Mao *et al. J Wireless Com Network* (2023) 2023:44
https://doi.org/10.1186/s13638-023-02254-3

EURASIP Journal on Wireless Communications and Networking

RESEARCH Open Access

A statistical approach for neural network pruning with application to internet of things

Chengchen Mao[1*], Qilian Liang[1*], Chenyun Pan[1] and Ioannis Schizas[1]

*Correspondence: chengchen.mao@mavs.uta.edu; liang@uta.edu

[1] Department of Electrical Engineering, The University of Texas at Arlington, Arlington, TX, USA

Abstract

Pruning is showing huge potential for compressing and accelerating deep neural networks by eliminating redundant parameters. Along with more terminal chips integrated with AI accelerators for internet of things (IoT) devices, structured pruning is gaining popularity with the edge computing research area. Different from filter pruning and group-wise pruning, stripe-wise pruning (SWP) conducts pruning at the level of stripes in each filter. By introducing filter skeleton (FS) to each stripe, the existing SWP method sets an absolute threshold for the values in FS and removes the stripes whose corresponding values in FS could not meet the threshold. Starting with investigation into the process of stripe wise convolution, we use the statistical properties of the weights located on each stripe to learn the importance between those stripes in a filter and remove stripes with low importance. Our pruned VGG-16 achieves the existing results by a fourfold reduction in parameter with only 0.4% decrease in accuracy. Results from comprehensive experiments on IoT devices are also presented.

Keywords: Prune, Stripe-wise, Edge device, Normal distribution, Internet of things

1 Introduction

In the Internet of Things (IoT) realm, sensors and actuators seamlessly integrate with the environment [1], enabling cross-platform information flow for environmental metrics, while numerous connected devices generate massive data, offering convenience but also high latency [2]. However, applications, such as vehicle-to-vehicle (V2V) communication which enhances the traffic safety by automobile collaboration, are highly latency-sensitive and security-sensitive [3]. Edge computing offers vast potential for consumers and entrepreneurs by bringing data processing closer to end users, enhancing response times, bandwidth availability, privacy, and alleviating information security threats [4, 5].

Even though chip giants are integrating more and more AI accelerators into their design for the IoT devices [6, 7], the massive number of parameters and the huge amount of computation would bring horrible experience to the consumers when Deep Neural Networks (DNNs) are employed in their devices [8]. To alleviate such kind of problems, researchers have made efforts in many directions, which could be mainly categorized into two types: unstructured ones and structured ones.

Pruning the individual weights whose values are close to 0 is one way to downsize the number of parameters in DNNs [9, 10]. This kind of unstructured method could wind up as a sparse structure and maintain the original performance. However, the random and unpredictable positions of the remaining weights bring the burden of extra records of themselves and make this method unable to utilize AI accelerators effectively [11].

By contrast, as shown in Fig. 1, structured methods remove the weights at higher levels and avoid the problem brought by unstructured ones. Filter (channel) pruning (FP)-based methods prunes weights at the level of filters or channels [12–14]. Usually, a traditional FP-based method needs to follow the "Train, Prune, Fine-tune" pipeline. Group-wise pruning-based methods delete the weights at the identical position among all the filters in a certain layer [15]. However, these approaches ignore the assumption of filters' independence. Stripe-wise pruning (SWP)-based methods trim all the weights laid in some stripes of certain filters [16]. The proposed method introduced the concept of filter skeleton (FS). During the training, when some values on FS are under a certain threshold, the corresponding stripes can be pruned.

However setting an absolute threshold sometimes could not express the relative importance of each stripe in a filter. To resolve this problem, in this work, we put forward a new method, using the statistical properties of the weights located on each stripe, to learn the importance between those stripes in a filter. The intuition of this method is triggered by the process during stripe wise convolution and the properties of normal distributions. Our principal contributions in this paper could be summarized as follows:

- New threshold determination approach for SWP: The research proposes a new method for determining which weights in a neural network can be pruned without

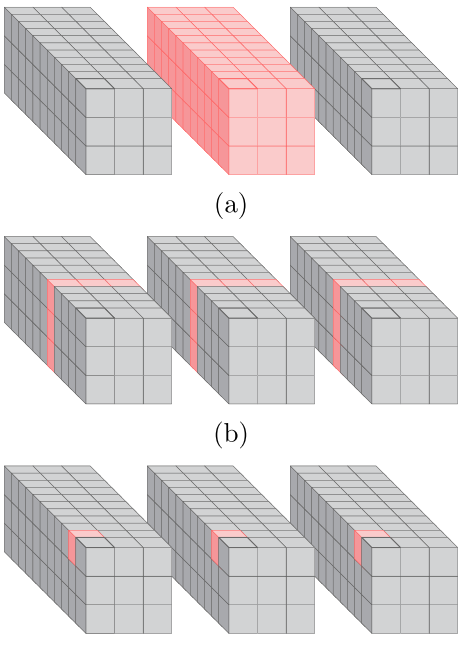

Fig. 1 Different types of pruning. (Red parts were pruned.) **a** Filter-wise. **b** Channel-wise. **c** Group-wise

sacrificing accuracy. Our pruned VGG16 achieves results comparable to the existing model, with a fourfold reduction in parameters and only a 0.4% decrease in accuracy.
- Stable theoretical basis: The proposed method is based on sound theoretical principles, making it more trustworthy and easier to understand and apply.
- Deployment of different deep layers on edge devices: The effectiveness of the proposed approach is tested on different neural network architectures (VGG11, VGG13, VGG16, and ResNet56) and evaluated on edge devices with limited computational resources.

To provide a brief outline of this study, we first review related work in Sect. 2. In Sect. 3, we present our method as well as the theoretical framework behind it. Section 4 explains the experimental details and data processing. In Sect. 5, we demonstrate comparisons between our method and the original method, as well as exhibit the performance of our method deployed on edge devices. In Sect. 6, we discuss the implications of our findings. Finally, concluding remarks are provided in Sect. 7.

2 Related work

Neural network pruning algorithms have undergone decades of research [17]. As mentioned in Sect. , these algorithms could be mainly categorized into two types, i.e., unstructured ones and structured ones.

Unstructured pruning methods prune individual weights based on the importance of themselves. For example, by using the second-order derivatives of the error function, Optimal Brain Damage and Optimal Brain Surgery proposed to remove unimportance weights from a trained network [9, 10]. Deep compression compressed neural networks by pruning the unimportant connections, quantizing the network, and applying Huffman coding [18]. With Taylor expansion that approximates the change in the cost function, [19] pruned convolutional kernels to enable efficient inference and could handle the transfer learning tasks effectively. Lookahead pruning scheme, a magnitude-based method, minimized the Frobenius distortion of multi-layer operation and avoids tuning hyper-parameters [20]. A major downside of the unstructured methods is the sparse matrix and the relative indices after pruning, which leads to the complexity and inefficiency on hardware [11].

Structured methods prune weights in a predictable way [12] pruned unimportant filters with L_1 norm. [21] pruned filters based on statistics information computed from its next layer, not the current layer. [13] pruned channels by LASSO regression. By using scaling factors from batch normalization layers, [14] removed unimportant channels. [15] revisited the idea of brain damage and extended it to group wise, obtaining the sparsities in new neural network. [22] put forward a structured sparsity learning (SSL) approach. With group Lasso regularization, SSL could learn a compressed structure, including filters, channels and filter shapes. To the best of our knowledge, one recent study [16] proposed a stripe-wise pruning-based methods by introducing filter skeleton to learn the shape of filters, and then performed pruning on the stripes according to the corresponding values of the filter skeleton. However, setting an absolute threshold sometimes is unable to distinguish the importance of the convolution result for one stripe from the other result for corresponding stripes.

Table 1 Unstructured and structured pruning methods

Methods	Advantages	Disadvantages
Unstructured [9–11]	Sparse structure original performance	High complexity and inefficiency on hardware
Structured [12–15, 22]	Prune weights in a predictable way	May not maintain unpruned performance

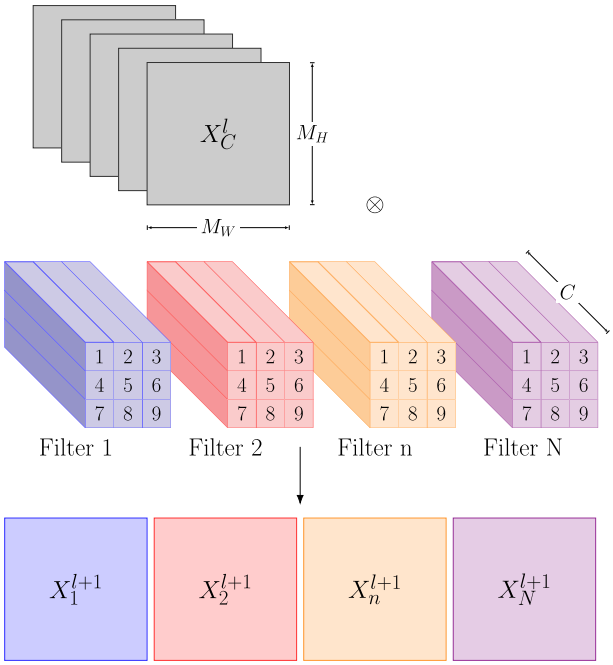

Fig. 2 Standard convolution (The kernel size of the filter is 3)

The comparisons of the two methods are summarized in Table 1.

3 The proposed method

In this section, we begin with introducing stripe wise convolution (SWC), and then analyze our threshold determination with stripe weight combination based on the properties of normal distributions. Furthermore, we discuss our approach for stripe-wise pruning (SWP).

3.1 Stripe wise convolution

In lth convolution layer, suppose the weight 4-D matrix W is of size $\mathbb{R}^{N\times C\times K\times K}$, where N, C and K are the numbers of filters, the channel dimension and the kernel size, respectively.

Let $x^l_{c,h,w}$ be one point of feature map in the lth layer and $x^{l+1}_{n,h,w}$ be the convolution result in the $l+1$th layer. Mathematically, the standard convolution in a neural network could be written as (1). We modify the calculation order as (2) to stripe wise convolution. These two types of convolution are illustrated in Figs. 2 and 3, respectively.

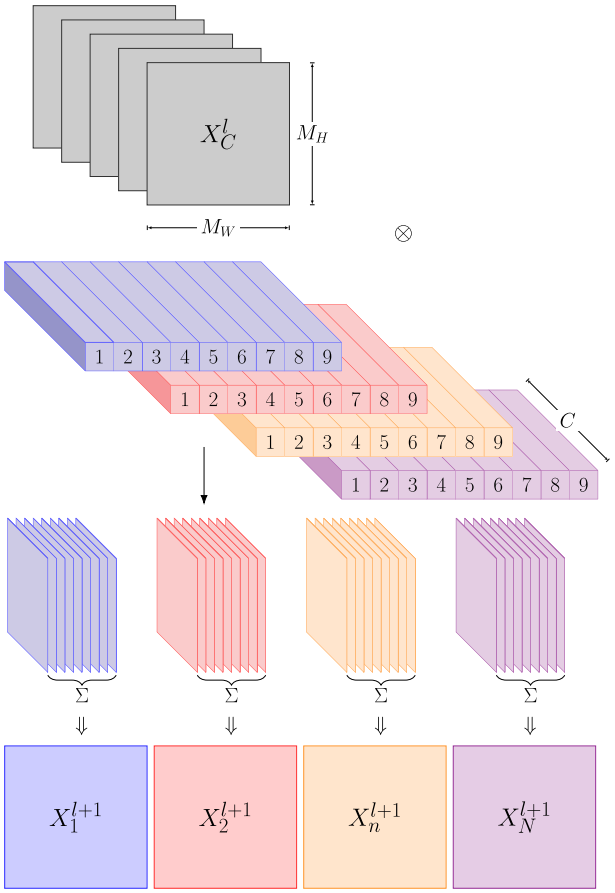

Fig. 3 Stripe wise convolution (The kernel size of the filter is 3)

$$x_{n,h,w}^{l+1} = \sum_{c}^{C}\sum_{i}^{K}\sum_{j}^{K}\left(w_{n,c,i,j}^{l} \times x_{c,h+i-\frac{K+1}{2},w+j-\frac{K+1}{2}}^{l}\right) \tag{1}$$

$$= \sum_{i}^{K}\sum_{j}^{K}\left(\sum_{c}^{C} w_{n,c,i,j}^{l} \times x_{c,h+i-\frac{K+1}{2},w+j-\frac{K+1}{2}}^{l}\right) \tag{2}$$

$$= \sum_{i}^{K}\sum_{j}^{K}(x_{n,h,w,i,j}^{l+1}) \tag{3}$$

$x_{c,p,q}^{l} = 0$, when $p < 1$ or $p > M_H$ or $q < 1$ or $q > M_W$. M_H is the height of the feature map, while M_W represents the width.

From Fig. 3, we could find that in stripe wise convolution, the convolution result of individual filter is the summation of the convolution result of the stripes which belongs to this filter. One intuition is that if the convolution result of the stripe 1 is much smaller than the convolution result of the stripe 2, stripe 1 could be pruned and stripe 2 could be kept as shown in Fig. 4. The following part will prove it in a theoretical manner.

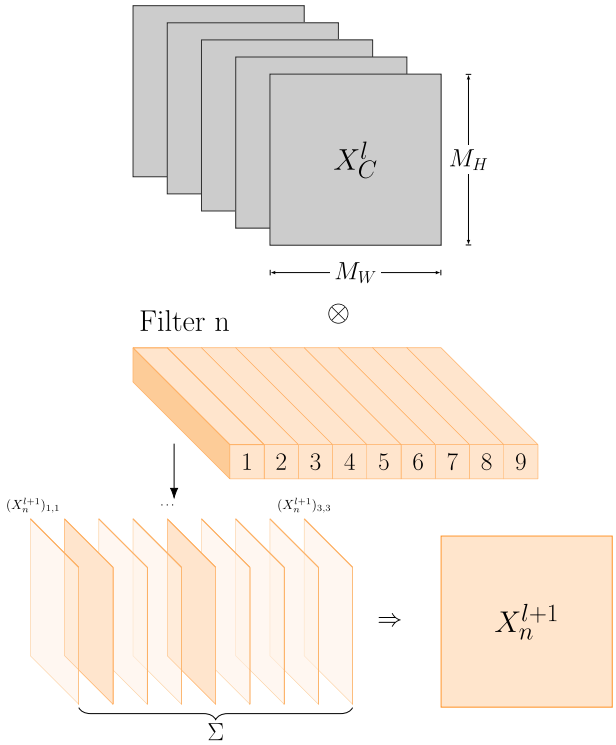

Fig. 4 Stripe wise convolution (Single filter case). The squares with dark orange indicate they have larger stripe convolution results than the light orange ones, which means the corresponding stripes could be remained during pruning

3.2 Theoretical analysis

Batch normalization (BN) is widely used in a neural network. This method could make DNN faster and more stable [23]. In one filter, suppose B is a mini-batch of size m, i.e., $B = \{a_1, ...a_m\}$. BN layer processes these following transformation steps:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} a_i \tag{4}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu_B)^2 \tag{5}$$

$$\hat{a}_i = \frac{a_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{6}$$

$$x_i = \gamma \hat{a}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(a_i) \tag{7}$$

where μ_B and σ_B are the empirical mean and standard deviation of B. To resume the representation ability of the network, scale γ and shift β are learned during the whole process.

After transformation in the BN layer, in cth channel of lth layer, the input feature map could be

$$X_c^l \sim \mathcal{N}(\beta_c^l, (\gamma_c^l)^2). \tag{8}$$

When M_H is large, $(X_c^l)_{i,j} \sim \mathcal{N}(\beta_c^l, (\gamma_c^l)^2)$. From (2), we could get

$$X_n^{l+1} = \sum_i^K \sum_j^K \left(\sum_c^C w_{n,c,i,j}^l \times (X_c^l)_{i,j} \right) \tag{9}$$

Assuming all data is independently identically distribution, with the properties of normal distribution [24], we have

$$X_n^{l+1} \sim \mathcal{N}(\mu_n^{l+1}, (\sigma_n^{l+1})^2) \tag{10}$$

where

$$\mu_n^{l+1} = \sum_i^K \sum_j^K \left(\sum_c^C w_{n,c,i,j}^l \beta_c^l \right) \tag{11}$$

$$(\sigma_n^{l+1})^2 = \sum_i^K \sum_j^K \left(\sum_c^C (w_{n,c,i,j}^l \right)^2 (\gamma_c^l)^2) \tag{12}$$

To reduce the number of parameters $w_{n,c,i,j}^l$ and avoid the value of μ_n^{l+1} in (11) change, we introducing an importance indicator $Q_{n,i,j}^l$ to the output of convolution of each stripe and have the following loss function.

$$L_n = \mathrm{loss}\left(\mu_{\mathrm{n}}^{\mathrm{l+1}}, \sum_{\mathrm{i}}^{\mathrm{K}} \sum_{\mathrm{j}}^{\mathrm{K}} \mathrm{Q}_{\mathrm{n,i,j}}^{\mathrm{l}} \left(\sum_{\mathrm{c}}^{\mathrm{C}} \mathrm{w}_{\mathrm{n,c,i,j}}^{\mathrm{l}} \beta_{\mathrm{c}}^{\mathrm{l}} \right) \right) + \alpha \mathrm{g_n(Q)} \tag{13}$$

where $g_n(Q) = \sum_i^K \sum_j^K \left| Q_{n,i,j}^l \right|, Q_{n,i,j}^l = 1$ or 0.

Let

$$s_{n,i,j}^l \triangleq \sum_c^C w_{n,c,i,j}^l. \tag{14}$$

If we assume $\beta_1^l = \beta_2^l \cdots = \beta_c^l = \beta^l$, combing with (11), (13) could be written as

$$L_n = \mathrm{loss}\left(\beta^{\mathrm{l}} \sum_{\mathrm{a}}^{\mathrm{K}} \sum_{\mathrm{b}}^{\mathrm{K}} \mathrm{s}_{\mathrm{n,a,b}}^{\mathrm{l}}, \beta^{\mathrm{l}} \sum_{\mathrm{i}}^{\mathrm{K}} \sum_{\mathrm{j}}^{\mathrm{K}} \mathrm{Q}_{\mathrm{n,i,j}}^{\mathrm{l}} \mathrm{s}_{\mathrm{n,i,j}}^{\mathrm{l}} \right) + \alpha \mathrm{g_n(Q)} \tag{15}$$

which can be further written as

$$
\begin{aligned}
L_n &= \mathrm{loss}\left(1, \frac{\sum_{\mathrm{i}}^{\mathrm{K}} \sum_{\mathrm{j}}^{\mathrm{K}} \mathrm{Q}_{\mathrm{n,i,j}}^{\mathrm{l}} \mathrm{s}_{\mathrm{n,i,j}}^{\mathrm{l}}}{\sum_{\mathrm{a}}^{\mathrm{K}} \sum_{\mathrm{b}}^{\mathrm{K}} \mathrm{s}_{\mathrm{n,a,b}}^{\mathrm{l}}}\right) + \alpha' \mathrm{g_n(Q)} \\
&= \mathrm{loss}\left(1, \sum_{\mathrm{i}}^{\mathrm{K}} \sum_{\mathrm{j}}^{\mathrm{K}} \mathrm{Q}_{\mathrm{n,i,j}}^{\mathrm{l}} \mathrm{T}_{\mathrm{n,i,j}}^{\mathrm{l}}\right) + \alpha' \mathrm{g_n(Q)}
\end{aligned}
\tag{16}
$$

where

$$
T_{n,i,j}^{l} = \frac{s_{n,i,j}^{l}}{\sum_{a}^{K} \sum_{b}^{K} s_{n,a,b}^{l}} \tag{17}
$$

Obviously,

$$
\sum_{i}^{K} \sum_{j}^{K} T_{n,i,j}^{l} = 1, 0 \leq T_{n,i,j}^{l} < 1 \tag{18}
$$

To minimize (16), we could set $Q_{n,i,j}^{l} = 0$ to those $T_{n,i,j}^{l}$ close to 0, which means the corresponding stripes will be pruned.

$T_{n,i,j}^{l}$ could be used to describe the relative importance of $\mathrm{stripe}_{i,j}$ in filter_{n}. When $T_{n,i,j}^{l} \to 1$, $\mathrm{stripe}_{i,j}$ contributes more than other stripes. When $T_{n,i,j}^{l} \to 0$, $\mathrm{stripe}_{i,j}$ contributes less than other stripes and could be pruned.

3.3 Method description

Before setting a threshold for $T_{n,i,j}^{l}$ to prune stripes, we need to impose regularization on the whole neural network to achieve sparsity. This method could avoid so-called "Train, Prune, Fine-tune" pipeline. The regularization on the FS will be

$$
L = \sum_{(x,y)} \mathrm{loss(f(x, W), y)} + \alpha \mathrm{g(W)} \tag{19}
$$

where α adjusts the degree of regularization. $g(W)$ is L_1 norm penalty on W and could be written as:

$$
g(W) = \sum_{l=1}^{L} \left(\sum_{n=1}^{N} \sum_{c=1}^{C} \sum_{i=1}^{K} \sum_{j=1}^{K} \left| W_{n,c,i,j}^{l} \right| \right) \tag{20}
$$

To avoid using sub-gradient at non-smooth point, instead of the L_1 penalty, we deploy the smooth-L_1 penalty [25].

Summarize the proposed algorithm below.

Algorithm 1 The Proposed Algorithm

The training dataset D, the unpruned neural network model, the regularization factor α, stripe pruning threshold T, the total iteration number i_{total}.
Initialize the FS with all-one matrix.
Initialize the iteration number $i = 0$.
repeat
  Have the stripe wise convolution by (2);
  Update weights W by (19) with the smooth-L_1 penalty;
  Calculate $T^l_{n,i,j}$ by (17);
  Prune stripes with $T^l_{n,i,j}$ smaller than T;
  $i = i + 1$;
until $i = i_{\text{total}}$;
Output the pruned model;

3.4 Computational complexity

Assuming the number of weight parameters to be P, the computational cost for implementing the smooth-L_1 penalty is $O(P)$. Assuming that the computational cost of each epoch of neural network training is $O(Q)$, the process of pruning costs $O(e \cdot Q)$ computations, where e denotes the percentage of pruning in the neural network of an epoch. Therefore, the computational cost for each epoch is $O(P + e \cdot Q)$. If K denotes the total number of iterations, the computational complexity of our method is $O(K(P + e \cdot Q))$.

4 Experiments

In order to assess the performance of the proposed model and confirm its effectiveness, we carry out experiments on two datasets including CIFAR-10 and bearings dataset from the Case Western Reserve University.

4.1 Experiments on CIFAR-10

4.1.1 Implementation details

Our method is implemented using the publicly available Torch [26].

Dataset and Model: CIFAR-10 [27] is one of the most popular image collection data sets. This dataset contains 60K color images from 10 different classes. 50K and 10K images are included in the training and testing sets respectively. By adopting CIFAR-10, we evaluated the proposed method mainly on VGG [28] and ResNet56 [29]. VGG16 and ResNet56 are the networks used to demonstrate the performance before and after network pruning. VGG11 and VGG13, which have sizes more compact than VGG16, are then deployed to make comparisons with VGG16 in terms of the total time which is required for classifying 3270 image patches of size 224×224, i.e., inference time.

Baseline Setting: We train the model using mini-batch size of 64 for 100 epochs. The initial learning rate is set to 0.1, and is divided by 10 at the epoch 50. Random crop and random horizontal flip are used as data augmentation for training images. Image is scaled to 256×256. Then, a 224×224 part is randomly cropped from the scaled image for training. The testing is the center crop with 224×224.

Experiment environment: NVidia 1080-TI and Intel Core i5-8500B are selected as two different computing platforms representatives of the server and the edge device, respectively. The first is a GPU which has high computation ability, however needs

communication with sensors and actuators. The second is a CPU to represent the restricted computer power of an edge device.

4.2 Experiments on bearings dataset

Rotating element bearings (REBs) are among the most common parts in rotating equipment, and their malfunction is a leading cause of machinery failure. The Case Western Reserve University (CWRU) Bearing Data Center's dataset has emerged as a benchmark in the domain of bearing diagnostics [30]. We will use the data from this center for the following part of the experiment. The fundamental configuration of the testing apparatus is displayed in Fig. 5.

The testing setup includes a 2 hp reliance electric motor that powers a shaft equipped with a torque transducer and encoder. Torque is exerted on the shaft via a dynamometer and electronic control system.

4.2.1 Symmetrized dot pattern

Symmetrized dot pattern (SDP) is a technique used for visual representation of acoustic and vibration signals to quickly identify any faulty condition of the system [31]. This technique transforms the time-domain signal into a scatter plot with sextuple symmetry.

The time-domain signal is $S = \{s_1, s_2..., s_i..., s_D\}$, s_i is the ith sampling point. Then, s_i could be transformed into its corresponding polar coordinate space $P(r(i), \theta(i), \phi(i))$.

$P(r(i))$ is the radius, which could be expressed as follows:

$$r(i) = \frac{s_i - s_{\min}}{s_{\max} - s_{\min}} \tag{21}$$

where $s_{\max}$ and $s_{\min}$ are the maximum and minimum amplitudes values of the time-domain signal sequence, respectively.

$\theta(i)$ is the clockwise rotation angle of the initial line, while $\phi(i)$ represents the counterclockwise one. These angles could be expressed as follows:

$$\theta(i) = \phi - \frac{s_i - s_{\min}}{s_{\max} - s_{\min}}\zeta \tag{22}$$

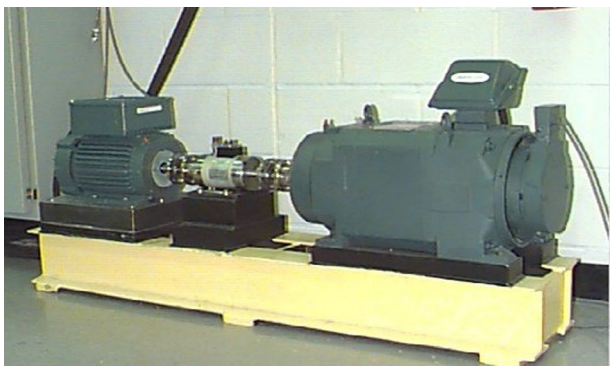

Fig. 5 CWRU bearing test rig. (The components of the test stand include a 2 hp motor on the left, a torque transducer/encoder in the center, and a dynamometer on the right)

$$\phi(i) = \phi + \frac{s_i - s_{\min}}{s_{\max} - s_{\min}}\zeta \tag{23}$$

where ϕ is the initial rotation angle ($\phi = 60\,\mathrm{m} + \phi_0, m = 1, ..., 6, \phi_0$ is a starting term that can rotate the plot). ζ is the amplification coefficient.

The time-domain signal S can be transformed into its corresponding polar plot by marking all θ points as red and all ϕ points as blue, as shown in Fig. 6.

4.2.2 Data categories and preprocess

Four different operational conditions were tested at varying bearing loads (0–3 hp) to collect vibration signals. Each operational condition contains datasets representing rolling element faults and inner and outer race faults. In addition to the normal condition, the SDP for these three operating faults are shown in Fig. 7. The outer race faults are further classified into three categories based on their location relative to the load zone: 'centered' (fault at 6 o'clock position), 'orthogonal' (fault at 3 o'clock position), and 'opposite' (fault at 12 o'clock position). Combining with fault size (0.007 to 0.028 in.), we choose twelve fault categories as shown in Table 2. Adding one healthy state resulted in a total of thirteen different bearing categories.

In order to test the performance of the proposed framework under various working environments, several sub-datasets are created as follows.

- Training data and testing data are both from vibration signals under the same working load.
  - e.g., Training data and testing data are both from vibration signals under working load of 0 hp.
- Training data come from vibration signals under a certain working load while testing data are from different working loads.
  - e.g., Training data come from vibration signals under working load 0 hp while testing data are from working load of 3 hp.

Therefore, there are 16 sub-datasets, with each bearing sub-dataset treated as a 13-class classification task for fault diagnosis. To facilitate the demonstration, we use Dij to

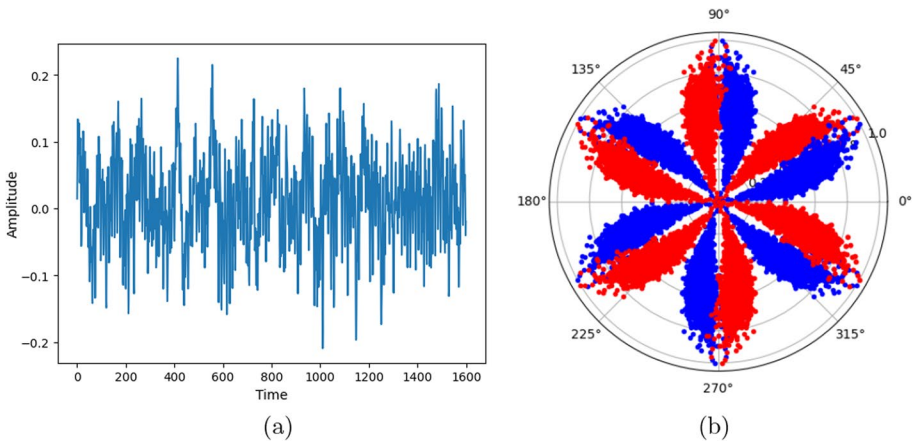

Fig. 6 Typical time-domain signal for SDP technique (**a**) and corresponding SDP plot (**b**)

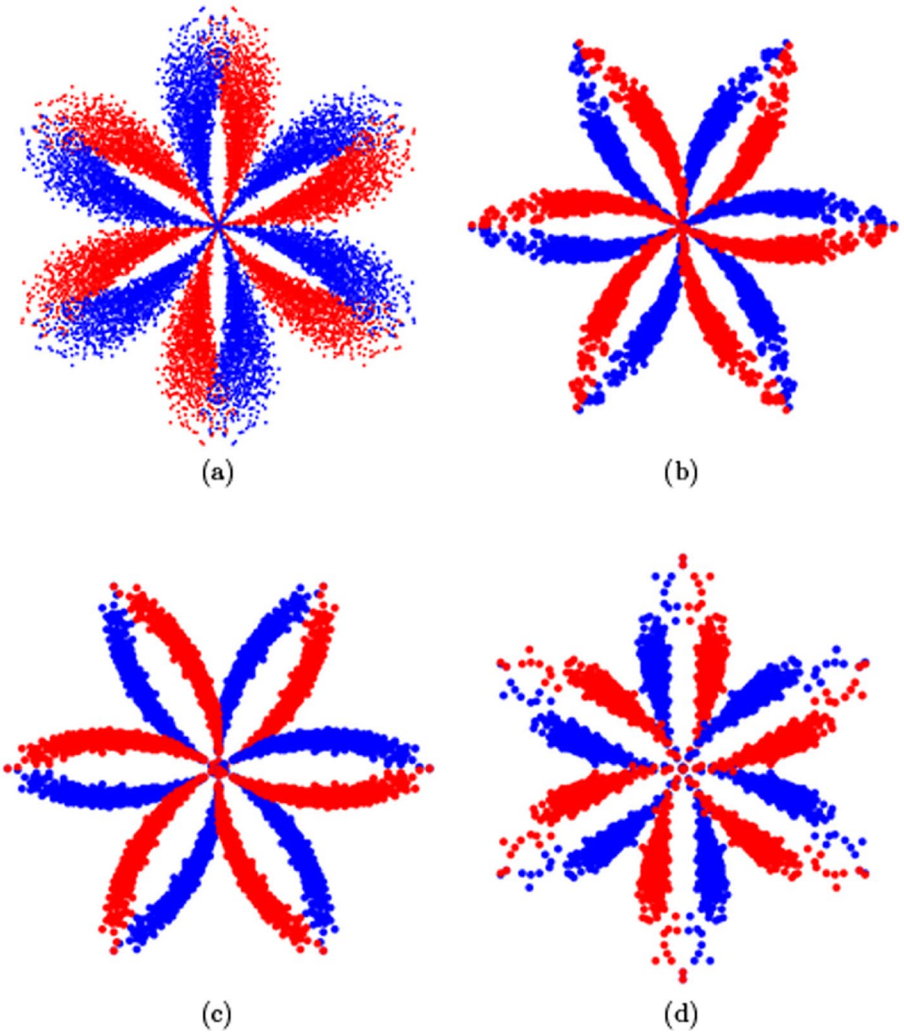

Fig. 7 SDP **a** Normal case. **b** Inner race fault. **c** Outer race fault. **d** Rolling element fault

Table 2 Twelve fault categories

Fault diameter	Inner race	Ball	Outer race position relative to load zone		
			centered @6	orthogonal @3	opposite @12
0.007″	IR007	B007	OR007@6	OR007@3	OR007@12
0.014″	IR014	B014	–	–	–
0.021″	IR021	B021	OR021@6	OR021@3	OR021@12

represent a sub-dataset, where the training set is derived from the workload of i hp, and the testing set is derived from the workload of j hp. Each sub-dataset consists of 500 samples for each machine state, resulting in a total of 6500 samples for the 13 classes. Each sample is derived from a randomly cropped 1600-length time-domain signal, and augmented by rotating the image using a random value of ϕ_0 in the SDP transformation to expand the dataset.

The following steps are the same as those used with CIFAR-10.

5 Results

5.1 Results on on CIFAR-10

5.1.1 Comparing with the original SWP

To compare our method with the original SWP, we revisit the concept of filter skeleton (FS) from [16]. As mentioned before, in lth layer, the weight W is of size $\mathbb{R}^{N\times C\times K\times K}$. Then, the size of FS in this layer is $\mathbb{R}^{N\times K\times K}$. Each value in FS corresponds to a stripe in the filter. During training, the filters' weights are multiplied with FS. With I representing the FS, the stripe wise convolution could be written as

$$x^{l+1}_{n,h,w} = \sum_{i}^{K}\sum_{j}^{K} I^{l}_{n,i,j}\left(\sum_{c}^{C} w^{l}_{n,c,i,j} \times x^{l}_{c,h+i-\frac{K+1}{2},w+j-\frac{K+1}{2}}\right) \tag{24}$$

where $I^{l}_{n,i,j}$ is initialized with 1.

The regularization on the FS will be

$$L = \sum_{(x,y)} \mathrm{loss}(\mathrm{f}(\mathrm{x}, \mathrm{W} \odot \mathrm{I}), \mathrm{y}) + \alpha \mathrm{g}(\mathrm{I}) \tag{25}$$

where $\odot$ denotes dot product and α adjusts the degree of regularization. $g(I)$ is written as:

$$g(I) = \sum_{l=1}^{L}\left(\sum_{n=1}^{N}\sum_{i=1}^{K}\sum_{j=1}^{K}\left|I^{l}_{n,i,j}\right|\right) \tag{26}$$

For convenience, in Table 3 for the comparison on CIFAR-10, both the original method and our method use FS to train and prune the whole neural network. Both of them use the coefficient α of regularization, which is set to 1e−5 and 5e−5. The difference is that for the original method, pruning is based on the value in FS which corresponds to a stripe and for our method, pruning is based on $T^{l}_{n,i,j}$ which combines the weights located in a stripe. Regarding the choice of T, we used the value corresponding to the highest accuracy.

From the table, we could find both methods could reduce the number of parameters and the amount of computation (FLOPs) in a considerable volume without losing network performance. For the backbone is VGG16 situation, when $\alpha = 1\mathrm{e}{-5}$, the number of parameters and the amount of computation of our method are larger than the original

Table 3 Comparison with the original SWP on CIFAR-10

Backbone	Metrics	Params	FLOPS	Accuracy
VGG16	Baseline	14.76M	627.37 M	93.76%
	Original ($\alpha = 1\mathrm{e}{-5}$)	3.62M	350.28 M	93.46%
	Original ($\alpha = 5\mathrm{e}{-5}$)	could not converge		
	Ours ($\alpha = 1\mathrm{e}{-5}, T = 0.0001$)	4.63 M	385.49 M	93.43%
	Ours ($\alpha = 5\mathrm{e}{-5}, T = 0.005$)	0.84 M	126.03 M	93.06%
ResNet56	Baseline	0.87M	251.50 M	93.11%
	Original ($\alpha = 1\mathrm{e}{-5}$)	0.60M	150.63 M	93.41%
	Ours ($\alpha = 5\mathrm{e}{-5}, T = 0.001$)	0.23 M	60.76 M	92.96%

Table 4 Different coefficient α and weight combination threshold

α	1e−5			5e−5			
T	0.0001	0.001	0.01	0.0001	0.0005	0.001	0.005
Params (M)	4.63	4,17	2.89	0.78	0.80	0.79	0.84
FLOPS (M)	385.49	327.17	200.09	130.96	135.56	134.68	126.03
Accuracy (%)	93.43	93.28	92.99	92.79	92.86	92.96	93.06

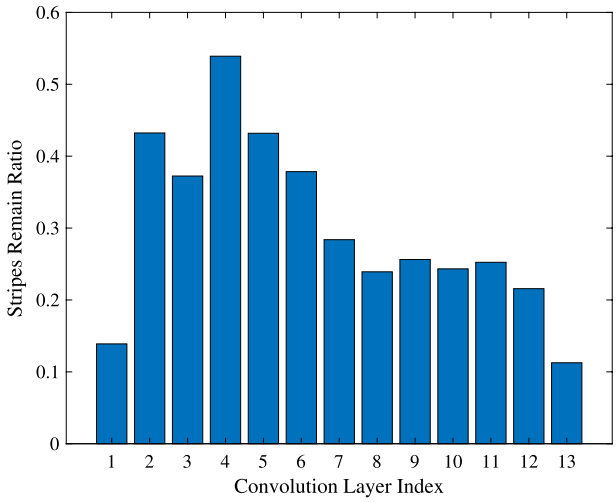

Fig. 8 The ratio of remaining stripes in each layer

approach. This is because our method will keep at least one stripe in a filter, while the original approach might prune a whole filter. However, when $\alpha = 5\text{e}{-5}$, the original approach could not converge and our method could reach a high compression rate both in the number of parameters and the amount of computation. Our pruned VGG16 could achieve 95% reduction in memory demands.

For the backbone is Resnet56 situation, we present our result of $\alpha = 5\text{e}{-5}$. To compare with the original approach's result of $\alpha = 1\text{e}{-5}$, our method could see a large reduction in the number of parameters and the amount of computation while sacrificing a bit of accuracy. Our pruned Resnet56 could achieve 75% reduction in memory demands.

5.1.2 Ablation study

In our method, there are two decisive hyper-parameters in the neural network, the coefficient α of regularization in (25) and the weight combination threshold T in (17). As the outcomes of the experiment demonstrated in Table 4, we display the effects of the hyper-parameters in pruning consequences. It could be noticed that $\alpha = 5e - 5$ and $T = 0.005$ holds an acceptable pruning ratio as well as test accuracy.

In Fig. 8, we show how many stripes in VGG16 on CIFAR-10 are kept after SWP. We could find that most stripes (around 85%) in the first and the last layers are pruned. There are higher pruning ratios in the second half layers than the first half ones. The pruning ratio for this VGG16 neural network is 26.25%.

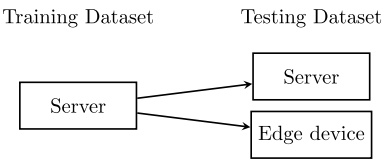

Fig. 9 Experiment setup of edge device performance

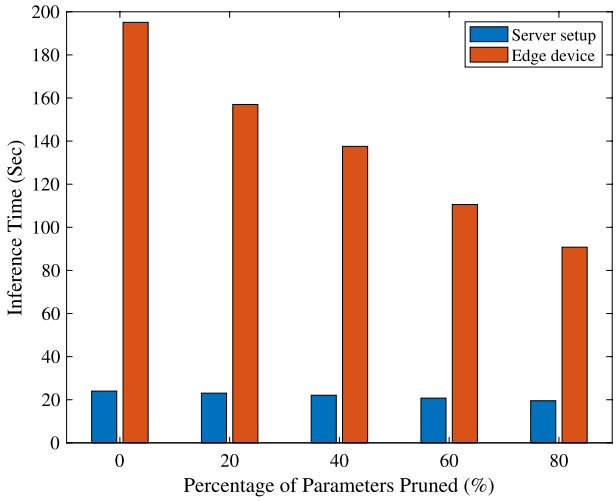

Fig. 10 Inference time required for pruned VGG16

5.1.3 Edge device performance

We further verify our approach in an edge device. As shown in Fig. 9, pruning is executed on the server as training consumes computing resources on learning the importance between the stripes and serval complete passes of the training dataset through the whole neural network. The pruned networks are then deployed on these two computing platforms to test results and get the inference time. The comparison is shown in Figs. 10, 11, 12 and 13. It should be noted that stripe wise convolution is not yet optimized in CUDA. Along with the increase in percentage of parameters pruned, the decline in inference time in servers is not quite clear. However, the inference time in edge device drops by half when 75–95% of parameters are pruned.

Separately speaking, Fig. 10 shows the variation of the inference time when using pruned VGG16. On edge device, the inference time decreases from 195.1 (s) to 77.9 (s). For the server setup, the inference time decreases from 24.01 (s) to 19.53 (s). Figures 11 and 12 show the variation of the inference time with pruned VGG13 and VGG11, respectively. Similar results of decrease in inference time could be observed on both VGG structures.

Figure 13 reports the results of using pruned ResNet56. On edge device, the inference time decreases from 277.7 (s) to 100.5 (s). For the server setup, the inference time decreases from 61.9 (s) to 54.9 (s).

Figure 14 compares the inference time of 3 types of VGG as well as ResNet56 when deployed on the edge device. Due to the reduction in the number of layers, regardless

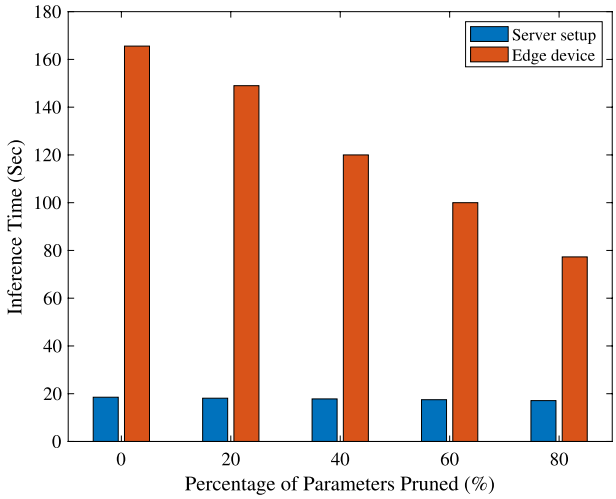

Fig. 11 Inference time required for pruned VGG13

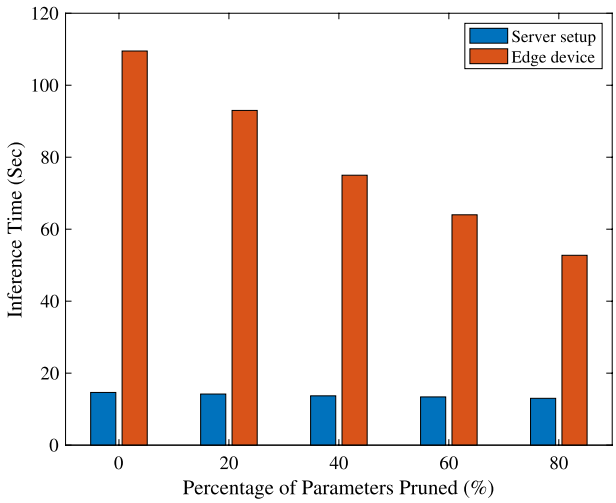

Fig. 12 Inference time required for pruned VGG11

of parameters pruned percentage, VGG11 could classify the most images at the same time in all pruned models, while ResNet56 classifies the least.

It could also be found in Table 3, in terms of memory requirement, the percentage reduction in parameters for our pruned ResNet56 is also not great as pruned VGG16.

5.2 Results on bearings dataset

5.2.1 Comparison of the original model and the pruned model

Without loss of generality, we choose VGG16 as the backbone and set the coefficient α to 5e−5 and the weight combination threshold T to 0.005 in the pruned model. The accuracy results of the classification are shown in Table 5. It can be observed that both the original and pruned models achieved good accuracy rates in each of the sub-datasets.

In Table 6, we compare the performance on the original method and our method. The sub-dataset used is D03 and the backbone is VGG16. It is noticeable that pruned models

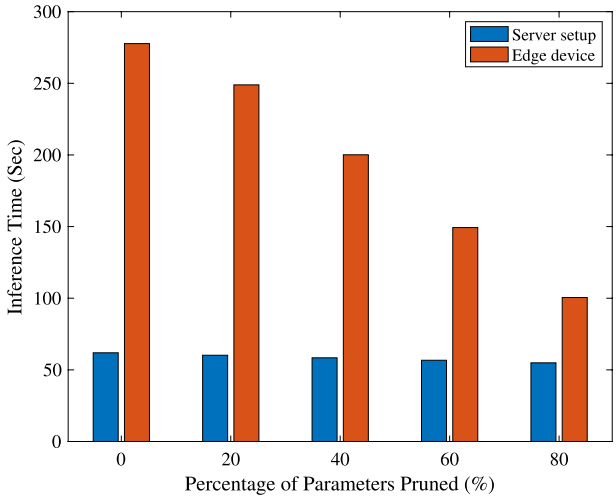

Fig. 13 Inference time required for pruned ResNet56

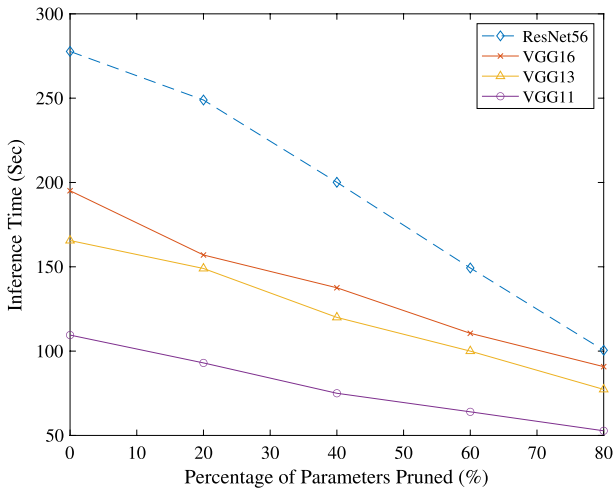

Fig. 14 Inference time required for different backbone models

are capable of reducing the number of parameters and FLOPs significantly, without compromising network performance.

5.2.2 Edge device performance

We also verify our approach in the edge device. The experimental setup and procedures are identical to that used for the CIFAR-10 dataset in the previous part. The sub-dataset used is D12 and the backbone is VGG16. The results are presented in Fig. 15, which shows that the inference time of the pruned model did not significantly change in terms of edge device performance.

Table 5 Comparison of accuracy under different sub-datasets

Sub-dataset	Accuracy
(a) Original model	
D00	0.94385
D01	0.99277
D02	0.92292
D03	0.97400
D10	0.98923
D11	0.96615
D12	0.97908
D13	0.98846
D20	0.98031
D21	0.98308
D22	0.97077
D23	0.98892
D30	0.98600
D31	0.98092
D32	0.98877
D33	0.96462
Pruned model	
D00	0.97846
D01	0.99323
D02	0.98800
D03	0.97246
D10	0.98785
D11	0.95077
D12	0.98554
D13	0.99031
D20	0.97477
D21	0.98600
D22	0.97000
D23	0.99031
D30	0.98923
D31	0.98062
D32	0.97754
D33	0.97615

Table 6 Comparison with the original SWP on D03

Backbone	Metrics	Params	FLOPS	Accuracy
VGG16	Baseline	14.76 M	627.37 M	97.40%
	Original ($\alpha = 1e-5$)	3.32 M	341.65 M	97.35%
	Ours ($\alpha = 1e-5, T = 0.0001$)	4.43 M	375.68 M	97.43%

6 Discussion

Our pruned VGG16 achieves results comparable to the existing model, with a fourfold reduction in parameters and only a 0.4% decrease in accuracy. When deployed on the edge device, the inference time in the pruned network could drop by half. In addition

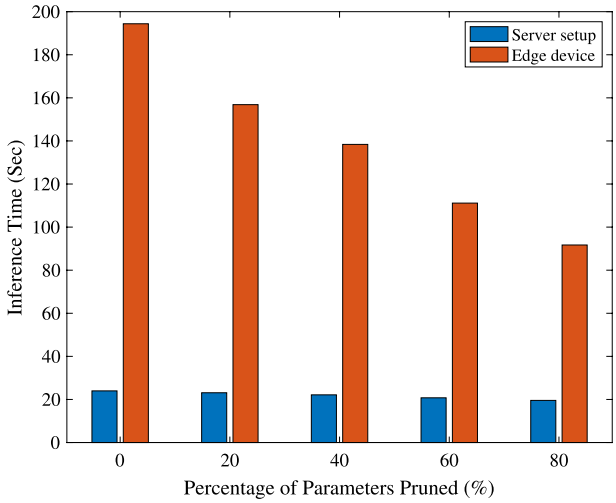

Fig. 15 Inference time required for pruned VGG16

to validating our pruned model on CIFAR-10, we also tested our model on the widely used bearing dataset from the Case Western Reserve University (CWRU) Bearing Data Center. The accuracy and the inference time are similar to those when using CIFAR-10. To the best of our knowledge, this is the first time that a stripe-wise pruning algorithm has been applied to the edge devices and Bearing datasets. However, due to the relatively small size of the CIFAR-10 and Bearing Data Center datasets, the pruned model may have limitations. We need to validate our theories on larger datasets.

7 Conclusion

In this work, we avoid using an absolute threshold in existing stripe-wise pruning by combining the weights located on each stripe. This allows us to learn the importance between stripes in a filter and remove those with low importance. Our pruned method effectively reduces the parameters and inference time of our VGG16 model without significantly impacting accuracy. In future work, we will explore the introduction of regularizers to prune filters with single stripes, which may further compress deep neural networks and improve performance.

Abbreviations

BN	Batch normalization
CIFAR	Canadian Institute for Advanced Research
CWRU	Case Western Reserve University
DNNs	Deep neural networks
FLOP	FLoating-point operations per second
FS	Filter skeleton
IoT	Internet of things
LASSO	Least absolute shrinkage and selection operator
REBs	Rotating element bearings
SDP	Symmetrized dot pattern
SWC	Stripe wise convolution
SSL	Structured sparsity learning
SWP	Stripe-wise pruning
V2V	Vehicle-to-vehicle
GG	Visual geometry group

Acknowledgements
Not applicable.

Author' contributions
CM proposed the algorithm, and all authors helped revising the algorithm and designing the experiment. All authors read and approved the final manuscript.

Funding
This work was supported by U.S. National Science Foundation (NSF) under Grant CCF-2219753.

Availability of data and materials
This study utilized two datasets: CIFAR-10 and the Bearings Dataset from The Case Western Reserve University. The CIFAR-10 dataset is publicly available and can be accessed and downloaded from the official website of the Canadian Institute for Advanced Research (CIFAR) at https://www.cs.toronto.edu/~kriz/cifar.html. The Bearings Dataset from The Case Western Reserve University is also publicly accessible. It can be obtained from the Bearing Data Center's website at https://engineering.case.edu/bearingdatacenter/download-data-file.

Declarations

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

Received: 14 April 2023 Accepted: 17 May 2023
Published online: 27 May 2023

References

1. J. Gubbi, R. Buyya, S. Marusic, M. Palaniswami, Internet of things (IoT): a vision, architectural elements, and future directions. Futur. Gener. Comput. Syst. **29**(7), 1645–1660 (2013)
2. P. Schulz, M. Matthe, H. Klessig, M. Simsek, G. Fettweis, J. Ansari, S.A. Ashraf, B. Almeroth, J. Voigt, I. Riedel et al., Latency critical IoT applications in 5g: perspective on the design of radio interface and network architecture. IEEE Commun. Mag. **55**(2), 70–78 (2017)
3. J. Mei, K. Zheng, L. Zhao, Y. Teng, X. Wang, A latency and reliability guaranteed resource allocation scheme for LTE V2V communication systems. IEEE Trans. Wirel. Commun. **17**(6), 3850–3860 (2018)
4. W. Yu, F. Liang, X. He, W.G. Hatcher, C. Lu, J. Lin, X. Yang, A survey on the edge computing for the internet of things. IEEE Access **6**, 6900–6919 (2018). https://doi.org/10.1109/ACCESS.2017.2778504
5. W. Shi, J. Cao, Q. Zhang, Y. Li, L. Xu, Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
6. J. Tang, D. Sun, S. Liu, J.-L. Gaudiot, Enabling deep learning on IoT devices. Computer **50**(10), 92–96 (2017)
7. S. Hooker, The hardware lottery. Commun. ACM **64**, 58–65 (2021)
8. 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)
9. Y. LeCun, J.S. Denker, S.A. Solla, Optimal brain damage, in *Advances in Neural Information Processing Systems* (1990), pp. 598–605
10. C.M. Bishop et al., *Neural Networks for Pattern Recognition* (Oxford University Press, Oxford, 1995)
11. S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: efficient inference engine on compressed deep neural network. ACM SIGARCH Comput. Archit. News **44**(3), 243–254 (2016)
12. H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710 (2016)
13. Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 1389–1397
14. Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 2736–2744
15. V. Lebedev, V. Lempitsky, Fast convnets using group-wise brain damage, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 2554–2564
16. F. Meng, H. Cheng, K. Li, H. Luo, X. Guo, G. Lu, X. Sun, Pruning filter in filter. Adv. Neural. Inf. Process. Syst. **33**, 17629–17640 (2020)
17. R. Reed, Pruning algorithms—a survey. IEEE Trans. Neural Netw. **4**(5), 740–747 (1993)
18. S. Han, H. Mao, W.J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv preprint arXiv:1510.00149 (2015)
19. P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440 (2016)
20. S. Park, J. Lee, S. Mo, J. Shin, Lookahead: A far-sighted alternative of magnitude-based pruning. arXiv preprint arXiv:2002.04809 (2020)
21. J.-H. Luo, J. Wu, W. Lin, Thinet: a filter level pruning method for deep neural network compression, in *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 5058–5066
22. W. Wen, C. Wu, Y. Wang, Y. Chen, H. Li, Learning structured sparsity in deep neural networks. Adv. Neural. Inf. Process. Syst. **29**, 2074–2082 (2016)
23. S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, in *International Conference on Machine Learning* (PMLR, 2015), pp. 448–456
24. W. Feller, *An Introduction to Probability Theory and Its Applications*, vol. 2 (Wiley, New York, 2008)

25. M. Schmidt, G. Fung, R. Rosales, Fast optimization methods for l1 regularization: a comparative study and two new approaches, in *European Conference on Machine Learning* (Springer, 2007), pp. 286–297
26. R. Collobert, K. Kavukcuoglu, C. Farabet, Torch7: a matlab-like environment for machine learning, in *BigLearn* (NIPS Workshop, 2021)
27. A. Krizhevsky, G. Hinton, Learning multiple layers of features from tiny images. Technical report, University of Toronto, Toronto, Ontario (2009)
28. K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv: 1409.1556 (2014)
29. K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2016), pp. 770–778
30. K. Loparo, Case western reserve university bearing data center. Bearings Vibration Data Sets, Case Western Reserve University, pp. 22–28 (2012)
31. C.A. Pickover, On the use of symmetrized dot patterns for the visual characterization of speech waveforms and other sampled data. J. Acoust. Soc. Am. **80**(3), 955–960 (1986)

Publisher's Note