

Variance-Guided Structured Pruning for Optimized Convolutional Neural Networks

Dildar Shah(✉), Muhammad Hanif, and Naveed Razzaq Butt

Ghulam Ishaq Khan Institute of Engineering Sciences and Technology, Swabi, Pakistan
dildarsyed711@gmail.com, {muhammad.hanif,naveed.butt}@giki.edu.pk

Abstract. Deep learning (DL) has changed the landscape of various fields, but its high computing needs make it hard to use in places with limited resources. To address these limitations, this paper presents a novel structured pruning approach that dynamically identifies and removes nonsignificant filters during training using a variance-based threshold. Unlike conventional pruning methods that permanently eliminate filters, our approach applies soft pruning by temporarily deactivating filters with low activation variance, setting their weights to zero, while preserving the original network structure. We retain filters that contribute to some fixed percentage of the total variance in each layer, pruning the rest. Although the variance percentage is fixed, the pruning threshold varies across layers. Importantly, pruned filters remain in the network but are not updated, as their gradients are masked to zero. Experimental evaluations on CIFAR-10, CIFAR-100, and Tiny ImageNet datasets demonstrate that our method reduces computational cost by approximately 58.09% with minimal impact on accuracy.

Keywords: Deep Learning · Resource-constrained AI · Neural network pruning · Structured pruning · Variance · train-time pruning

1 Introduction

Convolutional Neural Networks (CNNs) are commonly applied in areas like image and speech recognition, but they usually need a lot of computing power, such as GPUs or TPUs [12]. In resource-limited environments such as embedded systems, IoT, and mobile devices, model training is often constrained by the lack of computational power [14]. To efficiently utilize these models, it is essential to maintain a proper balance between the network architecture and the number of parameters. The key tunable parameters in neural networks are weights and biases. Adjusting these parameters determines how the network processes input data to achieve the desired outcome. Moreover, the number of parameters significantly impacts performance. Too many parameters increase tuning costs and require more memory, while too few parameters can limit the learning capacity

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2026
O. Gervasi et al. (Eds.): ICCSA 2025 Workshops, LNCS 15899, pp. 230–240, 2026.
https://doi.org/10.1007/978-3-031-97663-6_20

of the network, leading to suboptimal performance [3]. Therefore, optimizing the model structure while keeping computational constraints in mind is crucial for achieving efficiency and accuracy.

Considering these limitations, researchers are focusing on developing high-performing, lightweight networks that ensure better generalization and efficient parameter storage within a single-chip memory [9]. In Machine Learning (ML), finding an optimal network architecture for a specific task remains a challenging and less-understood problem. Additionally, computational overhead in CNNs is often caused by multiple convolutional layers, which significantly increase processing requirements.

Pruning methods can be broadly divided into two categories: structured and unstructured. Individual weights depending on magnitude thresholds are eliminated via unstructured pruning, which frequently results in high compression rates with negligible effect on accuracy [7]. On the other hand, structural pruning reduces computational cost and speeds up inference by eliminating entire sets of weights, such as filters or channels [1].

Researchers are focusing on optimally selecting the features and passing them to the next layer. This helps reduce energy consumption and memory usage. Methods like knowledge distillation, quantization, and pruning are used to make this possible. These methods optimize the model's size and complexity with minimal loss in accuracy. Particularly, network pruning is the most popular as it not only removes unnecessary and redundant connections but maintains the performance as well [1].

1.1 Network Pruning

Network pruning is a technique used to reduce the size and complexity of a trained model by removing nonsignificant parameters, such as connections or neurons. Pruning can be represented mathematically as follows.

$$\tilde{f}(x) = f(x; \mathcal{T}(W)) \quad \text{where} \quad \mathcal{T}(W) = M \odot W, \tag{1}$$

where $\tilde{f}(x)$ is the pruned model, $f(x; W)$ is the vanilla input model with parameters W, M is a binary mask, $\odot$ denotes element-wise multiplication, and $\mathcal{T}$ is a pruning operation. If fine tuning is employed, then Eq. (1) can be updated as:

$$\tilde{f}(x) = f(x; W^{'}), \tag{2}$$

where $W^{'}$ are the weights obtained by fine-tuning $M \odot W$. Likewise, if the pruning is applied iteratively, then for $t \in \{1, ..., T\}$ pruning steps, we have

$$\tilde{f}_t(x) = f(x; \mathcal{T}_t(W)) \quad \text{where} \quad \mathcal{T}_t(W) = M_t \odot W_t^{'} \tag{3}$$

where M_t represents a binary mask by thresholding the fine-tuned weights $W_{t-1}^{'}$ from the previous step, and $W_t^{'}$ denote the updated weights by applying the fine tuning to the masked weights.

Pruning can be performed at different stages of model training: pre-training, post-training, or during training. Pruning during training is the most effective as it allows the network to adapt to the pruning process. Various rules can be applied to decide which connections to remove, like pruning based on weight magnitude or how sensitive the model is to certain weights [4]. Previous studies have shown that many weights can be made close to zero without seriously impacting the model's performance [5]. Various pruning techniques in the literature rely on L_1 and L_2-norm-based loss functions to evaluate the importance of weights [13]. Some researchers focus on sensitivity analysis to identify and retain the most critical parameters, pruning the rest [11]. One of the most widely used criteria is loss-change-based pruning, where parameters are evaluated based on their impact on the overall loss, and the less important ones are removed accordingly [15]. However, most of the existing methods either apply hard pruning, which permanently removes weights from the network architecture, potentially degrading model performance, or use global thresholds that are not layer-specific, which may lead to the loss of important weights across several layers [10].

1.2 Contributions

To overcome these limitations, this work introduces a dynamic soft pruning technique that uses a variance-based threshold to remove less significant parameters during training. Unlike traditional pruning methods, our approach triggers pruning after every few iterations, with a specific threshold set for each layer. We maintain the original network topology while softly removing non-significant parameters, allowing us to achieve true sparsity without the risk of reactivation. Below is a list of our main contributions.

- We employ soft pruning inside the epoch, which differs from conventional methods where pruning typically occurs on an epoch-to-epoch basis.
- We retain only the filters that account for some predefined percentage of the total variance and discard the rest.
- Although the target variance percentage is fixed, each layer has its own threshold. This layer-specific approach ensures that useful information is retained across all layers.
- Depending on the model's complexity, we do not apply pruning after every epoch. Instead, pruning is triggered only after a few epochs, starting from a predefined point in training.

The rest of the paper is structured as follows. In Sect. 2, the idea of variance and its importance in pruning are covered. In the context of pruning, the ResNet-50 architecture is summarized in Sect. 3. Section 4 presents our variance-driven threshold-based pruning method. Results from experiments comparing ResNet-50's performance with and without pruning are reported in Sect. 5. The investigation is finally concluded, and the main findings are summed up in Sect. 6.

2 Variance

It is a statistical measure that illustrates how much the values in a dataset deviate from the mean. It reflects the degree of spread or variability of the data points. Mathematically, it is defined as:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2, \tag{4}$$

where σ^2 is the variance, x_i are the data points, μ is the mean, and N is the total number of samples. In Eq. 2, we use division by N because the variance is computed over all available data, not a sample.

2.1 Significance of Utilizing Variance in Pruning

In CNNs, different filters capture different levels of abstraction. High-variance filters indicate strong responses to diverse input patterns, while low-variance filters remain inactive or redundant. Let each convolutional layer L_i contain a set of filters:

$$\mathbb{F}_i = \{F_1, F_2, \ldots, F_{n_i}\} \tag{5}$$

where n_i represents the total number of filters in the i-th layer. Each filter $F_j \in \mathbb{R}^{C\times K\times K}$ is a tensor with C input channels and spatial dimensions $K \times K$. The importance of a filter is determined based on the variance of its activation outputs across the batch and spatial dimensions. The variance of the i-th filter's activation outputs is computed as:

$$\sigma_i^2 = \frac{1}{BHW}\sum_{b=1}^{B}\sum_{h=1}^{H}\sum_{w=1}^{W}\left(A_i(b,h,w) - \mu_i\right)^2, \tag{6}$$

where $A_i(b,h,w)$ is the activation value of filter i for batch index b and spatial position (h,w), and μ_i is the corresponding mean. Once the variance is computed, all the filters in layer L_i are arranged in ascending order, as given below in Eq. 7.

$$\sigma_{(1)}^2 \le \sigma_{(2)}^2 \le \cdots \le \sigma_{(n_i)}^2. \tag{7}$$

This ordering enables the identification of filters with the lowest variance, which should be pruned. For a given pruning percentage p, we define the number of filters to be retained as follows.

$$k_i = \lfloor n_i \cdot p \rfloor. \tag{8}$$

The floor function $\lfloor\cdot\rfloor$ in Eq. 8 returns the greatest integer less than or equal to the input. The number of filters to be pruned is:

$$r_i = n_i - k_i = \lfloor n_i \cdot (1-p) \rfloor. \tag{9}$$

The pruning threshold for layer L_i is the variance of the r_i-th filter in the sorted list as given in Eq. 10 below.

$$\tau_i = \sigma^2_{(r_i)}. \tag{10}$$

All filters with variances lower than τ_i are pruned by setting their weights to zero. Since each layer has a different number of filters and a different variance distribution, the threshold τ_i varies from layer to layer. i.e., some layers may lose more filters than others, depending on their variance distribution.

3 Residual Network in the Context of Pruning

This section gives a short overview of ResNet-50 concerning pruning. ResNet-50 is a deep CNN consisting of 50 layers, which include convolutional, pooling, and fully connected layers. Its key innovation, residual connections, helps bypass intermediate layers, improving gradient flow and addressing the vanishing gradient problem [6]. Originally, CNNs were used for tasks like handwritten digit recognition, but ResNet-50 significantly enhanced their depth and efficiency [8]. The huge amount of multiply-and-accumulate (MAC) operations in the convolutional and fully connected layers makes the architecture computationally demanding, even though it is successful. Optimization techniques, such as converting convolutions to matrix-matrix multiplications, have been proposed to mitigate this [2]. Our approach reduces this cost by dynamically pruning filters during training, using variance-based thresholding to remove less significant filters. The architecture is shown in Fig. 1, and the next section explains the pruning method in detail.

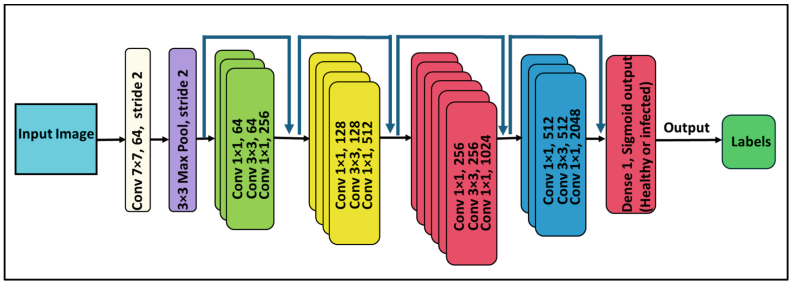

Fig. 1. ResNet-50 architecture with skip connections.

4 Methodology

This section describes our variance-based soft pruning approach for CNNs. The technique prunes the least significant convolutional filters after assessing each filter's significance based on its low activation variance.

4.1 Overview of the Pruning Methodology

The proposed method computes the variance of each filter's activation output and prunes the least significant ones. We employ soft pruning, i.e., pruned filters are set to zero rather than removed from the network architecture. A binary mask M is applied to enforce pruning:

$$M_k = \begin{cases} 1, & \text{if } I_k > \tau, \\ 0, & \text{otherwise.} \end{cases} \tag{11}$$

where I_k represents the importance of the k-th filter computed as the variance of its activation outputs. The pruning mask is applied in-place on the convolutional layer weights to zero out pruned filters after computing activations.

During the forward pass, the masked filters are used for computation:

$$\hat{F}_k = M_k \cdot F_k,$$

where F_k and $\hat{F}_k$ represent weights of original and masked filters respectively. The layer output is then computed as given below in Eq. 12.

$$Y = \mathcal{F}(X * \hat{F}), \tag{12}$$

where $\hat{F}$ represents the masked filter tensor, X is the input tensor, $\mathcal{F}$ is the activation function, and $*$ denotes the convolution operation. During the backward pass, gradients are only computed for unpruned filters as given in Eq. 13. For pruned filters, the gradients are explicitly set to zero.

$$\frac{\partial \mathcal{L}}{\partial \hat{F}_k} = M_k\left(\frac{\partial \mathcal{L}}{\partial F_k}\right). \tag{13}$$

The pruning process is adaptive and dependent on the variance distribution in each layer. It ensures that filters contributing the least to the model's learned representation are removed while preserving high-variance filters that encode more essential features. Algorithm 1 presents the pseudo-code for the proposed method.

Algorithm 1: VarianceâĂŚBased Soft Filter Pruning

Input: Model, optimizer, loss function, pruning parameters
Output: Pruned filters

```
for epoch = 1 to num_epochs do
    for each miniâĂŚbatch (x, y) in training data do
        forward pass; compute ℒ;
        backward pass; compute gradients;
        apply pruning mask M; update weights;
        for each convolutional layer in model do
            compute activation variance per filter;
            sort filters ascending by variance;
            set threshold τ from pruning_fraction;
            update pruning mask M;
        end
    end
    update learning-rate scheduler;
    check early stopping on validation;
end
compute and store pruning stats;
```

$$T_{train} = t_{end} - t_{start}. \tag{14}$$

where t_{start} and t_{end} are the timestamps recorded at the beginning and end of training, respectively. A stochastic gradient descent (SGD) optimizer with a learning rate of 0.003, momentum of 0.9, and weight decay of 0.0001 is used for training. During training, a step-based scheduler modifies the learning rate. Data augmentation methods like horizontal flipping, random cropping, and normalization are used to improve generalization.

5 Results and Discussion

This section presents the experimental results from training a ResNet-50 model on the publicly available datasets[1]: CIFAR-10, CIFAR-100, and Tiny ImageNet. datasets. We use ResNet-50 as a baseline dense model and train it on the two datasets. For the experiments, we utilized a system with an AMD Ryzen 9 7950X 16-Core Processor @ 4.50 GHz, 64.0 GB of RAM (63.1 GB usable), running a 64-bit operating system with an x64-based processor. Initially, we train the baseline model on CIFAR-10, adjusting the final fully connected layer to have 10 output units to match the number of classes in the dataset. The training lasted for 105 epochs, resulting in a final validation accuracy of 97.02%, as shown in Fig. 2 (top).

[1] https://www.cs.toronto.edu/~kriz/cifar.html, https://huggingface.co/datasets/Maysee/tiny-imagenet.

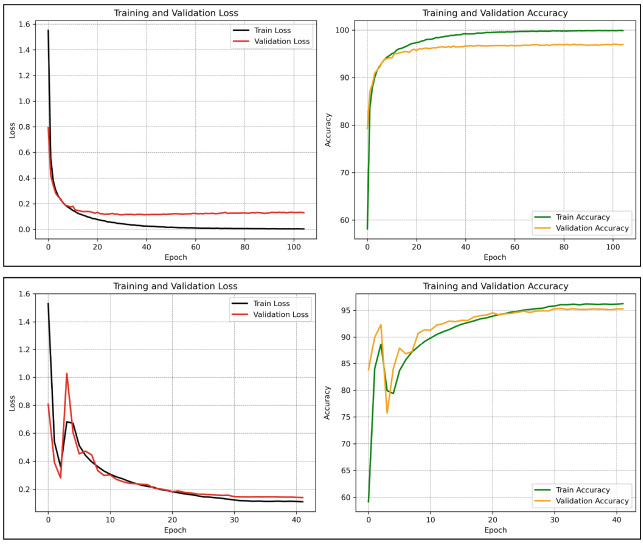

Fig. 2. Training and validation loss and accuracy of the baseline dense model (top) and the pruned model (bottom) on the CIFAR-10 dataset.

The baseline model consists of 23,528,522 trainable parameters, and the training process was completed in 4055.16 s. We significantly reduce computation time by integrating iterative variance-based pruning into the training process while maintaining high accuracy, as shown in Fig. 2 (bottom). The pruned model achieves a final validation accuracy of 96.66% on CIFAR-10, with a reduced training time of 1699.38 s. The initial performance drop in Fig. 2 (bottom) is due to structured pruning, where entire filters along with their associated connections are removed at once. This causes an abrupt drop in accuracy, which stabilizes over subsequent epochs as the model adapts.

After achieving strong results on CIFAR-10, we extend our experiments to the more complex CIFAR-100 dataset, which contains 100 classes. Similar to the previous setup, we modify the final fully connected layer to include 100 units to match the number of classes. The model is trained for 95 epochs, achieving a final validation accuracy of 84.47%, as shown in Fig. 3 (top).

The baseline model consists of $23,712,932$ trainable parameters, with the training process completing in $3,671.78$ s. By incorporating iterative pruning during training, the model achieves a significant reduction in training time with minimal accuracy loss. The pruned model attains a final validation accuracy of 83.37%, while the total training time is reduced to 2175.45 s, as illustrated in Fig. 3 (bottom).

Finally, we evaluate the baseline model on the Tiny ImageNet dataset, which comprises 200 classes and presents higher complexity and greater variability compared to CIFAR-10 and CIFAR-100. To accommodate the increased number of classes, the model architecture was adapted by modifying the final fully

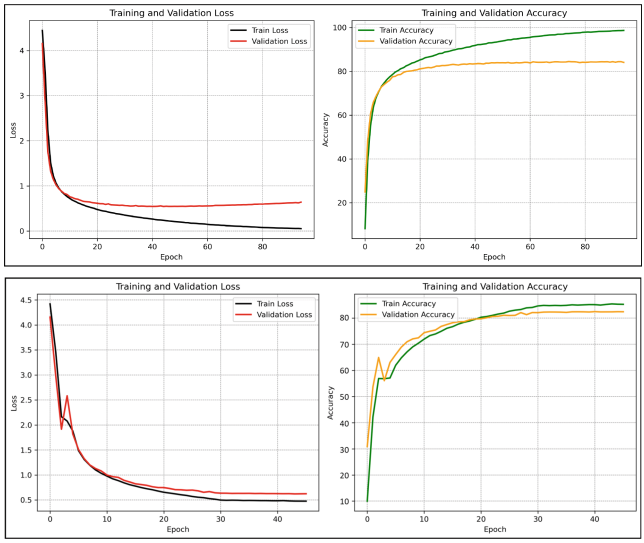

Fig. 3. Training and validation loss and accuracy of the baseline dense model (top) and the pruned model (bottom) on the CIFAR-100 dataset.

connected layer to produce 200 output logits. The model was trained for 60 epochs, achieving a final test accuracy of 82.36%, as illustrated in Fig. 4 (top). These results demonstrate the model's robustness and capacity to generalize effectively to large-scale, fine-grained image classification tasks.

The baseline model comprises 25, 966, 832 trainable parameters, with the complete training process requiring 5, 810.56 s. By integrating pruning into the training pipeline, consistent with the methodology used in previous experiments, the model achieves a final validation accuracy of 81.37% on the Tiny ImageNet dataset with a substantial reduction in training time to 3496.15 s, as shown in Fig. 4 (bottom).

The baseline model and its pruned counterpart are compared on CIFAR-10, CIFAR-100, and Tiny ImageNet datasets to evaluate the effect of pruning. The outcomes demonstrate the benefits of pruning, particularly its ability to reduce training times without compromising accuracy too much. Notably, there is very little accuracy loss between datasets, indicating that the suggested trimming technique works. Since this kind of work is rarely explored in the literature, we can not find a direct comparison with any method in the literature. Table 1 provides a thorough comparison of our baseline and pruned models' performance.

On CIFAR-10, pruning reduced the training duration from 4055.16 s to 1699.38 s, reflecting an approximate 58.09% reduction. Despite this significant computational improvement, the test accuracy declined only slightly from 97.02% to 96.66%, resulting in a minimal accuracy drop of 0.36.

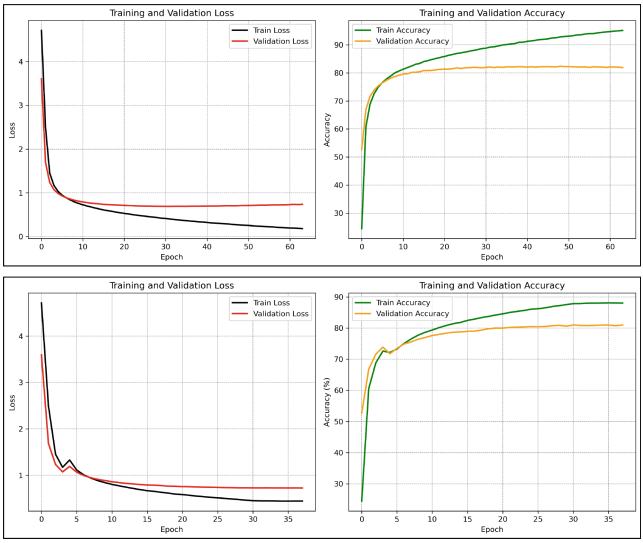

Fig. 4. Training and validation loss and accuracy of the baseline dense model (top) and the pruned model (bottom) on the Tiny ImageNet dataset.

Table 1. Results comparison of baseline model and its pruned counterpart

Dataset	Model	Training Time (s)	Time Reduction (%)	Test Accuracy (%)	Accuracy Drop (Baseline – Pruned)
CIFAR-10	Baseline	4055.16	58.09	97.02	0.36
	Pruned	1699.38		96.66	
CIFAR-100	Baseline	3671.78	40.75	84.47	1.1
	Pruned	2175.45		83.37	
Tiny ImageNet	Baseline	5810.56	39.83	82.36	0.99
	Pruned	3496.15		81.37	

On CIFAR-100, the pruning approach lowered the training time from 3671.78 s to 2175.45 s, marking a 40.75% reduction. The test accuracy decreased marginally from 84.47% to 83.37%, yielding a modest drop of 1.1.

On Tiny ImageNet, the training time reduced to 3496.15 s, showing a total training time reduction of 39.83% with a minimal loss of 0.99 in the final test accuracy.

These results demonstrate that the pruning technique effectively minimizes computational demands while preserving model performance.

6 Conclusion

In this article, we proposed a dynamic soft-pruning method based on variance analysis to address the computational challenges of deep learning models. Unlike

conventional pruning techniques that permanently remove weights, our method identifies and prunes less important weights during training. This layer-wise pruning approach, which does not depend on a global criterion, ensures precise pruning tailored to each layer's specific characteristics. Our experiments on the CIFAR-10, CIFAR-100, and Tiny ImageNet datasets show that this method significantly reduces training time with minimal impact on accuracy. The results demonstrate the effectiveness of our approach in producing efficient neural networks suitable for environments with limited computational resources.

References

1. Anwar, S., Hwang, K., Sung, W.: Structured pruning of deep convolutional neural networks. ACM J. Emerg. Technol. Comput. Syst. **13**(3), 1–18 (2017)
2. Chellapilla, K., Puri, S., Simard, P.: High-performance convolutional neural networks for document processing. In: Tenth International Workshop on Frontiers in Handwriting Recognition. Suvisoft (2006)
3. Dauphin, Y.N., Bengio, Y.: Big neural networks waste capacity. arXiv preprint arXiv:1301.3583 (2013)
4. Han, S., Mao, H., Dally, W.J.: A deep neural network compression pipeline: pruning, quantization, Huffman encoding. arXiv preprint arXiv:1510.00149 **10** (2015)
5. Han, S., Pool, J., Tran, J., Dally, W.: Learning both weights and connections for efficient neural networks. In: Advances in Neural Information Processing Systems, vol. 28 (2015)
6. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 770–778 (2016)
7. Lazarevich, I., Kozlov, A., Malinin, N.: Post-training deep neural network pruning via layer-wise calibration. In: Proceedings of the IEEE/CVF International Conference on Computer Vision, pp. 798–805 (2021)
8. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
9. Li, Z., Li, H., Meng, L.: Model compression for deep neural networks: a survey. Computers **12**(3), 60 (2023)
10. Liu, J., Zhang, X., Guo, Y., Lin, D., Hu, H.: Dynamic sparse training: Find efficient sparse network from scratch. arXiv preprint arXiv:2005.06870 (2020)
11. Santacroce, M., Wen, Z., Shen, Y., Li, Y.: What matters in the structured pruning of generative language models? arXiv preprint arXiv:2302.03773 (2023)
12. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
13. Sun, M., Liu, Z., Bair, A., Kolter, J.Z.: A simple and effective pruning approach for large language models. arXiv preprint arXiv:2306.11695 (2023)
14. Thompson, N.C., Greenewald, K., Lee, K., Manso, G.F.: The computational limits of deep learning. arXiv preprint arXiv:2007.05558 (2020)
15. You, Z., Yan, K., Ye, J., Ma, M., Wang, P.: Gate decorator: global filter pruning method for accelerating deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 32 (2019)