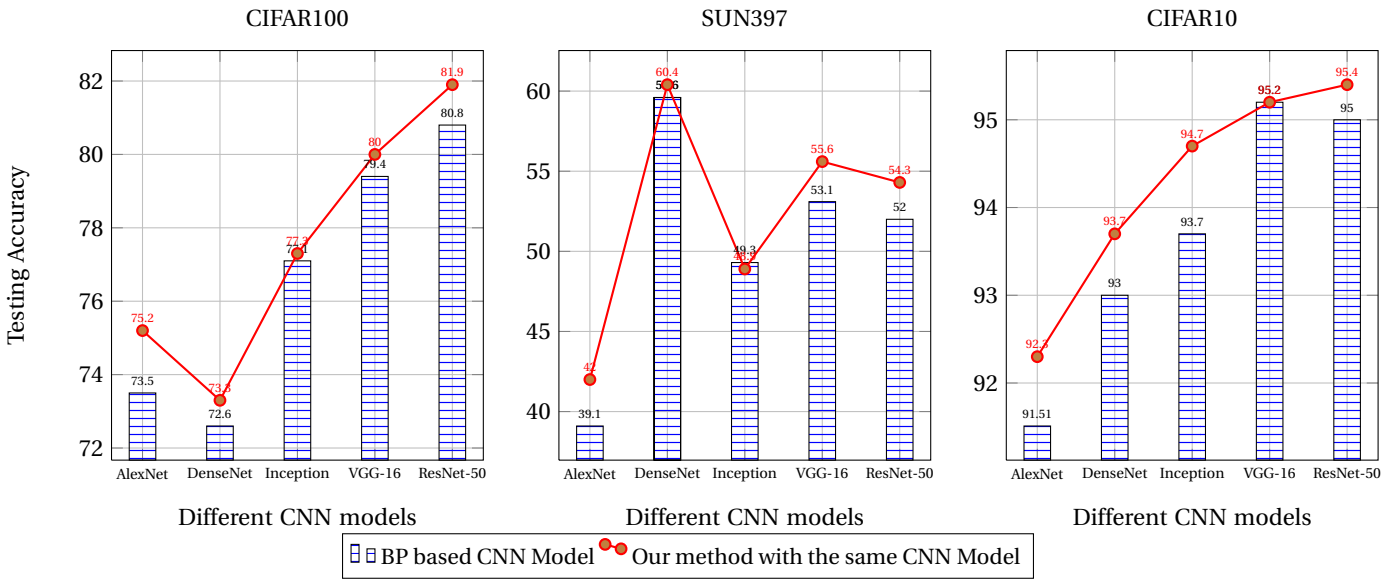


Fig. 8. Top-1 Testing Accuracy of CIFAR10/100 and SUN397: Our method with CNN models vs Original CNN models.

mid features [41], feature pooling [40], deep attention selective networks [50], sumproduct network with deep architecture [51].

3.2 Experimental Environment

3.2.1 Computation resource setting

For this experiment, we select some widely used image datasets to evaluate our method. For completeness, we select six image databases, including one small dataset (Scene15[52]), three medium datasets (CIFAR10/100, SUN397), and two large datasets (Places365 and ImageNet). For the small/medium dataset tests, the experiments are conducted in Matlab 2017b or Keras environment with 32 GB of memory, Geforce 1080 Ti 11GB GPU, and an I7-7700k processor. For the large-scale datasets, a workstation with 128 GB memory, and one/four Geforce 1080 TI GPU(s) is/are used to run the tests. All the results are obtained through three trials. To highlight general trends, we mark all results that outperform the existing state-of-the-art methods in boldface and the best result in blue color.

3.2.2 Experimental Environment

For the Caltech10/100, we use all 50,00 training images for training and the remaining 10,000 images for testing. For Scene15 dataset, we randomly select 100 image per category as training data and use the rest as test data. For SUN397, we randomly split the datasets into a training dataset and a testing dataset, each with 50 images per class. Subsequently, there are 19,850 images for both the training and testing datasets. For Places365 dataset which contains around 1.8 million images comprising 365 unique scene categories, we randomly select 500, 1000, and 1500 images per class from the dataset to produce the training set, and use the validation dataset

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2019.2917685, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 3
Experimental settings under the condition of transfer learning

		AlexNet	VGG16	Inception	ResNet50	DenseNet121[a]
CIFAR10	Initial Learn Rate	1.0^{-3}	1.0^{-3}	1.0^{-2}	1.0^{-2}	-
	# BP-based Batch Size	128	32	32	32	-
	Learn Rate Drop Factor	10 times	10 times	10 times	10 times	-
	Learn Rate Drop Period	3	3	3	3	-
	MaxEpoches	6 epoches	9 epoches	9 epoches	9 epoches	-
	Parameter C	4	6	2	4	-
	Dropout parameter in equation (13) (%)	50	50	50	50	-
	Training loops in retraining process	1	1	2	2	-
	Relu-operation in retraining process	model 1	model 2	[b]	[b]	[b]
CIFAR100	Initial Learn Rate	1.0^{-3}	1.0^{-3}	1.0^{-2}	1.0^{-2}	-
	# BP-based Batch Size	128	32	32	32	-
	Learn Rate Drop Factor	10 times	10 times	10 times	10 times	-
	Learn Rate Drop Period	3	3	3	3	-
	MaxEpoches	9 epoches	9 epoches	9 epoches	9 epoches	-
	Parameter C	4	4	2	4	-
	Dropout parameter in equation (13) (%)	50	50	50	50	-
	Training loops in retraining process	1	1	2	2	-
	Relu-operation in retraining process	model 1	model 1	[b]	[b]	[b]
SUN397	Initial Learn Rate	1.0^{-3}	1.0^{-3}	1.0^{-2}	1.0^{-2}	1.0^{-3}
	# BP-based Batch Size	128	32	32	32	16
	Learn Rate Drop Factor	10 times	10 times	10 times	10 times	10 times
	Learn Rate Drop Period	3	3	2	2	10
	MaxEpoches	9 epoches	9 epoches	6 epoches	6 epoches	30 epoches
	Parameter C	4	4	4	4	1
	Dropout parameter in equation (13) (%)	50	50	50	50	50
	Training loops in retraining process	1	1	2	2	2
	Relu-operation in retraining process	model 2	model 2	[b]	[b]	[b]

[a] The 121-layer DenseNet is running in Keras environment.
[b] No relu-operation because only one FC layer existed in the DCNN model.

TABLE 4
Experimental settings under the condition of training from scratch

		CIFAR10	CIFAR100	ImageNet mini	ImageNet mini[a]	ImageNet	ImageNet[a]
DenseNet	Number of layers	40	40	121	121	121	121
	Initial Learn Rate	0.1	0.1	0.1	0.1	0.1	0.1
	BP-based Batch Size	64	64	20	216	20	216
	Learn Rate Drop Factor	10 times	10 times	10 times	10 times	10 times	10 times
	Learn Rate Drop Period	[150,75,75][b]	[150,75,75][b]	30	30	30	30
	MaxEpoches	300 epoches	300 epoches	90 epoches	90 epoches	90 epoches	90 epoches
	Parameter C	1	1	1	1	1	1
	Dropout parameter in equation (13) (%)	50	50	50	50	50	50
	Training loops in retraining process	2	2	2	2	2	2
	Batch size in retraining process	50,000	50,000	20,000	20,000	20,000	28,800

[a] The 121-layer DenseNet is trained by 4 Geforce 1080 Ti GPUs.
[b] At the beginning 150 epoches (0-150), the learning rate equals 0.1. Then the learning rate will be dropped by 10 times, maintaining 0.01 in the later 75 epoches (151-225). Finally the learning rate will be dropped by 10 times again, maintaining 0.001 in the last 75 epoches (226-300).

(36,500 images) for the testing dataset. No data argumentation algorithms are used for above aformentioned datasets.

For ImageNet which consists around 1.2 million images, we adopt the same data argumentation scheme for training images as in [18], and apply a single-crop with size 224×224 at test time. First we generate an ImageNet Mini dataset by randomly selecting 200 images per category from the dataset. Then we further test our method with full version ImageNet dataset. But due to the huge training time (17 days per training time with 4 GPUs), we only carry out the comparative performance between DenseNet and DenseNet with our method at the current stage.

3.2.3 Validation methods selection

We compare our method with other state-of-the-art methods in two ways. (i) Our method vs. other classifiers; (ii) The DCNN model with our method vs. the same original DCNN model. For the results category (i) (in Subsection 3.3), we try to indicate that the proposed method could generally provide comparable results among the recent well-known image recognition methods. Moreover, for the results category (ii) (in Subsection 3.4), we demonstrate that any DCNN network with our proposed method could generally result in a better generalization performance than that same model with BP-based methods.

3.2.4 CNN models setting

Apart from comparing performance of the rival methods in Subsection 3.3, we further conduct comparison performance experiments in Subsection 3.4 to test the performance gain using the proposed method in the same CNN architecture. To validate these gains, we test our method with fairly known CNN models, including AlexNet, VGG16, ResNet, Inception, and DenseNet40/121.

For the same CNN architecture, the comparison experiments are conducted under the exact same experimental set-

tings such as batch size, learning rate, training epochs, dropout rate, and software/hardware environment. Thus once the performance gap occurs, we can ensure that these performance gain/loss are generated because of different training methods. We consider the following two conditions to conduct these comparison experiments.

1) Transfer Learning. For AlexNet, VGG16, Inception, DenseNet and ResNet-50, we used the existing ImageNet/Places365 pretrained CNN networks and associated the pre-trained weights as initial parameters with an end-to-end training on the target datasets. In this case, the CNN models contain knowledge from both the target datasets, as well as the innate priors (ImageNet or Places365). Due to the pretrained models used as an initial network, the required training epochs could significantly be reduced. The detained information of the CNN settings has been shown in Table 3. For Places365 dataset, the detailed information has been included in Fig.9-10.

2) Training from scratch. To further test our method with CNN models, we also conduct the comparison experiments with DenseNet-40 and DenseNet-121 under the training from scratch condition. In these experiments, we train a DenseNet from scratch with random initialization in pure GPU-based Keras environment. Furthermore, for big dataset ImageNet, it is impossible to extract all the features and to feed the extracted features for retraining stage due to GPU memory limitation. Therefore, batch-by-batch training strategy is used in the retraining stage. The detailed settings for this are shown in Table 4.

As seen in Table 3-4, we set up several special rules for parameter settings to avoid wasting time on finding "the best parameters combination" and involving unfairness performance comparisons.

1) we use similar the same parameter settings such as learn rate, drop factor, maxEpoches, and dropout rate.
2) we set the batch size as large as possible according to the actual amount of GPU memory. For example, to train ImageNet dataset, the maxiumal batch size of DenseNet is 20 with one 11GB memory GPU (1080 TI) because some of the GPU memory is used to store features for retraining process. However, with 4 GPU 1080 Ti GPUs, the maximal batch size of DenseNet could reach up to 216.
3) Similarly (2), we select the initial learn rate as large as possible from the vector [0.1, 0.01, 0.001] if the selected initial learn rate is not for convergence of the CNN network or the training accuracy does not show significant improvement in the first training epoch.
4) For other parameters only existent in retraining part, we also keep these parameters fairly consistent in Keras or Matlab environment.

3.3 Comparison performance of DCNN with our method vs. other 31 state-of-the-art methods

In the Subsection 3.3, we experiment our method on four classic DCNN models including AlexNet, VGG16, 50-layer ResNet and Inception-GoogleNet. The comparison results of Scene15 and CIFAR10/100 are shown in Table 2 and Table 5.

In Table 2, we include results from complex approaches that incorporate many cues and learning-optimal feature combinations and leading alternate approaches. For example, Zhou *et al.* [44] [45] term a new dataset (Places dataset), which contains approximately more than 7 million images from 205 or 365 place categories, making it the largest image database of scenes and places so far. After pre-trained the large-scale dataset, around 92% accuracy is obtained by [44], which approaches almost human-level performance at 95%. However, with the same model used in [44], our result was 94.8%, **which is almost equal to human-level performance**.

In Table 5, we take almost all the recent leading methods as rivals to evaluate our method, including ResNet, Inception, VGG-16, All Convnet Net [48], Densely Net [18], etc. It is easy to notice that our method with AlexNet, GoogleNet, and VGG16 model outperforms the existing state-of-the-art consistently on the three datasets.

3.4 Comparison performance with the same DCNN architecture

In the Subsection, we involve almost some of all the recent well-known DCNN models to show the comparative performance between BP-based learning strategy and our method. Hence except for learning strategy, all the other experimental settings, including the learning rate, the monument rate, the batch-size, **the network architecture**, etc., are maintained constant. Then, we train these DCNN models including AlexNet, VGG-16 Net, Google inception, DenseNet, and the 50-layer ResNet with both BP-based method and our proposed method. The comparison results of the CIFAR10/100, the SUN397, and the Places365 datasets are shown in Table 6-7 and Figure 8-10.

3.4.1 Comparison performance on Scene15, CIFAR10/100, and SUN397

The results shown in Fig.8 and Table 6 indicate that DCNN models with our method significantly boost the learning capacity. As seen in Fig.8, it can be observed that with the same DCNN architecture, our proposed method generally provides better performance than that with BP-method.

We carry out a series of experiments under the two training environmental conditions (training from scratch and training from a pretrained model) to evaluate the comparative performance which has been recorded in Table 6. The advantage is obvious; for CIFAR10/100 and SUN397, the top-1 accuracies are close to 1% to 2% higher than the same DCNN model with BP method. Although the 1 % to 2 % top-1 accuracy boost seems to lead to marginal improvement-it is not easy to obtain these improvements at the current stage. For example, in the well-known DCNN models, the VGG-16 is an ILSVRC winner in the year of 2014 which obtains 95.2 %, 79.4%, and 53.1 % top-1 accuracy on the three databases CIFAR10/100 and SUN397, respectively. However, for CIFAR100 datasets, the 2016 winner ResNet only provides 1.2 % boost. Moreover, for SUN397, ResNet even provides a 1 % lower than VGG-16.

Furthermore, unlike the aforementioned DCNN models which try to obtain performance improvements through structure optimization, our method does not access any network structure re-design task but achieves 1 % to 2 % boost by a non-iterative learning algorithm, which provides another direction in future research to further improve generalization the performance of DCNN models.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2019.2917685, IEEE Transactions on Pattern Analysis and Machine Intelligence

Fig. 9. Top-1 Testing Accuracy of Places365: Our method with AlexNet vs AlexNet.

TABLE 5
Classification accuracy for our method against other leading methods without data augmentation

Method	CIFAR10	CIFAR100
Hierarchical networks		
Sumproduct network with deep architecture [51]	84.1	-
Multi-column deep networks [47]	88.9	-
Deep attention selective networks [50]	90.7	66.3
Deep-supervised Nets [49]	90.2	-
96-layers Recurrent convolutional network [15]	89.7	65.8
160-layers Recurrent convolutional network [15]	91.3	68.3
44-layers ResNet [19]	92.8	-
110-layers ResNet [19]	93.5	-
Network in Networks, pretrained by ImageNet dataset [16]	89.6	64.4
Densely connected convolutional networks [18]	94.8	80.4
All convolutional net, ImageNet-pretrained [48]	92.0	75.6
Our method with CNN models		
Ours with AlexNet	92.3	75.2
Ours with Google Inception	94.7	77.3
Ours with 16-layer VGG	95.2	80.0
Ours with ResNet-50	**95.4**	**81.9**

3.4.2 Comparison performance on Places365 and ImageNet

As far as we know, Places365 and ImageNet could be the largest datasets in image recognition task area. To further test the performance of our method on large-scale datasets, we select the recent large-scale datasets Places365 and ImageNet to evaluate our method. Similarly, we also try to expose recent well-known DCNN models to two training conditions (training from a pretrained model and training from scratch). The experimental results are shown in Fig.9-10 and Table 7.

First, we use ImageNet pretrained DCNN models as initial neuron parameters to evaluate the comparative performance of Places365 dataset. As ImageNet and Places365 are two different databases, the specialty of the units in the object-centric DCNN (ImageNet) and scene-centric DCNN (Places365) yield very different performances of generic visual features on a variety of recognition benchmarks. We report the top-1 accuracy of both our method and other two well-known DCNN models (AlexNet and VGG16) on Places365. As seen in Table 7 and Fig.9-10, the results are quite similar as we have mentioned in Subsection 4.3.1 that our method has significantly beneficial in regarding both learning effectiveness and generalization performance. As shown in Fig.9, the AlexNet with 182,500 training images (1500 training images per class), needs six training epochs to provide 40.13% accuracy, while our method with two training epochs provides 40.49% accuracy. Similar trends can be observed in Fig.8, where the VGG-16 with our proposed method could also provide better performance than that of VGG-16 with pure BP method.

Second, we evaluate the comparative performance by training a scratch DenseNet model with both BP-method and our proposed method on ImageNet. To obtain the top-1 accuracy as fast as possible, in this study, we select 121-layer DenseNet model because DenseNet has a relatively smaller number of parameters than VGG-16, AlexNet, or ResNet models. After around 17 days training, we finally obtain this top-1 accuracy of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2019.2917685, IEEE Transactions on Pattern Analysis and Machine Intelligence

TABLE 6
Top-1 Classification accuracy with both our method and original BP method

Method	CIFAR10	CIFAR100	SUN397
Pretrained AlexNet			
AlexNet [13], ImageNet-pretrained	91.5	73.5	38.5
Ours with ImageNet-pretrained AlexNet	**92.3**	**75.2**	**40.5**
AlexNet, Places365-pretrained	-	-	39.1
Ours with Places365-pretrained AlexNet	-	-	**42.0**
Pretrained VGG-16			
16-VGG, ImageNet-pretrained [46]	95.2	79.4	53.1
Ours with ImageNet-pretrained 16-layer VGG	95.2	**80.0**	**55.6**
Google Inception			
Google-Inception, ImageNet-pretrained [17]	93.7	77.1	**49.3**
Ours with ImageNet-pretrained Google Inception	**94.7**	**77.3**	48.9
DenseNet			
40-layer DenseNet, training from scratch [18]	93.0[18]	72.6[18]	59.6[a]
Ours with 40-layer DenseNet,training from scratch	**93.7**	**73.3**	**60.4**[a]
ResNet-50			
ImageNet-pretrained ResNet-50	95.0	80.8	52.0
Ours with ImageNet-pretrained ResNet-50	**95.4**	**81.9**	**54.3**

[a] 121-layer ImageNet Pretrained DenseNet .

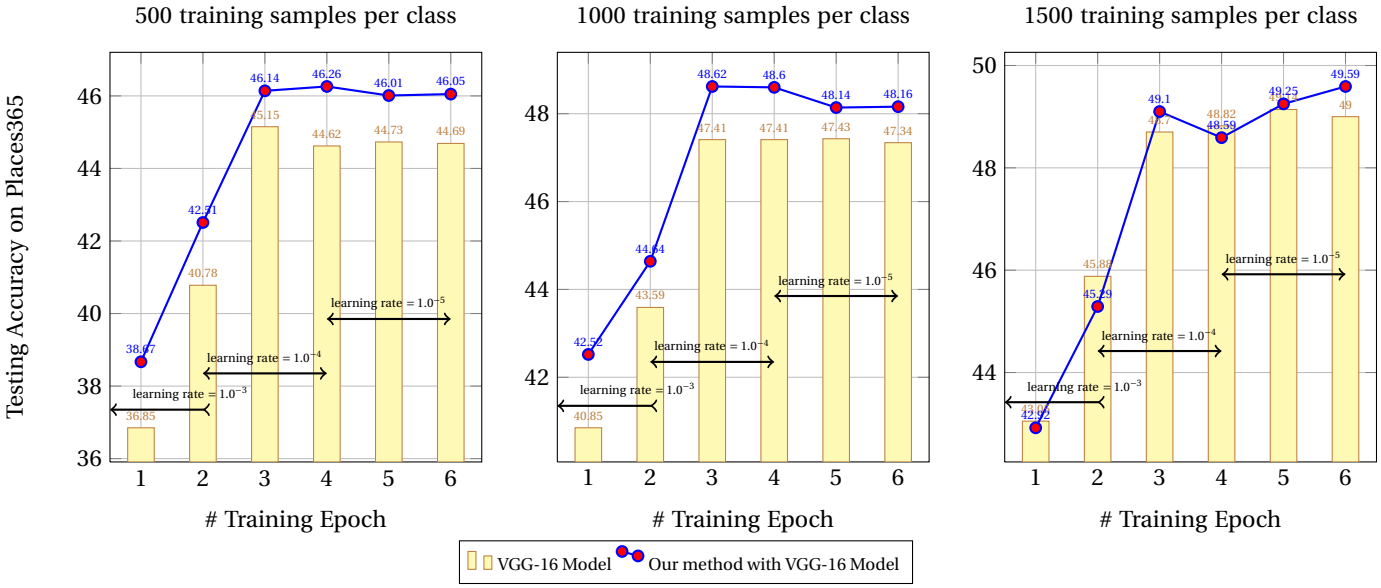


Fig. 10. Top-1 Testing Accuracy of Places365: Our method with VGG16 Vs VGG-16

TABLE 7
Classification accuracy on Scene-centric databases for the deep features of Object-centric databases (ImageNet). All the accuracy is the top-1 accuracy.

Method	Dataset	training image per category	Top-1 Accuracy
Training from the ImageNet pretrained model			
AlexNet, ImageNet-pretrained model	Place365	1500	40.13
Ours, ImageNet-pretrained AlexNet model	Place365	1500	**42.21**
VGG-16, ImageNet-pretrained model	Place365	1500	49.00
Ours, ImageNet-pretrained VGG-16 model	Place365	1500	**49.59**
Training from scratch			
121-layer DenseNet, batch size 20	ImageNet Mini	200	48.83
Ours with 121-layer DenseNet, batch size 20	ImageNet Mini	200	**51.45**
121-layer DenseNet, batch size 216,	ImageNet Mini	200	50.06
Ours with 121-layer DenseNet, batch size 216	ImageNet Mini	200	**51.60**
121-layer DenseNet, batch size 216,	ImageNet	732-1300	69.05
Ours with 121-layer DenseNet, batch size 216,	ImageNet	732-1300	**69.97**

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPAMI.2019.2917685, IEEE Transactions on Pattern Analysis and Machine Intelligence

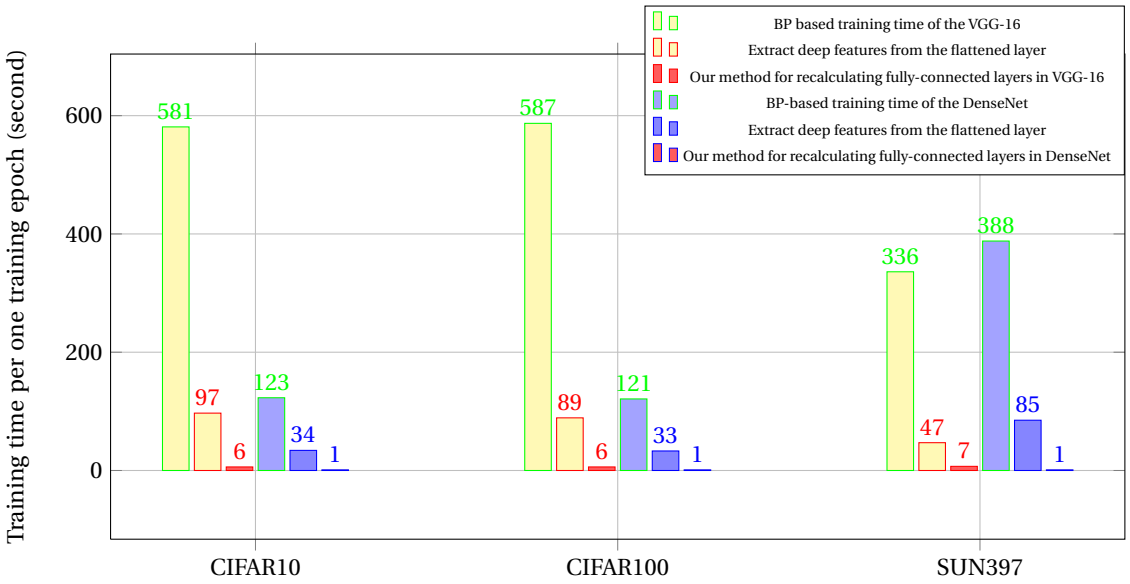


Fig. 11. Training time per training epoch on Keras version environment (one 1080 TI GPU): Our method vs original DCNN models. For DenseNet, we use 40-layer DenseNet on CIFAR10/100, and use 121-layer DenseNet on SUN397.

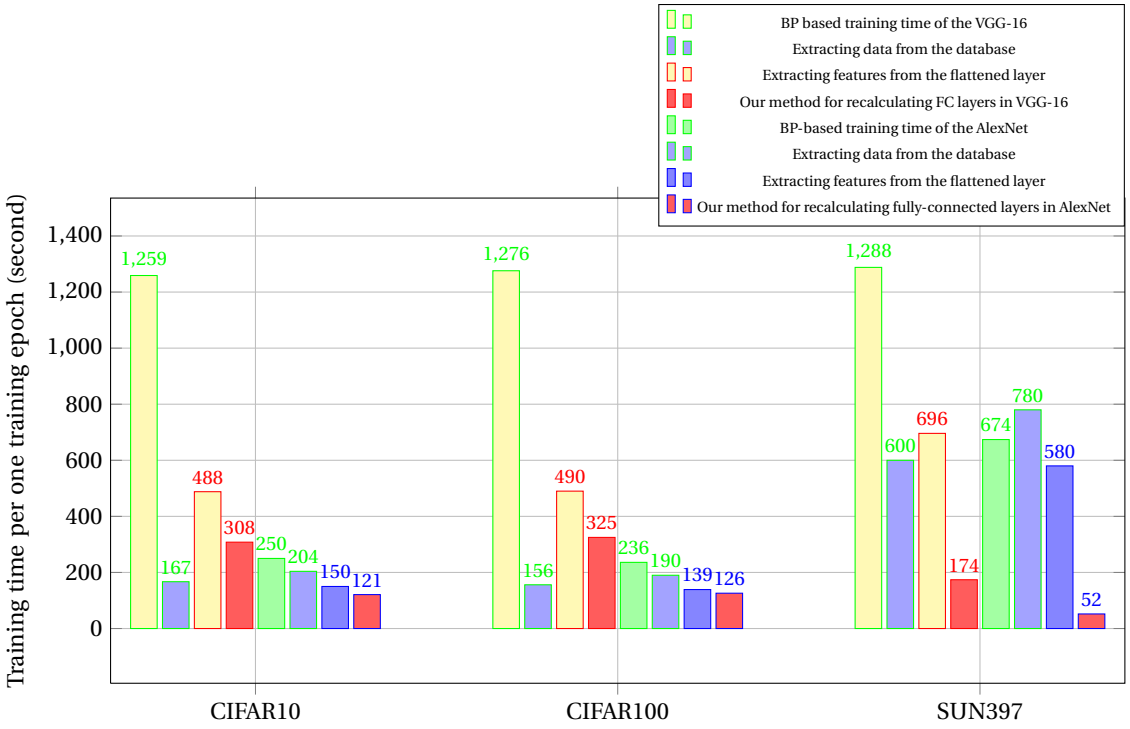


Fig. 12. Training time per training epoch on Matlab 2017b version environment (one 1080 Ti GPU, I7700k CPU): Our method vs original DCNN models. For DenseNet, we use 40-layer DenseNet on CIFAR10/100, and use 121-layer DenseNet on SUN397.

ImageNet. As seen from Table 7, the 121-layer DenseNet using our method provide around 0.4 to 2 percent top-1 accuracy boost than that of the original 121-layer DenseNet.

3.5 Computational Cost

The proposed method is unable to shorten the training speeds in each learning iteration if network structure/size is remained constant. However, our method does not add many computational workloads into the existing training process. As we tested our algorithm under the two environments, we provide the related computational information in the following two parts.

1. Keras environment. Here we use GPU, not CPU for retraining FC layers. The detailed computational time can be found in Fig. 11. We test our datasets in the following two conditions.

1) Directly extracting all the features and retraining the entire features once. Here we use one GPU (1080 TI, 11GB memory) to conduct the training process. There-

fore the total computational time was also composed of the same four computational parts as mentioned: (i) Loading data to GPU, (ii) BP-based computation with GPU, (iii) Extracting all the features from H0 layer, (iv) GPU-based retraining for FC layers.

2) Batch-by-batch training. For big dataset ImageNet, it is impossible to extract all the features and to feed the extracted features for retraining due to GPU memory limitation. Even in multi-GPU environment, 4 GPUs have 44 GB GPU-memory in total, but it is also unable to extract all the features of ImageNet. Therefore in this paper, DenseNet has been tested under the batch-by-batch training condition with our proposed method. For CIFAR10/100, SUN397, and the mini version of ImageNet, we used one 1080 Ti GPU with a batch size of 16. For ImageNet, we use four 1080 Ti GPUs with a batch size of 20 and 232.

2. Matlab environment. The detailed computational cost is shown in Fig.12. We use deep learning to train the CNN network with BP-based algorithm. Moreover, we extract all the features from H0 layer, and feed the features to our CPU-based Matlab code to retrain the FC layers. Therefore, in the Matlab environment, total computational time was composed of four computational parts: (i) Loading data to GPU (deep learning toolbox), (ii) BP-based computation with GPU (deep learning toolbox), (iii) Extracting features from H0 Layer with GPU, (iv) CPU-based retraining for FC layers. Although we use CPU to retrain the FC layers, the computational time is not considerable huge because we only retrain the parameters of the FC layer once;(in other words no iterative learning) in the retraining part.

3.6 Experimental discussion

Based on the experimental results, we cite the following facts.

1. DCNN models with the proposed method vs. original DCNN models. As observed through the experimental results, DCNN models with our method consistently outperform other original DCNN models almost across all the tested databases. As the exact the same experimental settings are used in both BP-based method and our method, the advantage of the performance improvements is evident. Furthermore, the design of a learning algorithm for DCNN models, rather than a new DCNN architecture, remains an important feature for users' capability of directly utilizing the proposed method to the existing DCNN models.

2. Computational cost. The proposed method retrain the weights in FC layers at the end of every training epoch, which is still the natural iterative learning method. Therefore, the proposed method is unable to shorten the training speeds. However, according to the results shown in Fig.11-12, the method does not add many computational workloads (especially in pure GPU environment) but provides improvements in the generalization performance. As seen in Fig.11, major extra computational workload is a feature of the extraction process, and the computational workload of the feature extraction stage theoretically could be entirely removed if Matlab or Keras/Tensorflow can give permissions to allow researchers to modify the core sources or special commends/tools could be released in the future to allow data transmission between GPUs inertly.

3. Retraining Epochs. As seen in Table 3-4, the training epochs used in the retraining process are set as one or two. In fact, all the experimental results of our method are obtained by a nearly non-iterative learning method although the entire training process should be considered an iterative method. As this paper shows that the alternating minimization algorithm used in FC layers benefits generalization performance, it may be an important direction to further investigate the performance of using non-iterative learning methods in Conv. layers in the future.

4 Conclusion

This paper introduces a learning strategy to retrain the parameters of fully-connected layers in deep convolutional neural networks which results in improvements in the efficiency at the training stage and performance boost in the testing accuracy. The experimental results demonstrate that the proposed model achieves state-of-the-art results across several benchmark datasets as compared to highly ranked object recognition methods.

References

[1] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Net.*, vol. 61, pp. 85–117, Jan. 2015.
[2] J. Weng, N. Ahuja, and T. S. Huang, "Cresceptron: a self-organizing neural network which grows adaptively," in *Proc. Int. Jt. Conf. Neural. Netw.*, vol. 1, (Baltimore, US.), pp. 576–581, Jun. 1992.
[3] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Back-propagation applied to handwritten zip code recognition," *Neural Comput.*, vol. 1, no. 4, pp. 541–551, 1989.
[4] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, pp. 504–507, July 2006.
[5] Y. Bengio, P. Lamblin, D. Popovici, and H. Larochelle, "Greedy layer-wise training of deep networks," in *Proc. Adv. Neural Inf. Process. Syst.*, (Vancouver, BC, Canada), 2007.
[6] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P. A. Manzagol, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion," *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec 2010.
[7] M. Chen, K. Weinberger, Z. Xu, and F. Sha, "Marginalizing stacked autoencoders," *J. Mach. Learn. Res.*, vol. 22, no. 2, pp. 191–194, 2015.
[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436–444, May 2016.
[9] K. Fukushima, "Neocognitron: A self-organizing neural network for a mechanism of pattern recognition unaffected by shift in position," *Biol. Cybern.*, vol. 36, no. 4, pp. 193–202, 1980.
[10] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, 1994.
[11] N. Schraudolph and T. J. Sejnowski, "Unsupervised discrimination of clustered data via optimization of binary information gain," in *Proc. Adv. Neural Inf. Process. Syst.*, (San Mateo, US.), pp. 499–506, 1993.
[12] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, June 2015.
[13] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, pp. 1097–1105, 2012.
[14] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," in *British Machine Vision Conference*, 2014.
[15] M. Liang and X. Hu, "Recurrent convolutiaonl neural network for object recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, (Boston, US.), pp. 3367–3375, Jun, 2015.
[16] M.Lin, Q. Chen, and S. Yan, "Network in network," *CoRR*, 2013.
[17] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, June 2015.

[18] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, June 2016.

[20] W. F. Schmidt, M. A. Kraaijveld, and R. P. W. Duin, "Feed forward neural networks with random weights," in *in Proc. Int. Conf. Neural. Netw.*, (The Hague, The Netherlands), pp. 1–4, 1992.

[21] G.-B. Huang, Q.-Y. Zhu, and C.-K. Siew, "Extreme learning machine," in *Technical Report ICIS/03/2004 (also in http://www.ntu.edu.sg/eee/icis/cv/egbhuang.htm)*, (School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore), Jan. 2004.

[22] G.-B. Huang, H.-M. Zhou, X.-J. Ding, and R. Zhang, "Extreme learning machine for regression and multiclass classification," *IEEE Trans. Syst. Man. Cy. B.*, vol. 42, pp. 513–529, April 2012.

[23] R. Zhang, Y. Lan, G.-B. Huang, and Z.-B. Xu, "Universal approximation of extreme learning machine with adaptive growth of hidden nodes," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 23, no. 2, pp. 365–371, 2012.

[24] D. Lam and D. Wunsch, "Unsupervised feature learning classification with radial basis function extreme learning machine using graphic processors," *IEEE Trans. Cybern.*, vol. PP, no. 99, pp. 1–8, 2016.

[25] Y. Yang and Q. M. J. Wu, "Multilayer extreme learning machine with subnetwork nodes for representation learning," *IEEE Transactions on Cybernetics*, vol. 46, pp. 2570–2583, Nov 2016.

[26] M. Carreira-Perpinan and W. Wang, "Distributed optimization of deeply nested systems," in *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics* (S. Kaski and J. Corander, eds.), vol. 33 of *Proceedings of Machine Learning Research*, (Reykjavik, Iceland), pp. 10–19, PMLR, 22–25 Apr 2014.

[27] Z. Zhang, Y. Chen, and V. Saligrama, "Efficient training of very deep neural networks for supervised hashing," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1487–1495, June 2016.

[28] A. Choromanska, S. Kumaravel, R. Luss, I. Rish, B. Kingsbury, M. Rigotti, P. DiAchille, V. Gurev, R. Tejwani, and D. Bouneffouf, "Beyond backprop: Alternating minimization with co-activation memory," *CoRR, arXiv:1806.09077*, 2018.

[29] J. Gemert, C. Geusebroek, C. Veenman, and A. Smeulders, "Kernel codebooks for scene categorization," in *Proc. IEEE Eur. Cof. Comput. Visi.*, (Marseille, France), pp. 696–709, 2008.

[30] L. Zhang, X. Zhen, and L. Shao, "Learning object-to-class kernels for scene classification," *IEEE Trans. Image Process.*, vol. 23, pp. 3241–3253, Aug. 2014.

[31] Y. Han, K. Yang, Y. Ma, and G. Liu, "Localized mutiple kernel learning via simaple-wise alternating optimization," *IEEE Trans. Cybern.*, vol. 44, pp. 137–148, Jan. 2014.

[32] Z. Jiang, Z. Lin, and L. S. Davis, "Label consistent K-SVD: Learning a discriminative dictionary for recognition," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 35, pp. 2651–2664, Nov 2013.

[33] L. Yang, R. Jin, R. Sukthankar, and F. Jurie, "Linear spatial pyramid matching using sparse coding for imge classification," in *Proc. IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, (Miami, US.), pp. 1794–1801, 2009.

[34] S. Gao, I. W.-H. Tsang, L.-T. Chia, and P. Zhao, "Local features are not lonely-laplacian sparse coding for image classification," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 3555–3561.

[35] J. Yu, D. Tao, Y. Cui, and J. Cheng, "Pariwise constraints based multiview features fusion for scene classification," *Pattern Recognit.*, vol. 46, pp. 483–496, Feb 2013.

[36] J. Van Gemert, V. C.J., A. Smeuldes, and J. Geusebroek, "Visual word ambiguity," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, pp. 1271–1283, July 2010.

[37] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognition natural scene categories," in *Proc.IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, (New York, NY., US.), pp. 2169–2178, 2006.

[38] L. Liu, L. Wang, and X. Liu, "In defence of soft-assignment coding," in *Proc. IEEE Int. Conf. Computer Vision*, (Barcelona, Spain), pp. 2486–2493, 2011.

[39] J. X. Wu and J. M. Rehg, "Centrist: A visual descriptor for scene categorization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, pp. 1489–1501, Aug. 2011.

[40] Y. Boureau, J. Ponce, and Y. Lecun, "A theoretical analysis of feature pooling in vision recognition," in *Proc. Int. Conf. Machine Learning*, (Haifa, Israel), 2010.

[41] Y.-l. Boureau and F. Bach, "Learning Mid-Level Features For Recognition," in *Proc.IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, pp. 2559–2566, 2010.

[42] H. Goh, N. Thome, M. Cord, and J.-h. Lim, "Learning Deep Hierarchical Visual Feature Coding," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 2212–2225, 2014.

[43] Y. Yuan, L. Mou, and X. Lu, "Scene recognition by manifold regularized deep learning architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2222–2233, 2015.

[44] B. Zhou, a. J. X. A. Lapedriza, A. Torralba, and A. Oliva, "Learning deep features for scene recognition using places database," in *Proc.Neural Inf. Process. Syst.*, 2014.

[45] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, "Places: A 10 million image database for scene recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 40, pp. 1452–1464, June 2018.

[46] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[47] C. Dan, M. Ueli, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Proc.IEEE Int. Conf. Comput. Vis. Pattern Recognit.*, CVPR '12, (Washington, DC, USA), pp. 3642–3649, 2012.

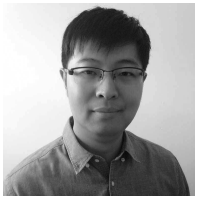
[48] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. A. Riedmiller, "Striving for simplicity: The all convolutional net," pp. 1–18, 2015.

[49] C. Lee, S. Xie, P. Gallagher, Z. Z.Y., and Z. Tu, "Deeply-supervised nets," *CoRR*, 2015.

[50] M. F. Stollenga, J. Masci, F. Gomez, and J. Schmidhuber, "Deep networks with internal selective attention through feedback connections," in *Advances in Neural Information Processing Systems 27*, pp. 3545–3553, 2014.

[51] R. Gens and P. Domingos, "Discriminative learning of sum-product networks," in *Advances in Neural Information Processing Systems 25*, pp. 3239–3247, Curran Associates, Inc., 2012.

[52] F.-F. Li and P. Perona, "A bayesian hierarchical model for learning natural scene categories," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, (San Diego, US.), pp. 524–531, Jun, 2005.



Yimin Yang (S'10-M'13) received his Ph.D. degrees in Electrical Engineering from Hunan University, China, in 2013.

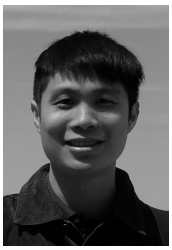
He is currently an Assistant Professor at Computer Science Department in Lakehead University, Thunder Bay, Ontario, Canada. From 2014 to 2018, he was a Post-Doctoral Fellow with the Department of Electrical and Computer Engineering at the University of Windsor, Ontario, Canada. He has authored or coauthored more than 40 refereed papers. His research interests are artificial neural networks, hybrid system approximation, and image feature selection.

Dr. Yang was the recipient of the Outstanding Ph.D. Thesis Award of Hunan Province, and the Outstanding Ph.D. Thesis Award Nominations of Chinese Association of Automation, China, in 2014 and 2015, respectively. He has been serving as a Reviewer for international journals of his research field, a Guest Editor of multiple journals, and a Program Committee Member of some international conferences.

Q. M. Jonathan Wu (M'92-SM'09) received his Ph.D. in Electrical Engineering from the University of Wales, Swansea, U.K., in 1990.

He was affiliated with the National Research Council of Canada for ten years beginning in 1995, where he became a Senior Research Officer and a Group Leader. He is currently a Professor with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, Ontario, Canada. He has published more than 300 peer-reviewed papers in computer vision, image processing, intelligent systems, robotics, and integrated microsystems. His current research interests include 3-D computer vision, active video object tracking and extraction, interactive multimedia, sensor analysis and fusion, and visual sensor networks.

Dr. Wu holds the Tier 1 Canada Research Chair in Automotive Sensors and Information Systems. He is an Associate Editor for the IEEE Transaction on Cybernetics, the IEEE Transactions on Circuits and Systems for Video Technology, and the journal of Cognitive Computation. He has served on technical program committees and international advisory committees for many prestigious conferences.



Xiexing Feng received his bachelor's degree in Mechanical Design, Manufacturing and Automation from North University of China, in 2013. He is currently a Ph.D. Candidate with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, Ontario, Canada. His research focuses on machine learning and computer vision.

Akilan Thangarajah (S'07) received his Ph.D. in Electrical and Computer Engineering from the University of Windsor, Windsor, Canada, in 2018.

He is a Postdoctoral fellow in the Electrical and Computer Engineering at the Computer Vision and Sensing Systems Laboratory, University of Windsor, Windsor, Ontario, Canada. His research interest includes object and action recognition, image/video processing and segmentation, and data fusion using statistical techniques, machine learning, and deep learning. He is a recipient of 2015-2016 Golden Key's premier Graduate Scholar Award. He serves as a reviewer for several journals, including IEEE Transactions on Multimedia and IEEE Transactions on Industrial Informatics.