EURASIP Journal on Advances in Signal Processing

RESEARCH **Open Access**

An efficient pruning scheme of deep neural networks for Internet of Things applications

Chen Qi[1], Shibo Shen[1], Rongpeng Li[1*], Zhifeng Zhao[2], Qing Liu[3], Jing Liang[3] and Honggang Zhang[1]

*Correspondence:
lirongpeng@zju.edu.cn
[1]College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China
Full list of author information is available at the end of the article

Abstract
Nowadays, deep neural networks (DNNs) have been rapidly deployed to realize a number of functionalities like sensing, imaging, classification, recognition, etc. However, the computational-intensive requirement of DNNs makes it difficult to be applicable for resource-limited Internet of Things (IoT) devices. In this paper, we propose a novel pruning-based paradigm that aims to reduce the computational cost of DNNs, by uncovering a more compact structure and learning the effective weights therein, on the basis of not compromising the expressive capability of DNNs. In particular, our algorithm can achieve efficient end-to-end training that transfers a redundant neural network to a compact one with a specifically targeted compression rate directly. We comprehensively evaluate our approach on various representative benchmark datasets and compared with typical advanced convolutional neural network (CNN) architectures. The experimental results verify the superior performance and robust effectiveness of our scheme. For example, when pruning VGG on CIFAR-10, our proposed scheme is able to significantly reduce its FLOPs (floating-point operations) and number of parameters with a proportion of 76.2% and 94.1%, respectively, while still maintaining a satisfactory accuracy. To sum up, our scheme could facilitate the integration of DNNs into the common machine-learning-based IoT framework and establish distributed training of neural networks in both cloud and edge.

Keywords: Deep neural networks, Deep learning, Internet of Things, Resource-limited edge computing, Pruning, Efficiency

1 Introduction

The Internet of Things (IoT), which aims to integrate the physical world by collecting and sharing information [1, 2], has been widely used in various areas, including smart city[3], smart transportation [1], smart home [4], and smart agriculture [5]. Moreover, extensive applications with IoT devices generate a large amount of data and it becomes incentive to utilize data-driven deep neural networks (DNNs) to further extract accurate information [6]. For example, a large number of biomedical data such as medical images could be smartly recognized by the convolutional neural network (CNN) to monitor human health [7]. Moreover, CNN has been widely used to process image data on IoT devices such as wireless sensor cameras [6, 8] and smart phones [7]. The authors in [3] propose several CNN-based applications in typical IoT scenarios, such as recognizing garbage images for waste management and monitoring parking spaces for smart parking lot management. Statistical evidence in [3] also shows that CNN is considered as one of the most extensively applied deep learning models for various IoT applications.

However, though it is generally believed that neural networks need to be complicated enough to represent the real-world targeted objects [9, 10], deep convolutional neural networks, usually with huge overhead and complexity in both storage and computing, are very difficult to be directly applied to resource-limited IoT devices [2]. It is essential to reduce the model size of CNN due to its large computational overhead. To address this issue, previous work primarily focuses on reducing the computational overhead and storage cost of DNNs by carefully designing the corresponding network architecture, e.g., VGG [11], GoogLeNet [12], ResNet [13], and MobileNet [14], in regarding to complex CNN for processing images. Besides, network-pruning is often adopted to compress the deep neural network itself by removing un-important inter-layer connections [15, 16], neurons or entire channels in CNN [17–19]. An intuitive overview is depicted in Fig. 1.

Pruning at the scale of kernel in the convolutional layer, called as filter-level pruning or channel pruning, has been extensively studied and achieved exciting results with huge reduction in computation and negligible performance loss in accuracy [17–20]. However, these pruning schemes generally follow the basic three-stage procedure (as shown in Fig. 2), i.e., training a redundant network from scratch, pruning it and re-training it for accuracy recovery [15], which is cumbersome and time-consuming especially for resource-limited IoT devices, leading to a huge gap between theoretical performance and practical applications. Therefore, it remains a critical concern in practical IoT scenarios to improve the traditional pruning-based DNN compression process before its efficient application.

To cope with the aforementioned issue, we put forward a more concise and lightweight deep learning scheme to reveal an efficient and compact CNN structure in a more efficient manner. The typical process of our proposed scheme is depicted in Fig. 2. Specifically, the proposed strategy can be divided into two phases, i.e., structure learning and weight learning, and the latter functions in the same way as the conventional training

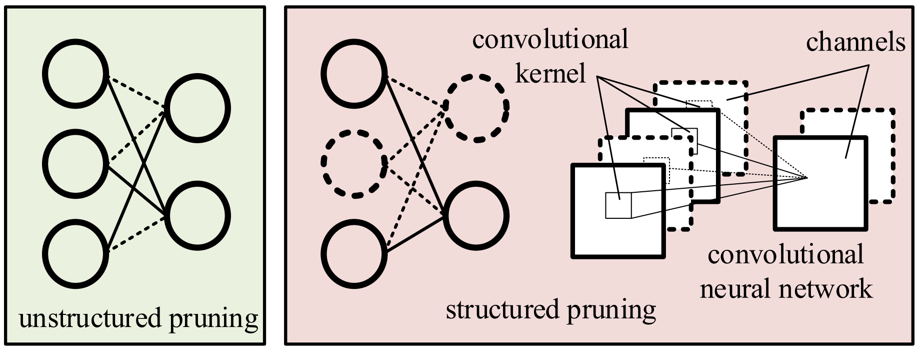


Fig. 1 A high level view of how pruning functions in compressing deep neural networks. Traditional pruning schemes can be divided into two classes, that is, unstructured pruning that simply removes inter-layer connections between neurons while structured pruning removes neurons as well as their connected weights, or channels with their corresponding kernels w.r.t. the convolutional neural network

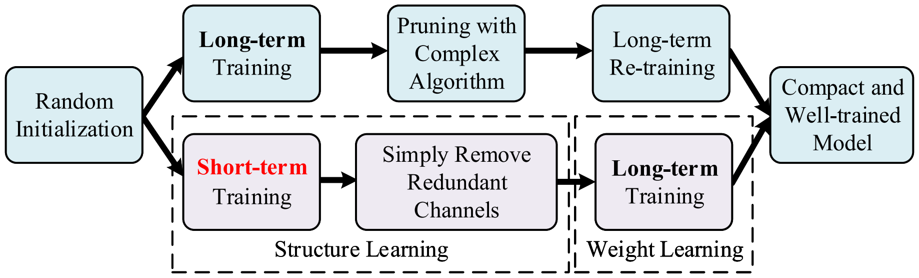
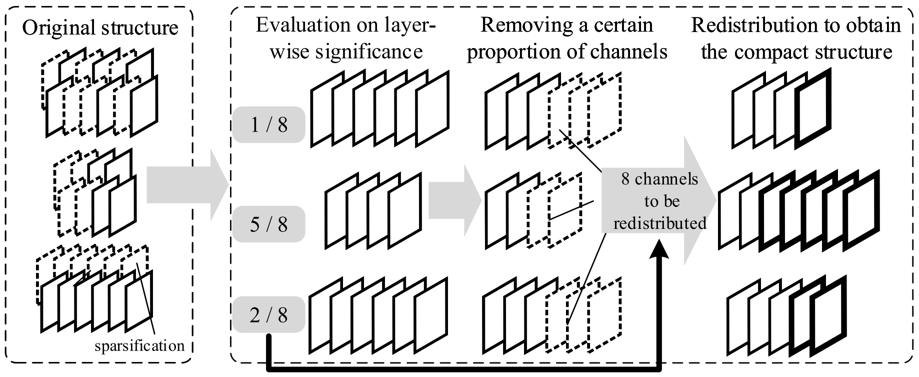


Fig. 2 The comparison of conventional pruning process (upper) and our proposed one (lower). We divide the original training and pruning procedure into two phases, that is, the short-term structure-learning and the long-term weight-learning

process. During the period of structure-learning, we focus on evaluating the significance of each channel and unveiling a compact yet effective structure. To achieve the objective, we propose to evaluate the channels' significance by Taylor criterion introduced by [17] and redistribute the remaining channels, which is stemed from weight-redistribution proposed by [21, 22]. The criterion of Taylor-expansion aims to discover those channels whose removal leads to more impact on the final loss. However, such a criterion is only calculated on the basis of a single mini-batch. In order to obtain all the channels' significance evaluation over the entire data set, we propose a long-term assessing variable called as *feature-saliency*, which is computed by the moving average on each batch's evaluation criterion. Simultaneously, considering the common finding that layers are not equally important in a deep neural network [18], we prefer to allocate more channels to sensitive layers, namely pruning less parameters in these layers. To achieve this goal, we extend the original algorithm with regard to weight-redistribution [21] to convolutional kernels and call it as channel-redistribution. Generally, inspired by the basic weights redistributing steps suggested in [21], we firstly calculate the saliency of different layers, then temporarily remove a certain proportion of channels in each layer and finally redistribute those removing channels according to the layer-wise saliency. We summarize the novel channel-redistribution algorithm in Fig. 3. Next, we have to remove the surplus channels with



Original structure
sparsification
Evaluation on layer-wise significance
Removing a certain proportion of channels
Redistribution to obtain the compact structure
1 / 8
5 / 8
2 / 8
8 channels to be redistributed

Fig. 3 An overview of channel-redistribution. The basic procedure includes evaluating the layer-level saliency, temporarily removing a certain proportion of channels and redistribution. Note that a process of sparsification is conducted at the beginning of channel-redistribution

their corresponding kernels and train the preserved weights after obtaining the compact structure.

It is noteworthy that the new training model at the final stage (i.e., weight learning) is much smaller than the original one in terms of both computational cost and number of hyperparameters, implying the training process is relatively fast. In other words, the time-consuming training of a large neural network in traditional pruning methods could be avoided in our proposed scheme. The process of learning the compact structure also solves the problem of how to design an efficient DNN, namely determining the appropriate structure of neural network to be used for resource-stringent IoT devices. On the other hand, there are also some researches on solving the time-consuming training of DNNs for IoT applications by introducing the distributed architecture [2, 6, 23, 24]. A typical distributed learning process for IoT consists of training a redundant deep neural network at the cloud computing servers, and then pushing it to edge nodes. Actually, our scheme can further bridge the connection between redundant and compact neural networks at cloud and edge nodes, as depicted in Fig. 4. Considering a kind of application scenario where a large DNN model has been trained generally at a cloud server, we can retrain and prune it to gain compact networks to be more suitable for some specific IoT tasks. Compared with directly training a compact neural network from scratch, our proposed scheme transfers the knowledge of original neural networks and is able to achieve better performance, e.g., faster convergence and higher efficiency.

Our major contribution can be summarized as follows.

- Inspired by previous work on pruning [17, 21], we propose a novel training strategy for learning compact and efficient neural networks. The proposed scheme can achieve comparatively good performance with significantly reduced model size, computational complexity, and negligible accuracy loss. Compared with the traditional pruning-based DNN compression methods, our scheme is more concise and realizes end-to-end DNN compression. Moreover, our scheme also overcomes the dilemma of designing neural network structures through adaptive structure-learning.
- We incorporate our lightweight scheme into the common IoT applications and establish a novel paradigm for applying DNN to IoT scenarios with resource constraints yet heavy tasks. The proposed paradigm is also capable to migrate large deep neural networks to edge computing nodes through compression and re-training, which efficiently facilitates to adapt to any specific edge tasks.

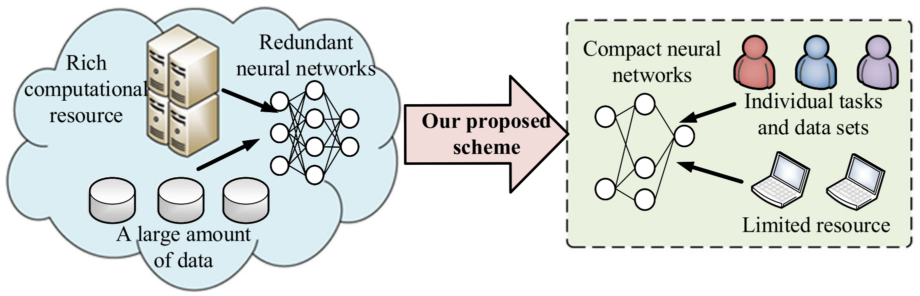

Rich computational resource
Redundant neural networks
A large amount of data
Our proposed scheme
Compact neural networks
Individual tasks and data sets
Limited resource

Fig. 4 A high level view of how our proposed scheme functions in converting a large neural network at cloud level to a lightweight neural network in edge nodes

- We conduct extensive experiments on various standard benchmark datasets, including CIFAR-10 [25] and ILSVRC-12 [9], and compare with the well-recognized advanced CNN architectures, including VGG [11], ResNet [13], and MobileNet [14]. Simulation results verify the effectiveness of our scheme.

The remainder of this paper is organized as follows: Section 2 talks about some necessary backgrounds on deep neural networks and formulates the DNN-based IoT applications scenario. Section 3 gives the details of our proposed pruning scheme in terms of mathematical formulation and algorithm, while Section 4 presents the detailed experimental results. Finally, Section 5 summarizes the paper and offers future directions.

2 Background of CNN pruning

2.1 DNN-powered IoT

As mentioned before, a large amount of data produced by IoT devices promotes the application of data-driven deep nerural networks to automatically extract useful representations from raw data [2, 6]. Among many deep learning methods, CNN has been extensively used to process two-dimensional data and is further applied to IoT devices, such as smart wireless cameras [6, 8], or applications [3, 6–8, 26]. Typically, CNN, being composed of convolutional layers, pooling layers, and fully connected layers (as shown in Fig. 5), has a large number of parameters and huge computational overhead that limits its extensive application for resource-constrained IoT devices. Therefore, reducing the complexity of CNN has become an imperative research topic and pruning belongs to one popular means.

2.2 Related works on pruning

Unstructured pruning Early works generally focus on pruning deep neural networks by removing redundant weights according to their magnitude [15, 16]. However, in order to obtain the significance of various weights, they have to start from training a redundant neural network in advance. In addition, the pruning weights are determined by rigidly setting a global threshold of magnitude for the whole deep neural network. Later work [27] proposes to improve the traditional pruning process by selectively learning the corresponding weights with greater impact on loss while discarding the others through cutting off their gradient flow. Moreover, both [22] and [21] propose a smoother way, namely redistributing the remaining weights, in order to obtain a proper compact structure instead of setting a threshold. They both suggest to allocate more weights to the sensitive layers, although the detailed approaches differ concretely. Our scheme aims to

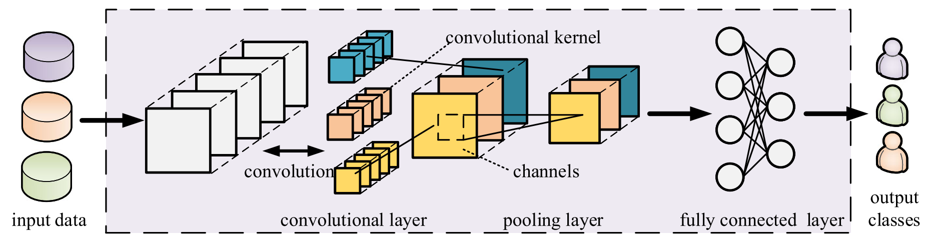


Fig. 5 A typical architecture of CNN that consists of several convolutional layers, pooling layers, and fully-connected layers

extend their work to another kind of redistribution at the scale of convolutional kernel, namely channel-wise redistribution for structured pruning.

Structured pruning Due to the reason that weight-pruning does not significantly reduce the amount of computation load, researchers begin to pay attention to large-scale pruning, i.e., filter-pruning or channel-pruning. Specifically, both the work [28] and [19] introduce an extra loss of "Group LASSO" to compel some kernels or the corresponding weights in batch-normalization layers [29] to zero and prune them at the end of each training. In addition, the work [30] introduces a discrimination-aware loss to keep channels that contribute to the discriminative power of neural networks. Some other methods propose to prune channels through optimizing the formulation of reconstruction error [20, 31], reducing the similarity between features [32, 33], and directly evaluating channels' significance [17, 18]. Our algorithm is based on the evaluation of channel saliency as well. Furthermore, some recent pruning methods introduce advanced machine-learning-based approaches, such as meta-learning [34] and generative-adversarial-learning [35], which also achieve remarkable results.

Pruning with new paradigms Nearly none of the aforementioned methods deviates from the three basic steps of pruning, that is, training an over-parameterized neural network, pruning, and fine-tuning it. Based on the argument in [36] that a compact DNN model trained from scratch can reach competitive performance compared with its redundant counterpart, the traditional pruning strategy may be too time-consuming and outdated, thereby not suitable for the cloud-to-edge distributed computing architecture for IoT applications. Recent work like [37] introduces a novel pruning strategy that temporally removes unimportant kernels but keeps them updated in the phase of training, namely soft pruning. Moreover, the paper [38] proposes to prune the model from scratch on the basis of random initialization. This model in [38] to find a compact structure by introducing group LASSO loss to the batch-normalization layers as same as network slimming [19]. However, our scheme differs in that we are inspired by the works in [17] and [21] and design a completely different structure-learning algorithm through evaluating channels' importance and channel redistribution accordingly. In addition, all the parameters of the neural network are also updated simultaneously during the structure-learning process, in contrast to [38], since some prior weights in the training of large neural networks are still effective for the training of the compact counterparts, which is better than random initialization. In addition, some other pruning methods [39] propose to learn an efficient structure by automatic search that functions in a similar way to Network Architecture Search (NAS) [40]. In fact, many NAS schemes [41–43] aim to find a proper structure with excellent performance on exact datasets. However, NAS-based schemes require much more computing resources and data to search for connections between neurons or convolutional channels from scratch, while pruning-oriented schemes, based on exiting models, aim to reduce the complexity by searching over a smaller space with less resource overhead, and therefore are more suitable for IoT terminal deployment.

2.3 Potential applications in IoT scenarios

In order to reduce both training and inference cost of DNN, previous works take into account the cloud-and-edge computing architecture for data-heavy IoT applications and propose a distributed computing paradigm [2, 6, 23, 24]. As illustrated in Fig. 4, one may regard our proposed paradigm as a supplement to the original architecture, in which we improve the conventional process of copying the parameters from the cloud to the edge by introducing an efficient re-training scheme with structure-learning and weight-optimization, thereby making the model adaptable to any personalized IoT applications as well as reducing the redundant parameters and computational overhead.

3 Methods

3.1 Notations

Beforehand, we formally give some symbol notations used throughout the paper. Suppose we have a deep neural network with L convolutional layers, $\mathbf{w}_k^l$ and $\mathbf{z}_k^l$ are used to represent the convolutional kernel and the individual output channel of the l-th convolutional layer, respectively. The subscript $k \in [1, \cdots, C^l]$ represents the channel index, where C^l indicates the total number of output channels in the corresponding layer. We further use H^l and W^l to indicate the height and width of channels in the l-th layer, respectively. Pruning the k-th channel in layer l signifies removing the corresponding kernel $\mathbf{w}_k^l$. Moreover, we define $\mathbf{f}^l$ to represent long-term evaluation of channels, i.e., feature saliency, $\mathbf{f}^l \in \mathbb{R}^{C^l}$. Overall, we summarize all notations in Table 1.

At the beginning of training, each layer retains the same proportion of channels, which is controlled by the pruning rate p. These preserved channels will be adaptively redistributed at the end of each training epoch. Succinctly, the preserved channels are called as *activated channels*. We further define $[a^l]_i$ to represent the number of activated channels in the l-th layer where the subscript i refers to the iterative epoch of training. The initialized values of $[a^l]_i$ are

Table 1 Notations and their definitions

Notation	Definition
L	The number of convolutional layers
p	The overall pruning rate for all channels
C^l	The original total number of channels in each layer, $1 \le l \le L$
$\mathbf{w}_k^l$	The convolutional kernel, $\mathbf{w}_k^l \in \mathbb{R}^{C^{l-1}\times 3\times 3}, 1 \le l \le L, 1 \le k \le C^l$
H^l	The height of channels, $1 \le l \le L$
W^l	The width of channels, $1 \le l \le L$
$\mathbf{z}_k^l$	The feature map or channel, $\mathbf{z}_k^l \in \mathbb{R}^{H^l\times W^l}, 1 \le l \le L, 1 \le k \le C^l$
$\mathbf{f}^l$	The feature saliency, $\mathbf{f}^l \in \mathbb{R}^{C^l}$ $f_k^l \in \mathbf{f}^l, 1 \le k \le C^l$
$[a^l]_i$	The remaining channels in each layer in the i-th training epoch, $1 \le l \le L$
Θ_k^l	The evaluation on channels' significance w.r.t. a single mini-batch, $1 \le l \le L, 1 \le k \le C^l$
$[\xi^l]_i$	The layers' significance evaluation in the i-th training epoch, $1 \le l \le L$
J	The loss function adopted to evaluate the difference between the observed values and the actual ones
ϵ	The smoothing factor
s	The proportion of redistributing channels

$$[a^l]_0 = pC^l \quad \forall\, l, 1 \le l \le L \tag{1}$$

3.2 Criterion of channel significance

In order to evaluate the channels' saliency, we adopt a Taylor-expansion [17] based criterion. Considering a mini-batch $B = \left\{X = \{x_1, x_2, ..., x_m\}, Y = \{y_1, y_2, ..., y_m\}\right\}$, the final loss on the batch B can be defined as $J(B, W)$ where W represents the network parameters. Suppose a kernel $\mathbf{w}_k^l$ with respect to its activation $\mathbf{z}_k^l$ is removed, the corresponding impact on the cost function J can be expressed as

$$\left|\Delta J(\mathbf{z}_k^l)\right| = \left|J(B, \mathbf{z}_k^l) - J(B, \mathbf{z}_k^l \to 0)\right| \tag{2}$$

We use the Taylor series to expand the cost function at point $\mathbf{z}_k^l = 0$

$$J(B, \mathbf{z}_k^l \to 0) = J(B, \mathbf{z}_k^l) - \frac{\partial J}{\partial \mathbf{z}_k^l}\mathbf{z}_k^l + o\left(\left(\mathbf{z}_k^l\right)^2\right) \tag{3}$$

Ignoring the higher-order remainder and substituting (3) to (2), we have

$$\begin{aligned} \Theta_k^l \triangleq \left|\Delta J(\mathbf{z}_k^l)\right| &= \left|J(B, \mathbf{z}_k^l) - J(B, \mathbf{z}_k^l) + \frac{\partial J}{\partial \mathbf{z}_k^l}\mathbf{z}_k^l\right| \\ &= \left|\frac{\partial J}{\partial \mathbf{z}_k^l}\mathbf{z}_k^l\right| \end{aligned} \tag{4}$$

The criterion can be regarded as a measure of the significance of feature maps for a single-entry mini-batch. For a channel with multi-variate output, the item Θ_k^l can be rewritten as

$$\Theta_k^l = \left|\frac{1}{M}\sum_{m=1}^{M} \frac{\partial J}{\partial z_{k,m}^l} z_{k,m}^l\right| \tag{5}$$

where M is the total number of channel's entries. The computation of item Θ_k^l requires the activation and the gradient, which can be calculated from the forward and backward propagation, respectively. Furthermore, we impose an extra re-scaling method with max-normalization, that is

$$\hat{\Theta}_k^l = \frac{\Theta_k^l}{\max\limits_j \left\{\Theta_j^l\right\}} \tag{6}$$

Such normalization process is essential since we need to ensure that these evaluation values of each layer are at the same scale. Its function is similar to batch-normalization [29], which ensures that the statistics of layer-wise evaluation values are under the same distribution. Equation (6) indicates that the maximum criterion values regarding different layers are all normalized to 1, resulting in comparable scale of feature saliency $\mathbf{f}^l$, which is defined as a long-time estimating variable for individual channels

$$\left[f_k^l\right]_{\text{new}} = \epsilon \left[f_k^l\right]_{\text{old}} + \hat{\Theta}_k^l, f_k^l \in \mathbf{f}^l, 1 \le k \le C^l \tag{7}$$

where the hyper-parameter ϵ is a smoothing factor set to 0.98 for all experiments in this paper. The feature-saliency helps determine which channels are retained when the structure is fixed. Note that the values of $\mathbf{f}^l$ update with each mini-batch in a training epoch, we omit the iterative epoch index i for simplicity of representation.

3.3 Channel redistribution

The proposed channel redistribution process occurs at the end of each training epoch, which is indicated by the subscript i. Note that the aforementioned feature saliency evaluation is based on a single channel, it is necessary to calculate the significance of each layer which has several channels in order to obtain an efficient structure. Suppose $[\xi^l]_i$ indicates the corresponding layer's significance for the iterative epoch i

$$[\xi^l]_i = \frac{\sum_k \hat{f}_k^l}{[a^l]_{i-1}}, \ \hat{f}_k^l \in \hat{\mathbf{f}}^l \tag{8}$$

where $\hat{\mathbf{f}}^l \in \mathbf{f}^l$ is its subset which contains several large values of feature saliency of the corresponding layer and the total number of elements in $\hat{\mathbf{f}}^l$ is $[a^l]_{i-1}$. Next we need to normalize all $[\xi^l]_i$ so that the sum of these values is 1.

$$[\hat{\xi}^l]_i = \frac{[\xi^l]_i}{\sum_j [\xi^j]_i} \tag{9}$$

Looking again at the channel redistribution process shown in Fig. 3, after obtaining each layer's significance evaluation, the following step is to temporarily remove a fixed proportion of channels to release some reallocating space, followed by redistributing channels according to the calculated values about layers' significance, that is, updating the number of activated channels $[a^l]_{i-1}$ in each layer. Given that the updated value may exceed the maximum number of channels in the corresponding layer, the value of $[a^l]_i$ is limited to C^l which is the total number of channels in the original structure as the following formula

$$[a^l]_i = \min\left\{(1-s)[a^l]_{i-1} + [\hat{\xi}^l]_i \left(s \sum_{l=1}^{L} [a^l]_{i-1}\right), C^l\right\} \tag{10}$$

where the sparsity s is a hyper-parameter which is predefined to indicate how many channels are reallocated each time. Different from the original work [21] that adjusts the value of s throughout training, the sparsity s is fixed to 0.5 in our experiments. Moreover, we allocate the extra channels evenly among the other layers if necessary. All the relevant details are shown in Algorithm 1.

Furthermore, after uncovering a suitable compact structure, we need to remove those insignificant channels with their convolutional kernels and train the remaining weights to obtain the representative capability, as depicted in Fig. 2. In the period of pruning, the remaining channels are determined according to their feature saliency $\mathbf{f}^l$ as well as the number of the activated channels a^l in the corresponding layer. Overall, we summarize the total process of our pruning scheme in Algorithm 2. For the reason that channel-pruning is simply applied to convolutional layers, we have omitted the general batch-normalization [29] layers, activation layers, pooling layers, and fully connected layers for simplicity.

3.4 Discussion

As most of the heavy computation is concentrated on convolutional layers, we only need to pay attention to computational overhead or saving in these layers. Suppose the output channel size of the l-th layer is $H^l \times W^l$ and the final number of activated channels is A^l, accordingly $\left(C^l - A^l\right)$ kernels in the corresponding layers will be removed. Therefore, the dimension of remaining channels in the lth layer is $H^l \times W^l \times A^l$ and the computation

Algorithm 1 Channel Redistribution

```
Input: feature saliency $\{\mathbf{f}^l,\ 1 \le l \le L\}$, activated channels in last epoch $\{[a^l]_{i-1}, 1 \le l \le L\}$, the sparsity $s$, maximum number of channels in each layer $\{C^l, 1 \le l \le L\}$
Output:the updated numbers
  for $l \leftarrow 1$ to $L$ do
    calculate for layer-wise significance
      $[\xi^l]_i \leftarrow \left(\sum_k f_k^l\right) / [a^l]_{i-1}$
  end for
  normalization across all layers $[\xi^l]_i \leftarrow [\xi^l]_i / \sum_j [\xi^j]_i$
  remove and redistribute channels, define a variable $e$ to count for extra number of which the reallocated channels exceed the original maximum $e \leftarrow 0$
  for $l \leftarrow 1$ to $L$ do
    if $(1-s)[a^l]_{i-1} + [\hat{\xi}^l]_i \left(s \sum_l [a^l]_{i-1}\right) <= C^l$ then
      $[a^l]_i \leftarrow (1-s)[a^l]_{i-1} + [\hat{\xi}^l]_i \left(s \sum_l [a^l]_{i-1}\right)$
    else
      $[a^l]_i \leftarrow C^l$
      $e \leftarrow e + (1-s)[a^l]_{i-1} + [\hat{\xi}^l]_i \left(s \sum_{l=1}^{L} [a^l]_{i-1}\right) - C^l$
    end if
  end for
  reallocate extra channels in $e$
  repeat
    for $l \leftarrow 1$ to $L$ do
      if $[a^l]_i < C^l$ then
        update both $[a^l]_i$ and $e$ by
          $[a^l]_i \leftarrow [a^l]_i + 1, \ e \leftarrow e - 1$
      end if
    end for
  until $e <= 0$
  return the update numbers $\{[a^l]_i, 1 \le l \le L\}$
```

in terms of FLOPs (floating-point operations) in such layer decreases from $K^2 \times C^{l-1} \times H^l \times W^l \times C^l$ to $K^2 \times A^{l-1} \times H^l \times W^l \times A^l$, where the label K indicates the kernel size. Compared to the raw FLOPs with respect to individual layers, a reduced ratio of $\left(1 - \frac{A^{l-1}A^l}{C^{l-1}C^l}\right)$ is obtained, leading to large decrease in the computational cost of CNN.

4 Results and discussion

4.1 Experimental setting

We evaluate our scheme on various representative benchmark datasets, including CIFAR-10 [25] and ILSVRC-12 [9], and compare with the advanced DNN architectures, including VGG [12], ResNet [13], and MobileNet [14]. CIFAR-10 contains 50,000 training images and 10,000 testing images, which are categorized into 10 classes. We follow the common data augmentation suggested by [13] with shifting and mirroring. Both architectures are trained from scratch using Stochastic Gradient Descent (SGD) with an initial learning rate of 0.1. The learning rate is decayed by 10 times in every one third of the total number of iterations. The weight decay and momentum is 10^{-4} and 0.9, respectively. ILSVRC-12

Algorithm 2 The proposed pruning scheme

```
Input: model $\{\text{conv}^l,\ 1 \le l \le L\}$, pruning rate $p$,other parameters in neural-network training
Output:the compact model
  # Initialization Process
  for $l \leftarrow 1$ to $L$ do
    obtain maximum number of channels in each layer $C^l$
    Initialize feature-saliency $\mathbf{f}^l \leftarrow \mathbf{0}$
    Initialize activated channels' number $[a^l]_0 \leftarrow pC^l$
  end for
  Initialize smoothing factor $\epsilon \leftarrow 0.98$
  Initialize sparsity $s \leftarrow 0.5$
  # Structure Learning
  set the epoch index $i \leftarrow 1$, stop criterion prop $\leftarrow 1.0$
  repeat
    for each mini-batch $\mathbf{z}^0$ in the $i$-th epoch do
      forward $\mathbf{z}^l \leftarrow \text{conv}^l(\mathbf{z}^{l-1})$ for $l \leftarrow 1$ to $L$
      set $\mathbf{g}^{L+1} \leftarrow 1$, $\mathbf{z}^{L+1} \leftarrow J$
      backward $\mathbf{g}^l \leftarrow \mathbf{g}^{l+1} \cdot \frac{\partial \mathbf{z}^{l+1}}{\partial \mathbf{z}^l}$ for $l \leftarrow L$ to 1
      compute the criterion according to (5) and (6)
        $\Theta^l \leftarrow \text{Max-norm}\left[\text{average}\left(\mathbf{g}^l \cdot \mathbf{z}^l\right)\right]$
      update feature saliency by $\mathbf{f}^l \leftarrow \epsilon \mathbf{f}^l + \Theta^l$
    end for
    reallocate channel numbers by Algorithm 1
      $[a^l]_i \leftarrow \text{Channel-Redistribution}(\mathbf{f}^l, [a^l]_{i-1}, s, C^l)$
    calculate the redistributed proportion
      prop $\leftarrow \sum_l |[a^l]_i - [a^l]_{i-1}| / \sum_l [a^l]_i$
    update the training epoch $i \leftarrow i + 1$
  until prop $<= 0.01$
  # Pruning
  prune insignificant channels with their corresponding kernels and copy the remaining parameters to a compact model
    $\{\text{conv}^l, 1 \le l \le L\} \leftarrow \text{Pruning}(\text{conv}^l, \mathbf{f}^l, a^l)$
  # Weight Learning
  train the compact model in the remaining epochs
  return the compact and well-trained CNN $\{\text{conv}^l, 1 \le l \le L\}$
```

contains 1.3 million training images and 50,000 validating images without test set. While evaluating on ILSVRC-12, we also follow the training settings and the strategy of data augmentation suggested by [13] and adopt Pytorch [44] which is the fundamental framework of our experiments. Note that those advanced CNN architectures including VGG and MobileNet are designed for large dataset like ImageNet, re-training, and pruning them to match small dataset like CIFAR-10 could be viewed as a suitable verification platform of distributed training process in both cloud and edge nodes for IoT applications.

In order to verify the effectiveness of our proposed scheme, we formally compare our scheme's performance with that of various state-of-the-art pruning approaches, including PFEC [18], NS [19], CP [31], ThiNet [20], SFP [37], CFP [32], DCP [30], FPGM [45], COP [33], GAL [35], PFS [38], and ASS [39]. Moreover, we present the performance of our scheme in terms of both theoretical acceleration and practical acceleration with respect to various pruning rates to show the robustness and efficiency of our scheme. Overall, our proposed method has achieved comparable and satisfactory results even with more concise pruning program, which could be effectively incorporated into the common distributed training paradigm for anticipated IoT applications.

4.2 Experiments on CIFAR-10

Pruning VGG. Though VGG is not designed for small data set like CIFAR-10, previous work have studied its performance at extremely high pruning rates. We firstly train an original non-pruned 16-layer VGG as baseline (no pruning) and then run several experiments with different pruning rates from scratch. We compare the testing accuracy with that of the previous state-of-the-art approaches and summarize the corresponding results in Table 2.

As shown in Table 2, our proposed scheme can achieve comparable results with the aforementioned state-of-the-art methods with different reduced FLOPs and parameters. For instance, a compact model with 49.3% in FLOPs drop achieves superior accuracy compared to the baseline performance. In the case where the FLOPs and number of parameters are reduced by 72.6% and 94.1%, respectively, the pruned VGG based on our scheme can still maintain an applicable accuracy of 93.27% for such dataset.

Pruning ResNet. Note that compact ResNet architectures with less channels in each layer are built up in [13] for recognizing images from CIFAR-10, we adopt the recommended 32-layer and 56-layer ResNet as baselines (no pruning). Specifically, for the reason that the input/output number of channels within a residual block must be consistent to ensure the short-cut connection, we only prune the first layer's output channels per block.

It can be observed from Table 3 that our proposed strategy can achieve competitive results. For example, the compact ResNet-32 with 49.0% reduction in FLOPs and 60.1% reduction in parameters still retains an accuracy of 92.50% (i.e., 93.20–0.70% = 92.50%). In addition, more experiments on pruning ResNet-56 further verify the effectiveness of our

Table 2 Results of pruning VGG on CIFAR-10

Method	Baseline (%)	Accuracy (%)	FLOPs pruned (%)	Parameters pruned (%)
PFEC[18]	93.25	↑ 0.15	34.2	64.0
NS[19]	93.66	↑ 0.14	51.0	88.5
CFP[32]	93.49	↓ 0.51	81.9	-
COP[33]	93.56	↓ 0.25	73.5	92.8
GAL[35]	93.96	↓ 0.54	45.2	82.2
PFS[38]	93.44	↑ 0.19	50.0	-
Ours	93.50	↑ 0.25	49.3	83.8
		↓ 0.23	72.6	94.1

We report both baseline and after-pruning accuracy of the state-of-the-art methods referring to their papers. The label "-" in last column indicates that such item is not reported

Table 3 Results of pruning ResNet on CIFAR-10

Architecture	Method	Baseline (%)	Accuracy (%)	FLOPs pruned (%)	Parameters pruned (%)
ResNet-32	SFP[37]	92.63	↓ 0.55	41.5	-
	FPGM[45]	92.63	↓ 0.70	53.2	-
	COP[33]	92.64	↓ 0.67	53.9	57.5
	Ours	93.20	↓ 0.21	33.0	31.7
			↓ 0.70	49.0	60.1
ResNet-56	PFEC[18]	93.04	↑ 0.02	27.6	13.7
	CFP[32]	93.57	↓ 0.25	61.5	-
	DCP[30]	93.80	↓ 0.31	50.3	50.7
	PFS[38]	93.23	↓ 0.18	50.0	-
	ASS[39]	93.26	↓ 0.03	54.1	54.2
	GAL[35]	93.26	↓ 1.68	60.2	65.9
	Ours	93.65	↑ 0.08	35.0	41.1
			↓ 0.17	49.6	58.0
			↓ 0.51	56.7	62.2

algorithm. For example, in the case where the FLOPs reduction and the parameters reduction are 49.6% and 58.0%, respectively, the performance of the compact model established by our scheme only decreases by 0.17% in accuracy.

Pruning MobileNet. We design a MobileNet-like neural network with less layers for simplicity. Its primeval structure contains ten blocks with each block including a depth-wise convolutional layer and a point-wise convolutional layer [14]. Since the output channels of depth-wise convolutional layer change as soon as the channel number of its previous point-wise layer changes, we only need to focus on pruning channels in the point-wise convolutional layers. The pruning results are shown in Table 4. Overall, our algorithm can still achieve good performance even for such computationally efficient architecture. For example, when FLOPs and parameters compression ratio increases to 61.3% and 92.9%, respectively, the accuracy loss is only 0.27%.

4.3 Experiments on ImageNet

We adopt a widely studied architecture ResNet-50 as in the previous pruning approaches. Different from general ResNet architecture, ResNet-50 contains a special structure called "bottleneck" [13], which includes three convolutional layers with only the middle layer being expressive in each residual block. Similar to pruning ResNet on CIFAR-10, we focus on pruning the channels of the first two layers in a bottleneck, so that we do not need to worry about the identity mapping when copying the parameters to a compact model. We summarize the experimental results on ILSVRC-12 in Table 5 where we report the performance of both the advanced approaches and ours. It can be observed that the pruned model based on our scheme can reach a comparable accuracy along with significant reduction in both FLOPs and parameters.

Table 4 Results of pruning MobileNet on CIFAR-10

Network	Baseline (%)	Accuracy (%)	FLOPs pruned (%)	Parameters pruned (%)
MobileNet	91.80	↑ 0.07	45.7	87.3
		↓ 0.27	61.3	92.9

Table 5 Results of pruning ResNet-50 on ILSVRC-12

Method	Top-1 baseline (%)	Top-1 Accuracy (%)	Top-5 baseline (%)	Top-5 Accuracy (%)	FLOPs pruned (%)	Parameters pruned (%)
CP[31]	-	-	92.20	↓ 1.40	50.0	-
ThiNet[20]	72.88	↓ 1.87	91.14	↓ 1.12	55.8	51.6
SFP[37]	76.15	↓ 1.54	92.87	↓ 0.81	41.8	-
CFP[32]	-	-	92.20	↓ 0.80	49.6	-
DCP[30]	76.01	↓ 1.06	92.93	↓ 0.61	55.7	51.5
FPGM[45]	76.15	↓ 1.21	92.87	↓ 0.48	42.2	-
PFS[38]	77.20	↓ 1.60	-	-	51.2	57.2
GAL[35]	76.15	↓ 4.20	92.87	↓ 1.93	43.0	16.9
ASS[39]	76.01	↓ 2.49	92.96	↓ 1.45	56.6	56.0
Ours	76.13	↓ 2.38	92.86	↓ 1.09	49.5	66.3

Note-worthily, our method is indeed not as effective as some start-of-the-art algorithms. However, these advanced algorithms have added additional training strategies or enlarged the training time, but our algorithm is very efficient and simple, thus being deployed in a wide range of IoT scenarios.

4.4 Trade-off between performance and compression rate

In practical IoT scenarios, it is necessary to balance the performance and compression rates according to different computing requirements and energy consumption restrictions. On the other hand, showing the performance with various compression rates can also illustrate the robustness and efficiency of an pruning algorithm. Thus in this section, we explore the performance of our scheme upon different pruning rates. For all experiments with different network architectures, we use the same hyper-parameter settings. We summarize the results in Figs. 6, 7, and 8, which corresponds to ResNet, VGG, and MobileNet, respectively.

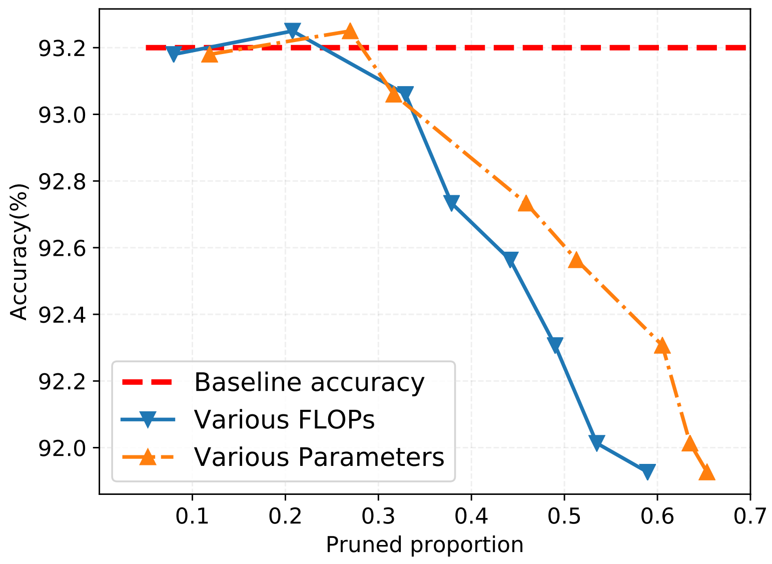


Fig. 6 The pruned results with respect to various pruning rates, which are obtained by pruning ResNet-32 on CIFAR-10

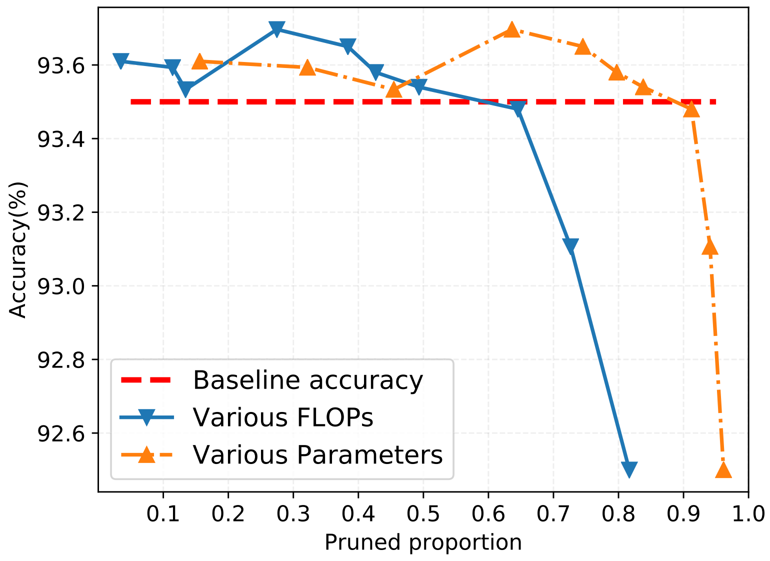


Fig. 7 The results are based on pruning VGG on CIFAR-10 with various pruned proportions in both FLOPs and parameters

As can be observed in Fig. 6, ResNet architecture is sensitive to pruning. When the FLOP reduction proportion increases to 0.6, the performance in accuracy drops by nearly 1.0%. In addition, when pruning VGG and MobileNet, our proposed scheme is more robust in terms of various reduced FLOPs as well as pruned parameters. As depicted in Figs. 7 and 8 with regard to pruning VGG and MobileNet, respectively, our proposed strategy can achieve efficient neural network structures with even higher testing accuracies compared to their baselines at the low level of compression rate for both VGG and MobileNet. Such interesting results also indicate that the performance of compact models

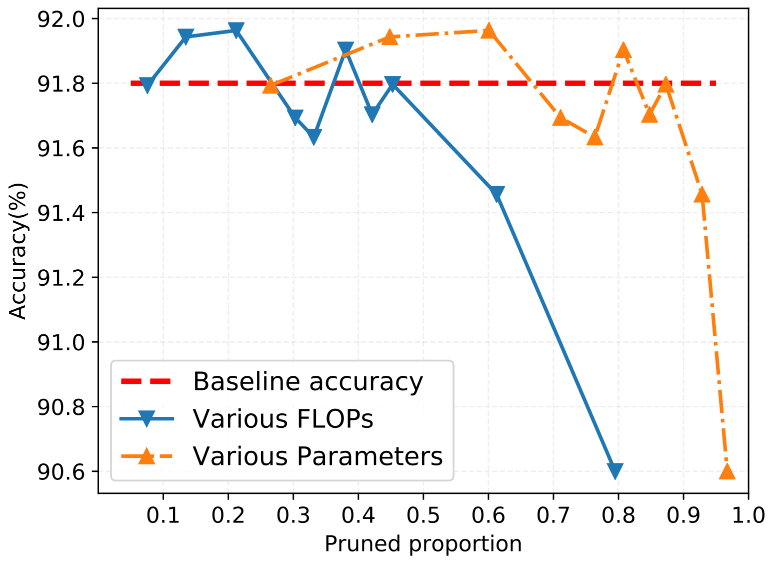


Fig. 8 Pruning MobileNet on CIFAR-10 with the compression-rate varying from 0.0 to 0.9

may outperform that of redundant models to some extent, which implies that the premise of efficient training is to unveil superior neural network with a suitable structure.

4.5 The uncovered compact structures

In this section, we take advantage of the sub-neural-network architectures revealed by our proposed method. Note that a practical problem of deploying DNNs is how to design appropriate lightweight structures to adapt to resource-limited IoT computing tasks, so learning the compact structures can help us design efficient neural networks beyond the state-of-the-art architectures. As seen from Fig. 9, compared with the original deep neural networks with no pruning, our scheme keeps more channels in the middle layers of the designed neural networks while effectively pruning more channels in the last layers and the first layer in the case of pruning VGG on CIFAR-10. The discovered structure suggests that the middle layers are more sensitive whereas the first layer and the last layers are easier to be pruned, which is consistent with the previous findings in [18, 19], indicating the effectiveness of our proposed method.

It can be observed from Fig. 10 that when pruning ResNet on CIFAR-10, the compact model tends to maintain more channels in layers where the number of channels doubles, suggesting those layers are more salient. Similar interesting phenomenon is found when pruning ResNet on ImageNet as well. As depicted in Fig. 11, although the distribution of the pruned channels appears to be disordered to some extent, more channels are still retained in the "turning-point" layers where the number of channels in the original neural network jumps abruptly. Our proposed compact ResNet structure is consistent with the conclusion of sensitivity analysis in [18].

4.6 Acceleration in practice

In this section, we show the running-time acceleration performance of the designed compressive neural networks in practice. We test all compact CNNs on several Intel E5 CPUs

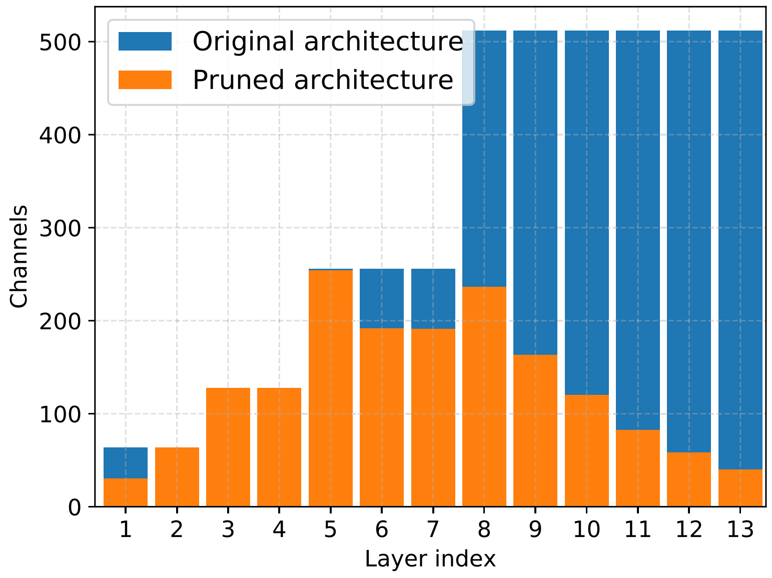


Fig. 9 Channel distribution of the pruned VGG on CIFAR-10. The abscissa indicates the indices of layers and the ordinate indicates the number of reserved channels accordingly

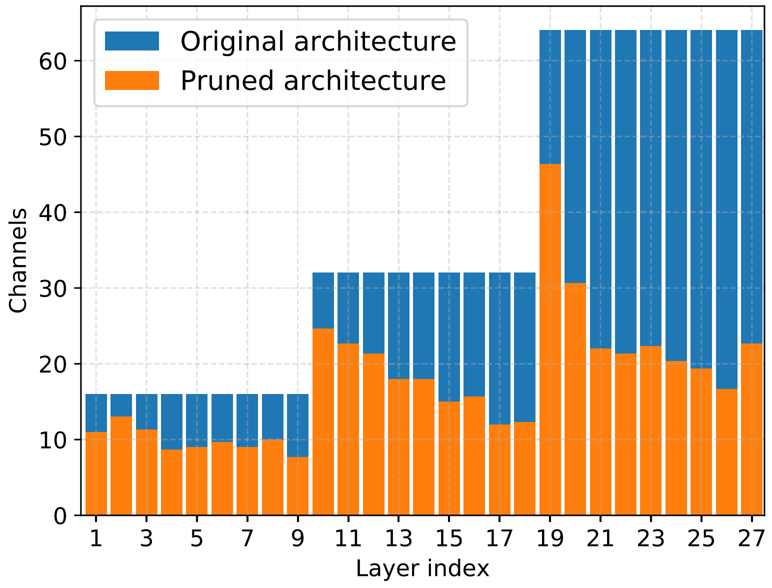


Fig. 10 Channel distribution of the pruned model for ResNet on CIFAR-10. We only present the pruned layers, namely the first layer within each residual block

with the software platform of Pytorch deep learning framework in the operating system of Ubuntu 16.04. Due to the reason that the running time on GPUs is too short to manifest the differences among different methods as well as running on GPUs is not suitable for practical IoT devices, we have not shown the actual acceleration performance on such devices. For each compact neural network, we measure the time of forward propagation for 100 rounds and average them. The overall experimental results are organized in Table 6 where we present both theoretical amount of computation in FLOPs and practical acceleration results.

As shown in Table 6, the test results of each row are obtained by reducing the FLOPs of the corresponding neural network model by 50%, and the practical acceleration

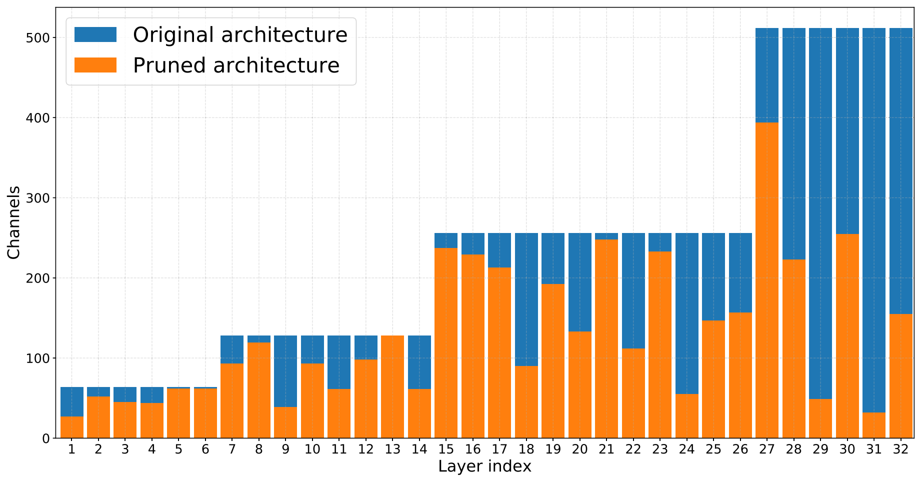


Fig. 11 The architecture of pruned ResNet-50 on ImageNet

Table 6 Theoretical amount of computation in FLOPs and the corresponding acceleration in practice

Archi.	Original FLOPs	Original time (s)	Pruned FLOPs	Pruned time (s)	Acceleration (%)
VGG	3.13×10^8	0.428	1.59×10^8	0.259	39.5
ResNet-56	1.25×10^8	0.256	6.43×10^7	0.215	16.1
MobileNet	2.97×10^7	0.421	1.15×10^7	0.200	52.5
ResNet-50	3.38×10^9	5.453	1.71×10^9	4.098	25.0

performance is consistently effective and impressive for all representative CNN architectures. In addition, the actual acceleration performance of MobileNet is significantly higher than that of both ResNet-50 and ResNet-56, indicating its potential suitability for resource-stringent IoT devices.

4.7 Training time measurement

In fact, one important issue hindering the application of DNN is its complexity in training time. However, our scheme is more efficient as both structure and weight learning are relatively faster in terms of the common training time, especially in the case where initial weights are transferred from post-training models (e.g., inheriting the network parameters from cloud). To be specific, we experiment on one Nvidia RTX-2080 GPU, with the software platform of Pytorch and the dataset of CIFAR-10. Figure 12 provides the performance comparison in terms of the normalized training time of all neural networks. It can be observed from Fig. 12 that the time cost of structure learning is much shorter than that of parameter optimization, which indicates that our scheme is very efficient in finding the compact structures. In addition, the total training time decreases as the pruning rate increases in all experiments, implying our proposed scheme's efficiency as well.

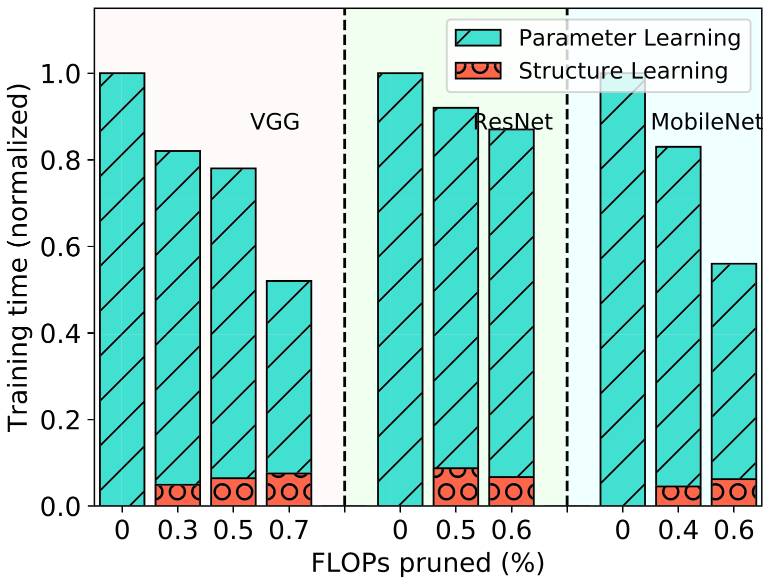


Fig. 12 The normalized training time w.r.t. different pruning rates and CNN architectures. The training time refers to the total time spent in training the neural networks to achieve the highest testing accuracy

5 Conclusions

In this paper, we proposed a novel pruning-based paradigm that aims to apply DNN, especially CNN, to resource-limited IoT scenarios. Our proposed scheme has the capability to train and compress deep neural networks simultaneously. Specifically, we introduce a heuristic algorithm to learn both the architecture and weights of the targeted neural network. Once a compression rate is given, our scheme can train a redundant and randomly initialized neural network into a compact, representative one. A large number of experiments have illustrated the effectiveness of our scheme, which can reduce the complexity of the redundant CNN while maintaining its performance, for example, a satisfying accuracy of 93.27% of the pruned VGG with dramatic reduction in FLOPs and the number of the involved parameters (i.e., 72.6% and 94.1%, respecitvely). In addition, extensive experiments also verify the performance of our scheme regarding various pruning rates in terms of both theoretical acceleration and practical running time reduction.

As mentioned before, our proposed strategy can realize efficient end-to-end training and compression of CNN and is able to be incorporated into the conventional distributed computing paradigm to apply deep learning to resource-limited IoT applications. Moreover, our scheme is lightweight and can be easily extended to other types of DNNs. For future work, we will apply the proposed pruning scheme to actual IoT scenarios to further testify its effectiveness.

Acknowledgements
Not applicable.

Authors' contributions
QC is the major contributor of this paper. She has written most of the sections of the paper and carried out most simulations. SS completes the simulation program and adjusted the hyperparameters of the algorithm. RL is the corresponding author. He participates in discussing the main core content of the paper and approved the submitted manuscript. QL participates in designing the simulation program and revising the final manuscript. JL analyzed the effectiveness of the algorithm and revised the paper. The authors read and approved the final manuscript.

Funding
This work was supported in part by National Key R&D Program of China (No. 2020YFB1804804), National Natural Science Foundation of China (No. 62071425, 61731002), Zhejiang Key Research and Development Plan (No. 2019C01002, 2019C03131), Huawei Cooperation Project, the Project sponsored by Zhejiang Lab (2019LC0AB01), and Zhejiang Provincial Natural Science Foundation of China (No. LY20F010016).

Availability of data and materials
Both CIFAR-10 and ILSVRC-12 data sets are public and can be searched on Google.

Declarations

Competing interests
The authors declare that they have no competing interests.

Author details
[1]College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China. [2]Zhejiang Lab, Hangzhou, China. [3]Huawei Technologies Co. Ltd., Shanghai, China.

Received: 29 January 2021 Accepted: 13 June 2021
Published online: 29 June 2021

References
1. J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, W. Zhao, A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications. IEEE Internet Things J. **4**(5), 1125–1142 (2017)
2. M. A. Al-Garadi, A. Mohamed, A. Al-Ali, X. Du, M. Guizani, A survey of machine and deep learning methods for Internet of Things (IoT) security. arXiv preprint arXiv:1807.11023 (2018)
3. M. Mohammadi, A. Al-Fuqaha, S. Sorour, M. Guizani, Deep learning for IoT big data and streaming analytics: a survey. IEEE Commun. Surv. Tutorials. **20**(4), 2923–2960 (2018)
4. E. Park, Y. Cho, J. Han, S. J. Kwon, Comprehensive approaches to user acceptance of Internet of Things in a smart home environment. IEEE Internet Things J. **4**(6), 2342–2350 (2017)

5. O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, M. N. Hindia, An overview of Internet of Things (IoT) and data analytics in agriculture: Benefits and challenges. IEEE Internet Things J. **5**(5), 3758–3773 (2018)
6. H. Li, K. Ota, M. Dong, Learning IoT in edge: Deep learning for the Internet of Things with edge computing. IEEE Netw. **32**(1), 96–101 (2018)
7. X. Ma, T. Yao, M. Hu, Y. Dong, W. Liu, F. Wang, J. Liu, A survey on deep learning empowered IoT applications. IEEE Access. **7**, 181721–181732 (2019)
8. X. Xie, K.-H. Kim, in *The 25th Annual International Conference on Mobile Computing and Networking*, Source compression with bounded DNN perception loss for IoT edge computer vision (ACM, Los Cabos, 2019), pp. 1–16
9. D. Jia, D. Wei, S. Richard, L. Li-Jia, L. Kai, L. Fei-Fei, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Imagenet: A large-scale hierarchical image database (IEEE, Miami, 2009)
10. A. Krizhevsky, I. Sutskever, G. E. Hinton, in *Advances in Neural Information Processing Systems*, Imagenet classification with deep convolutional neural networks (Curran Associates, Inc., Harrahs and Harveys, Lake Tahoe, 2012), pp. 1097–1105
11. K. Simonyan, A. Zisserman, in *International Conference on Learning Representations (ICLR)*, Very deep convolutional networks for large-scale image recognition (OpenReview.net, San Diego, 2015)
12. C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Going deeper with convolutions (IEEE, Boston, 2015)
13. K. He, X. Zhang, S. Ren, J. Sun, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Deep residual learning for image recognition (IEEE, Las Vegas Nevada, 2016)
14. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, H. Adam, Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861 (2017)
15. S. Han, J. Pool, J. Tran, W. Dally, in *Advances in Neural Information Processing Systems 28*. ed. by C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Learning both weights and connections for efficient neural network (Curran Associates, Inc., Montreal, 2015), pp. 1135–1143
16. S. Han, H. Mao, W. Dally, in *International Conference on Learning Representations (ICLR)*, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding (OpenReview.net, Caribe Hilton, San Juan, 2016)
17. P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, in *International Conference on Learning Representations (ICLR)*, Pruning convolutional neural networks for resource efficient inference (OpenReview.net, Palais des Congreptune, Toulon, 2017)
18. H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, in *International Conference on Learning Representations (ICLR)*, Pruning filters for efficient convnets (OpenReview.net, Palais des Congreptune, Toulon, 2017)
19. Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, in *The IEEE International Conference on Computer Vision (ICCV)*, Learning efficient convolutional networks through network slimming (IEEE, Venice, 2017)
20. J. Luo, J. Wu, W. Lin, in *The IEEE International Conference on Computer Vision (ICCV)*, Thinet: A filter level pruning method for deep neural network compression (IEEE, Venice, 2017)
21. T. Dettmers, L. Zettlemoyer, Sparse networks from scratch: Faster training without losing performance. arXiv preprint arXiv:1907.04840 (2019)
22. H. Mostafa, X. Wang, in *Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol.97*. ed. by K. Chaudhuri, R. Salakhutdinov, Parameter efficient training of deep convolutional neural networks by dynamic sparse reparameterization, (Long Beach, 2019), pp. 4646–4655
23. E. De Coninck, T. Verbelen, B. Vankeirsbilck, S. Bohez, P. Simoens, P. Demeester, B. Dhoedt, in *International Internet of Things Summit*, Distributed neural networks for Internet of Things: The big-little approach (Springer, 2015), pp. 484–492
24. R. Hu, Y. Guo, E. P. Ratazzi, Y. Gong, Differentially private federated learning for resource-constrained Internet of Things. arXiv preprint arXiv:2003.12705 (2020)
25. A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images. Technical Report (2009)
26. H. Amroun, M. H. Temkit, M. Ammi, in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, Best feature for CNN classification of human activity using IoT network (IEEE, Exeter, 2017), pp. 943–950
27. X. Ding, G. Ding, X. Zhou, Y. Guo, J. Han, J. Liu, in *Advances in Neural Information Processing Systems 32*, Global sparse momentum SGD for pruning very deep neural networks (Curran Associates, Inc., Vancouver, 2019)
28. W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, in *Advances in Neural Information Processing Systems 29*, Learning structured sparsity in deep neural networks (Curran Associates, Inc., Vancouver, 2016), pp. 2074–2082
29. S. Ioffe, C. Szegedy, in *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, Batch normalization: Accelerating deep network training by reducing internal covariate shift (ACM, Lille, 2015)
30. Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, J. Zhu, in *Advances in Neural Information Processing Systems 31*, Discrimination-aware channel pruning for deep neural networks (Curran Associates, Inc., Montreal, 2018), pp. 875–886
31. Y. He, X. Zhang, J. Sun, in *The IEEE International Conference on Computer Vision (ICCV)*, Channel pruning for accelerating very deep neural networks (IEEE, Venice, 2017)
32. P. Singh, V. K. Verma, P. Rai, V. P. Namboodiri, Leveraging filter correlations for deep model compression. arXiv e-prints, 1811–10559 (2018)
33. W. Wang, C. Fu, J. Guo, D. Cai, X. He, in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, Cop: Customized deep model compression via regularized correlation-based filter-level pruning (Morgan Kaufmann, Macao, 2019)
34. Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, K.-T. Cheng, J. Sun, in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Metapruning: Meta learning for automatic neural network channel pruning (IEEE, Seoul, 2019), pp. 3296–3305

35. S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, D. Doermann, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Towards optimal structured CNN pruning via generative adversarial learning (IEEE, Long Beach, 2019)
36. Z. Liu, M. Sun, T. Zhou, G. Huang, T. Darrell, in *International Conference on Learning Representations (ICLR)*, Rethinking the value of network pruning (OpenReview.net, New Orleans, 2019)
37. Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, in *IJCAI International Joint Conference on Artificial Intelligence*, Soft filter pruning for accelerating deep convolutional neural networks (Morgan Kaufmann, Stockholm, 2018)
38. Y. Wang, X. Zhang, L. Xie, J. Zhou, H. Su, B. Zhang, X. Hu, Pruning from scratch. arXiv e-prints, 1909–12579 (2019)
39. M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, Y. Tian, Channel pruning via automatic structure search. arXiv e-prints, 2001–08565 (2020)
40. T. Elsken, J. H. Metzen, F. Hutter, Neural architecture search: A survey. J. Mach. Learn. Res. **20**(55), 1–21 (2019)
41. M. Tan, B. Chen, R. Pang, V. Vasudevan, M. Sandler, A. Howard, Q. V. Le, in *2019 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Mnasnet: Platform-aware neural architecture search for mobile (IEEE, Long Beach, 2019)
42. H. Liu, K. Simonyan, Y. Yang, in *International Conference on Learning Representations(ICLR)*, DARTS: Differentiable architecture search (OpenReview.net, New Orleans, 2019)
43. Y. Xu, L. Xie, X. Zhang, X. Chen, G.-J. Qi, Q. Tian, H. Xiong, in *International Conference on Learning Representations(ICLR)*, Pc-darts: Partial channel connections for memory-efficient architecture search (OpenReview.net, Virtual Conference, Formerly Addis Ababa ETHIOPIA, 2020)
44. A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, A. Lerer, in *NIPS-W*, Automatic differentiation in pytorch (Curran Associates, Inc., Long Beach, 2017)
45. Y. He, P. Liu, Z. Wang, Z. Hu, Y. Yang, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Filter pruning via geometric median for deep convolutional neural networks acceleration (IEEE, Long Beach, 2019)

Publisher's Note