Advancing Internet of Things Through Statistical Pruning of Neural Networks

Chengchen Mao, Qilian Liang, Chenyun Pan, and Ioannis Schizas

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 4-fold reduction in parameter with only 0.4% decrease of 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. However, applications such as vehicle-to-vehicle communication which enhances the traffic safety by automobile collaboration, require low latency and high security. Edge computing is a promising technology that has the potential to improve the performance and security of IoT applications [2].

C. Mao (✉) · Q. Liang · C. Pan · I. Schizas
Department of Electrical Engineering, The University of Texas at Arlington, 701 S. Nedderman Drive, Arlington, TX 76019, USA
e-mail: chengchen.mao@mavs.uta.edu

W. Wang et al. (eds.), *Communications, Signal Processing, and Systems*, Lecture Notes in Electrical Engineering 1032, https://doi.org/10.1007/978-981-99-7505-1_35

Even though chip giants are integrating more and more AI accelerators into their design for the IoT devices, 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 [3]. 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.

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 [4]. Deep Compression compressed neural networks by pruning the unimportant connections, quantizing the network, and applying Huffman coding [5]. With Taylor expansion that approximates the change in the cost function [6], pruned convolutional kernels to enable efficient inference and could handle the transfer learning tasks effectively. 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 [7].

Structured methods prune weights in a predictable way. Li et al. [8] pruned unimportant filters with L_1 norm. Luo et al. [9] pruned filters based on statistics information computed from its next layer, not the current layer. He et al. [10] pruned channels by LASSO regression. By using scaling factors from batch normalization layers, [11] removed unimportant channels. Lebedev and Lempitsky [12] revisited the idea of brain damage and extended it to group wise, obtaining the sparsities in new neural network. To the best of our knowledge, one recent study [13] 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 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: (1) The research proposes a new method for determining which weights in a neural network can be pruned without 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. (2) The proposed method is based on sound theoretical principles, making it more trustworthy and easier to understand and apply. (3) The effectiveness of the proposed approach is tested on different neural network architectures (VGG16 and ResNet56) and evaluated on edge devices with limited computational resources.

The paper is arranged as follows: In Sect. 2, we present our method as well as the theoretical framework behind it. In Sect. 3, we explain the experimental details and demonstrate comparisons between our method and the original method. Additionally, we showcase the performance of our method deployed on edge devices. Finally, concluding remarks are provided in Sect. 4.

2 The Proposed Method

2.1 Stripe Wise Convolution

In l-th 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 l-th layer and $x^{l+1}_{n,h,w}$ be the convolution result in the $l+1$-th layer. We modify the calculation order of standard convolution in the stripe-wise way (1) as illustrating in Fig. 1a.

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

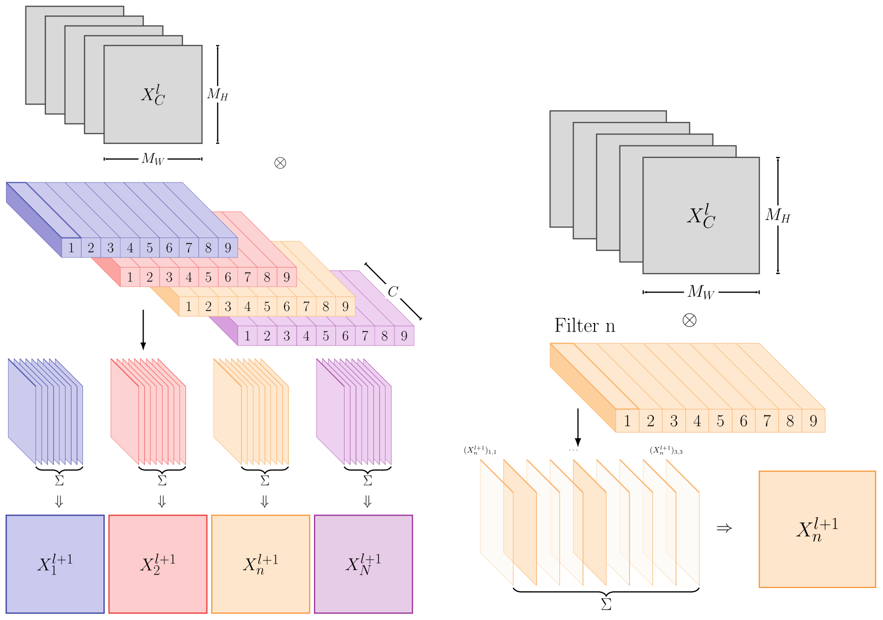
$x^{l}_{c,p,q} = 0$, when $p < 1$ or $p > M_H$ or $q < 1$ or $q > M_W$. M_H is the height of the feature map, while M_W represents the width.

From Fig. 1a, 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. 1b. The following part will prove it in a theoretical manner.

2.2 Theoretical Analysis

Batch normalization (BN) is widely used in a neural network. This method could make DNN faster and more stable [14]. In one filter, suppose B is a mini-batch of size m, i.e., $B = \{a_1, \ldots a_m\}$. BN layer processes these following transformation steps: $\mu_B = \frac{1}{m}\sum_{i=1}^{m} a_i$, $\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(a_i - \mu_B)^2$, $\hat{a}_i = \frac{a_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$, $x_i = \gamma \hat{a}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(a_i)$, 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 c-th channel of l-th layer, the input feature map could be $X^{l}_{c} \sim \mathcal{N}(\beta^{l}_{c}, (\gamma^{l}_{c})^2)$. When M_H is large, $(X^{l}_{c})_{i,j} \sim \mathcal{N}(\beta^{l}_{c}, (\gamma^{l}_{c})^2)$. From (1), we could get $X^{l+1}_{n} = \sum_{i}^{K} \sum_{j}^{K} (\sum_{c}^{C} w^{l}_{n,c,i,j} \times (X^{l}_{c})_{i,j})$.



(a) The filter's kernel size is 3 (b) Single filter case

Fig. 1 Stripe wise convolution

Assuming all data is independently identically distribution, with the properties of normal distribution [15], we have $X_n^{l+1} \sim \mathcal{N}(\mu_n^{l+1}, (\sigma_n^{l+1})^2)$, where $\mu_n^{l+1} = \sum_i^K \sum_j^K (\sum_c^C w_{n,c,i,j}^l \beta_c^l)$ and $(\sigma_n^{l+1})^2 = \sum_i^K \sum_j^K (\sum_c^C (w_{n,c,i,j}^l)^2 (\gamma_c^l)^2)$

To reduce the number of parameters $w_{n,c,i,j}^l$ and avoid the value of μ_n^{l+1} 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.

$$L_n = \text{loss}\left(\mu_n^{l+1}, \sum_i^K \sum_j^K Q_{n,i,j}^l \left(\sum_c^C w_{n,c,i,j}^l \beta_c^l\right)\right) + \alpha g_n(Q) \tag{2}$$

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 $s_{n,i,j}^l \triangleq \sum_c^C w_{n,c,i,j}^l$. If we assume $\beta_1^l = \beta_2^l \cdots = \beta_c^l = \beta^l$, (2) could be written as $L_n = \text{loss}(\beta^l \sum_a^K \sum_b^K s_{n,a,b}^l, \beta^l \sum_i^K \sum_j^K Q_{n,i,j}^l s_{n,i,j}^l) + \alpha g_n(Q)$ and it can be further written as

$$L_n = \text{loss}\left(1, \sum_i^K \sum_j^K Q_{n,i,j}^l T_{n,i,j}^l\right) + \alpha' g_n(Q) \tag{3}$$

where

$$T^l_{n,i,j} = \frac{s^l_{n,i,j}}{\sum_a^K \sum_b^K s^l_{n,a,b}} \tag{4}$$

Obviously, $\sum_i^K \sum_j^K T^l_{n,i,j} = 1, 0 \leq T^l_{n,i,j} < 1$ To minimize (3), we could set $Q^l_{n,i,j} = 0$ to those $T^l_{n,i,j}$ close to 0, which means the corresponding stripes will be pruned. $T^l_{n,i,j}$ could be used to describe the relative importance of stripe$_{i,j}$ in filter$_n$. When $T^l_{n,i,j} \to 1$, stripe$_{i,j}$ contributes more than other stripes. When $T^l_{n,i,j} \to 0$, stripe$_{i,j}$ contributes less than other stripes and could be pruned.

Before setting a threshold for $T^l_{n,i,j}$ 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

$$L = \sum_{(x,y)} \text{loss}(\text{f}(\text{x}, \text{W}), \text{y}) + \alpha \text{g}(\text{W}) \tag{5}$$

where α adjusts the degree of regularization. $g(W)$ is L_1 norm penalty on W and could be written as $g(W) = \sum_{l=1}^L (\sum_{n=1}^N \sum_{c=1}^C \sum_{i=1}^K \sum_{j=1}^K \left| W^l_{n,c,i,j} \right|)$.

To avoid using sub-gradient at non-smooth point, instead of the L_1 penalty, we deploy the smooth-L_1 penalty [16].

3 Experiments

In order to assess the performance of the proposed model and confirm its effectiveness, we carry out experiments using the CIFAR-10 dataset. Our method is implemented using the publicly available Torch. **Dataset and Model:** CIFAR-10 [17] 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 [18] and ResNet56 [19]. The inference time refers to the total amount of time needed to classify 3270 image patches with a size of 224×224. **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.

3.1 Comparing with the Original SWP

To compare our method with the original SWP, we revisit the concept of filter skeleton (FS) from [13]. 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

$$x^{l+1}_{n,h,w} = \sum_{i}^{K}\sum_{j}^{K} I^{l}_{n,i,j}\left(\sum_{c}^{C} w^{l}_{n,c,i,j} \times x^{l}_{c,h+i-\frac{K+1}{2},w+j-\frac{K+1}{2}}\right) \tag{6}$$

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

The regularization on the FS will be

$$L = \sum_{(x,y)} \mathrm{loss}(\mathrm{f}(\mathrm{x}, \mathrm{W} \odot \mathrm{I}), \mathrm{y}) + \alpha \mathrm{g}(\mathrm{I}) \tag{7}$$

where $\odot$ denotes dot product and α adjusts the degree of regularization. $g(I)$ is written as: $g(I) = \sum_{l=1}^{L}(\sum_{n=1}^{N}\sum_{i=1}^{K}\sum_{j=1}^{K}\left|I^{l}_{n,i,j}\right|)$.

For convenience, in Table 1 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 $\alpha = 1e-5$, the number of parameters and the amount of computation of our method are larger than the original 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 $\alpha = 5e-5$, 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 $\alpha = 5e-5$. To compare with the original approach's result of $\alpha = 1e-5$, 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.

In our method, there are two decisive hyper-parameters in the neural network, the coefficient α of regularization in (7) and the weight combination threshold T in (4). As the outcomes of the experiment demonstrated in Table 2, 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.

Table 1 Comparison with the original SWP on CIFAR-10

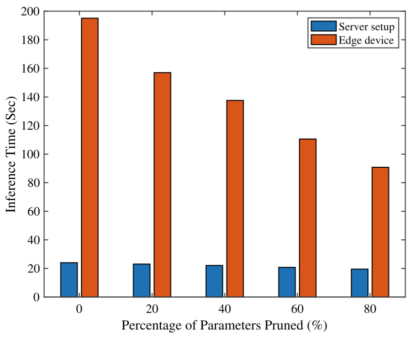
Backbone	Metrics	Params	FLOPS	Accuracy
VGG16	Baseline	14.76 M	627.37 M	93.76 %
	Original($\alpha = 1e-5$)	3.62M	350.28M	93.46 %
	Original($\alpha = 5e-5$)	Could not converge		
	Ours ($\alpha = 1e-5$, $T = 0.0001$)	4.63M	385.49M	93.43 %
	Ours ($\alpha = 5e-5$, $T = 0.005$)	0.84M	126.03M	93.06 %
ResNet56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

Table 2 Different coefficient α and weight combination threshold

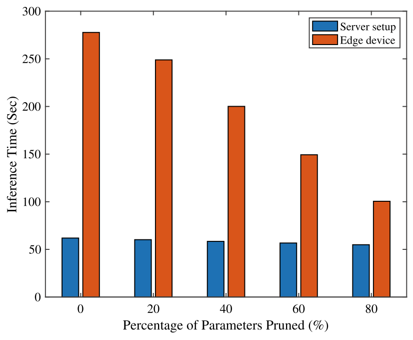
α	1e − 5			5e − 5			
T	0.0001	0.001	0.01	0.0001	0.0005	0.001	0.005
Params (M)	4.63	4.17	2.89	0.78	0.80	0.79	0.84
FLOPS (M)	385.49	327.17	200.09	130.96	135.56	134.68	126.03
Accuracy (%)	93.43	93.28	92.99	92.79	92.86	92.96	93.06

3.2 *Edge Device Performance*

We further verify our approach in an edge device. 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 Fig. 2a, b. 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.



(a) Pruned VGG16



(b) Pruned ResNet56

Fig. 2 Required inference time for pruned models

4 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.

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

References

1. Gubbi J, Buyya R, Marusic S, Palaniswami M (2013) Internet of things (IoT): a vision, architectural elements, and future directions. Fut Gener Comput Syst 29(7):1645–1660
2. Yu W, Liang F, He X, Hatcher WG, Lu C, Lin J, Yang X (2018) A survey on the edge computing for the internet of things. IEEE Access 6:6900–6919
3. Li H, Ota K, Dong M (2018) Learning IoT in edge: deep learning for the internet of things with edge computing. IEEE Netw 32(1):96–101
4. Bishop CM et al (1995) Neural networks for pattern recognition. Oxford University Press, Oxford
5. Han S, Mao H, Dally WJ (2015) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149
6. Molchanov P, Tyree S, Karras T, Aila T, Kautz J (2016) Pruning convolutional neural networks for resource efficient inference. arXiv preprint arXiv:1611.06440
7. Han S, Liu X, Mao H, Pu J, Pedram A, Horowitz MA, Dally WJ (2016) EIE: efficient inference engine on compressed deep neural network. ACM SIGARCH Comput Architect News 44(3):243–254

8. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2016) Pruning filters for efficient convnets. arXiv preprint arXiv:1608.08710
9. Luo JH, Wu J, Lin W (2017) Thinet: a filter level pruning method for deep neural network compression. In: Proceedings of the IEEE international conference on computer vision, pp 5058–5066
10. He Y, Zhang X, Sun J (2017) Channel pruning for accelerating very deep neural networks. In: Proceedings of the IEEE international conference on computer vision, pp 1389–1397
11. Liu Z, Li J, Shen Z, Huang G, Yan S, Zhang C (2017) Learning efficient convolutional networks through network slimming. In: Proceedings of the IEEE international conference on computer vision, pp 2736–2744
12. Lebedev V, Lempitsky V (2016) Fast convnets using group-wise brain damage. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 2554–2564
13. Meng F, Cheng H, Li K, Luo H, Guo X, Lu G, Sun X (2020) Pruning filter in filter. Adv Neural Inform Process Syst 33:17629–17640
14. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: International conference on machine learning, pp 448–456
15. Feller W (2008) An introduction to probability theory and its applications, vol 2. Wiley, Amsterdam
16. Schmidt M, Fung G, Rosales R (2007) Fast optimization methods for l1 regularization: a comparative study and two new approaches. European conference on machine learning. Springer, New York, pp 286–297
17. Krizhevsky A, Hinton G (2009) Learning multiple layers of features from tiny images. University of Toronto, Toronto
18. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556
19. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778