Hindawi
Scientific Programming
Volume 2021, Article ID 9971669, 11 pages
https://doi.org/10.1155/2021/9971669

Research Article

SuperPruner: Automatic Neural Network Pruning via Super Network

Yu Liu,[1] **Yong Wang**,[2] **Haojin Qi**,[2] **and Xiaoming Ju**[1]

[1]*East China Normal University, Software Engineering Institute, Shanghai, China*
[2]*State Grid Ningbo Electric Power Company, Information and Communication Branch, NingBo, China*

Correspondence should be addressed to Xiaoming Ju; xmju@sei.ecnu.edu.cn

Received 22 March 2021; Revised 11 July 2021; Accepted 16 August 2021; Published 14 September 2021

Academic Editor: Pengwei Wang

Most network pruning methods rely on rule-of-thumb for human experts to prune the unimportant channels. This is time-consuming and can lead to suboptimal pruning. In this paper, we propose an effective SuperPruner algorithm, which aims to find optimal pruned structure instead of pruning unimportant channels. We first train a VerifyNet, a kind of super network, which is able to roughly evaluate the performance of any given network structure. The particle swarm optimization algorithm is then used to search for optimal network structure. Lastly, the weights in the VerifyNet are used as the initial weights of the optimal pruned structure to make fine-tuning. VerifyNet is a network performance evaluation; our algorithm can quickly prune the network under any hardware constraints. Our algorithm can be applied in multiple fields such as object recognition and semantic segmentation. Extensive experiment results demonstrate the effectiveness of SuperPruner. For example, on CIFAR-10, the pruned VGG16 achieves 93.18% Top-1 accuracy and reduces 74.19% of FLOPs and 89.25% of parameters. Compared with state-of-the-art methods, our algorithm can achieve higher pruned ratio with less accuracy cost.

1. Introduction

In recent years, deep neural networks have achieved remarkable results in various fields (including object recognition [1–4], object detection [5–7], semantic segmentation [8, 9], and autonomous driving [10]). However, the trained neural network needs to save a large number of parameters, and at the same time, one forward propagation requires a large number of matrix calculations, which prevents the application of the neural network on edge devices with limited resources. In order to alleviate this problem, the researchers proposed several CNN compression techniques, including low-rank decomposition [11, 12], parameter quantification [13–15], network pruning [16–21], and knowledge distillation [22, 23]. Among them, network pruning is widely concerned as a simple and efficient method. The traditional network pruning method [16, 17, 24] consists of three steps: (1) pretraining, (2) filter pruning, and (3) fine-tuning. In the filter pruning process, human experts design rules to evaluate the importance of filter and delete unimportant filters. At the same time, the pruning rate of each layer (pruning rate affects the network structure) requires a lot of experiments to determine. In traditional network pruning, the pruning results are highly dependent on human experts, which often lead to suboptimal pruning.

In order to reduce the impact of human experts on pruning results, the automatic pruning method [18, 19, 25, 26] came into being. The automatic pruning method uses the ideas of reinforcement learning [18] or intelligent search algorithm [19] and automatically prunes the network model through continuous iteration. These methods free human experts from rule design and choose of pruning rate, not only saving a lot of time but also improving the performance of the pruned network model. In addition, Liu et al. [27] and Wang et al. [28] believed that the essence of network pruning is pruning the network structure, rather than pruning unimportant filters. Therefore, we propose the SuperPruner algorithm, which automatically prunes the model by finding the optimal network structure.

Assuming that a deep neural network has l layers and each layer has fixed n channels, the total search space is l^n. However, the search space will increase exponentially with increased channels. It is obviously unacceptable to search all network structures in the search space. We limit the number of channels that can be reserved for each layer to αn, $\alpha \in \{10\%, 20\%, \ldots, 100\%\}$, which means the convolutional layer of the pruned network has only $|\alpha|$ possible values. We reduce the search space from the original l^n to $l^{|\alpha|}$ from which we propose SuperPruner based on the above search space reduction. The algorithm is inspired by NAS [29–32], especially the one-shot model [32, 33]. As shown in Figure 1, we first train a VerifyNet which can quickly predict the performance of any network structure in the search space and then find the optimal network structure through the search algorithm. And finally, we fine-tune the optimal network structure to obtain the pruned network model. When our algorithm predicts the performance of the network structure, it only needs one inference to obtain the accuracy on the validation set, without any fine-tuning, and the whole algorithm is simple and efficient. The SuperPruner algorithm we propose alleviates the slow and expensive problem of performance evaluation in the optimal network structure search process. Compared with the SOTA method, our algorithm can achieve higher pruned ratio with less accuracy cost.

Our contribution mainly includes the following three aspects:

(1) We propose an automatic pruning algorithm, SuperPruner. The core of this algorithm is to train a VerifyNet, which can directly predict the performance of all pruning structures. Because network search and performance predictions are decoupled by VerifyNet, we can prune the network structure under arbitrary resource constraints.

(2) Our algorithm can prune common network structures such as VGG [2], GoogLeNet [3], ResNet [4], and UNet [9]. We applied the model compression algorithm to the semantic segmentation task for the first time and achieved competitive results.

(3) Compared with traditional network pruning algorithms, our algorithm obtains an improved pruning with little participation of human experts. Compared with the automatic pruning method, SuperPruner can directly get the performance of the network structure without any fine-tuning.

2. Related Work

Since the method proposed in this paper belongs to network pruning, we have summarized the recent work of network pruning in the following.

2.1. Traditional Network Pruning. Traditional network pruning is divided into two categories, unstructured pruning and structured pruning. Unstructured pruning [16, 34, 35] is fine-grained, and its purpose is to cut off the unimportant weight connections in the pretrained neural network. This will result in sparse CNNs with irregularities, which usually require special software and hardware accelerators to speed up the inference speed.

In contrast, structured pruning [17, 24, 36, 37] is coarse-grained and can completely remove unimportant filters. It is easy to achieve the purpose of computing acceleration. Li et al. [38] used the $l1$-norm, and Liu et al. [39] used the learnable scaling factor of the BN to remove unimportant filters. Luo et al. [36] proposed reconstruction errors to prune filters. Lin et al. [20] proposed a new global and dynamic pruning scheme, which can prune redundant filters to achieve CNN acceleration. However, the abovementioned methods require a lot of experiments to determine hyperparameter (the pruning rate of each layer). Furthermore, the pruning results are affected by human subjectivity, which likely cause suboptimal pruning. Different from the traditional network pruning method, we propose an automatic pruning algorithm. The whole pruning process hardly requires the participation of human experts, and the results obtained are better.

2.2. AutoML. Recent years have seen that the emergence of AutoML frees human experts from tedious rule-of-thumb and hyperparameter design. He et al. proposed the AMC [18] method that automatically generates the pruning rate of each layer through the DDPG [40] in reinforcement learning. Lin et al. [41] trained a GAN and let the generator directly generate the pruned network model. Dong et al. [21] proposed to train an unpruned network and search for the most suitable depth and width of a network minimizing the computation cost. The parameters of the searched/pruned networks are then learned by knowledge transfer from unpruned network. Liu et al. [25] proposed to train a PruningNet which can predict the weight of the network structure after pruning and then search the optimal network structure through PruningNet. However, Meta-pruning evaluates the performance of the searched network structure, and it needs to perform another calculation based on the weights generated by PruningNet to predict the performance of the network structure. Luo et al. [26] used search methods instead of reinforcement learning to compress the network model which the ADMM [35] as the core optimization algorithm. Lin et al. proposed the ABCpruner [19] that used the ABC algorithm to search the network structure. However, ABCpruner introduced the process of retraining when evaluating the performance of a searched network structure, which takes a lot of time and computing resources. Even if the weights in the pretraining model are used as the initial weights of the searched model and a few steps of fine-tuning are performed on this basis, the cost of ABCpruner on performance evaluation is unacceptable.

Different from the above method, our algorithm needs to train a super network which is requires only one forward propagation to predict the performance of the searched network. It does not require any fine-tuning during network performance evaluation, saving resource consumption.

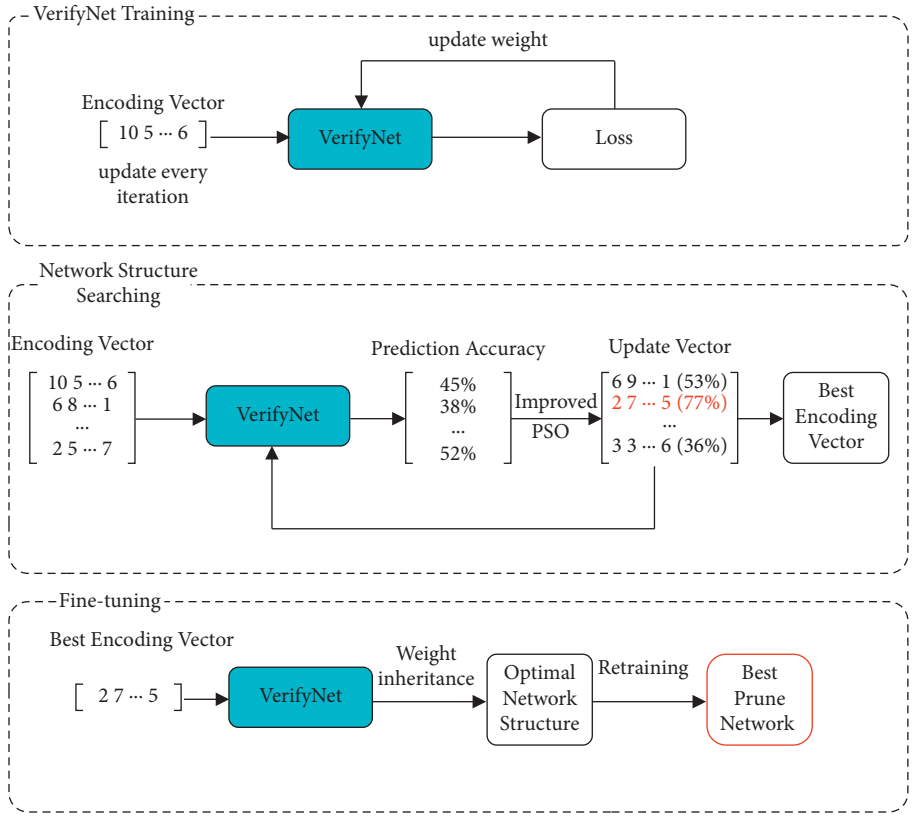

FIGURE 1: Our SuperPruner has three steps. (1) Train a VerifyNet, the input is the coding vector of the network structure, and the output is the accuracy of the prediction on the given data set. Update the encoding vector once per iteration. (2) Search for the best network structure. We use the improved PSO algorithm to find the optimal network structure on VerifyNet. In the search process, only one inference is needed to predict the accuracy of the network structure on given data set without any retraining. (3) *Fine-Tuning*. The searched optimal network structure inherits the weight on VerifyNet and fine-tunes it to obtain the best pruned network.

2.3. NAS. Our algorithm is inspired by one-shot architecture search in NAS. The core idea of one-shot architecture search is to reuse the trained network as much as possible by weight sharing weight generation [25, 31] or [30, 32] so that when evaluating the performance of the searched network structure, there is no need to retrain from scratch and reduce a lot of calculations. For example, Brock et al. [31] and Liu et al. [25] trained hypernetworks to generate the weights of the searched network structure. Pham et al. [30] proposed directed acrylic graph (DAG) representing the search space, all the subnetworks in the DAG mandatory sharing parameters. Guo et al. [32] and Li et al. [33] proposed to train a super network that includes all substructures. The common edges of different substructures share the weights in the super network. Only trained once, all substructures can get their weights directly from the super network.

In NAS, the input and output of each layer are fixed. However, during the channel pruning process, the input of the current layer will change with the output of the previous convolutional layer. It is not feasible to directly apply the one-shot architecture search in the NAS to the channel pruning task. Therefore, we design a pruning module to replace the convolutional layer in the one-shot model, which perfectly solves the problem of unfixed input of the convolutional layer in the network pruning task.

3. Materials and Methods

In this chapter, we will introduce the SuperPruner, which can efficiently prune convolutional neural networks. A represents the entire search space, and the pruned network structure a is subset of A. We define $M(a, \omega)$ as a network model with structure a, and the accuracy of a on the test set is used to measure the network performance. As equation (1) shows, the purpose of network pruning is to find a compressed network structure with the optimal accuracy on the test set.

$$a^* = \arg\max_{a \in A} ACC_{\text{test}}(M(a, \omega_a)). \tag{1}$$

However, in real application scenarios, typically the parameter of the model, FLOPs, inference speed, and energy consumption have certain requirements. A common practice is to limit the parameters, such as the following equation:

$$\text{Para}(a)^* \leq \text{Para}_{\max}. \tag{2}$$

Therefore, we need to optimize equation (1) under the conditions of equation (2) to obtain the optimal network structure, such as the pruned network with the highest accuracy.

However, solving the real accuracy requires retraining the searched network from scratch, which will cost a lot of computing resources. As shown in Figure 1, to solve this problem, we propose to train an auxiliary network (VeriftyNet), which can quickly predict the accuracy of all subnetworks on the test set without retraining. Then, the PSO algorithm is used to search the optimal network structure in VeriftyNet. Because structure search and performance evaluation are separated, our algorithm can obtain the optimal network structure under arbitrary hardware constraints.

3.1. VerifyNet Structure. The input of VerifyNet is the encoding vector of the network structure, and the output is the prediction of the accuracy on the given data set. When the test set is given, we can use the following equation to calculate accuracy to predict the true performance of the network structure.

$$\operatorname{acc}(a) = \underset{a \in A}{\textbf{Verify Net}}\left(a; T_{\text{test}}\right). \tag{3}$$

We limit the number of channels that can be reserved for each layer to αn, $\alpha \in \{10\%, 20\%, \ldots, 100\%\}$. This means that no matter which layer of the neural network is concerned, there are only 10 cases where the number of channels is reserved. When constructing VerifyNet, we allocate a channel block for each possible situation and use 10 channel blocks corresponding to 10 feasible solutions of this layer. The same channel block is shared between different paths. In this way, through the sharing of channel blocks, only $10L$ channel blocks are needed for an L-layer convolutional neural network to represent all possible situations in the search space of L_{10}. Figure 2 shows the structure of VerifyNet with three convolutional layers. Given the encoding vector, we can predict the Top-1 accuracy of the network structure corresponding to the encoding vector.

In the network pruning task, the input of the current convolutional layer is determined by the output of the previous convolutional layer. Using the same channel block for different types of network models can be difficult to handle. As shown in Figure 3, in order to make our algorithm effective for various common network models, we design three different blocks. Block (a) is composed of Conv, BN, and ReLu, suitable for LeNet, VGG, MobileNet, and other types of networks without shortcut. For block (a), the output and maximum input of the channel block are fixed, and the real input can change according to the output of the previous convolutional layer. We can easily implement this function by slicing the convolution kernel. Block (b) is suitable for shortcut networks such as ResNet and UNet. We fix the input and output of block (b) unchanged so that the convolution kernel at the short connection will not be changed and only the middle layer of the block (b) will be pruned. The block (c) is suitable for GoogLenet and other similar network structures. For block (c), only 1×1 convolutional layer on both sides of the branch is not pruned.

We use the appropriate block to construct a VerifyNet according to the type of network to be pruned. VerifyNet is 5.5x the size of the original network. After training, any network structure can be evaluated with only one inference. Compared with the computational power consumption of retraining the subnetwork, it is acceptable to have 5.5x more memory consumption in the training.

3.2. VerifyNet Training. The purpose of VerifyNet is to train VerifyNet only once to predict the performance of all subnetworks through channel sharing. We hope that the subnetwork weight inherited from VerifyNet and the subnetwork weight trained from scratch are as close as possible. This requires equal training of all paths in VerifyNet. To solve this problem, we propose a random sampling path strategy to train VerifyNet.

In forward propagation, the encoding vector (representing the structure of the neural network) is randomly generated as the input of VerifyNet. The path corresponding to the coding vector is activated, while the remaining paths are in an inactive state. The coding vector is updated every time a batch size is trained. In backpropagation, unlike traditional training, we do not update all weights. Only the activated path will perform gradient calculation and update the weight of the channel block on the path.

In our VerifyNet, overlapping parts in different paths share the same block. For the path that is not sampled, the blocks on this path will be trained in other paths. When all blocks have been trained, it also means that this path without sampling has also been trained. All paths in the solution space will be trained equally due to the special shared structure of VerifyNet. Because only one path is selected each time, our VerifyNet is not significantly different from the normal network during training, and it can quickly reach convergence.

We cannot guarantee that the order of network performance predicted by VerifyNet is the same as the real order, but we can guarantee that it will not differ too much. Because each path in VerifyNet is trained by random sampling, it is as close as possible to the weight of the real training. When we choose another path for training, it will affect the originally trained channel block in this path, but every channel block in VerifyNet will be affected in this way, so the entire VerifyNet will maintain a dynamic balance. The network performance predicted by VerifyNet is often lower than the real result, but it is almost the same as the real performance ranking.

3.3. Network Structure Search. After completing the training, the VerifyNet at this time is no longer a network in the traditional sense but a network estimator. If the encoding vector and the validation set are input, only one inference gets accuracy to the subnetwork corresponding to the encoding vector, without any fine-tuning. Because of the large search space, random search is not advisable. In order to find the optimal network structure, SuperPruner uses PSO to search the network structure on VerifyNet.

We first randomly initialize m one-dimensional particles $\{C_i\}_1^m$, with position of the particle representing the network structure (the encoding vector input by VerifyNet) and

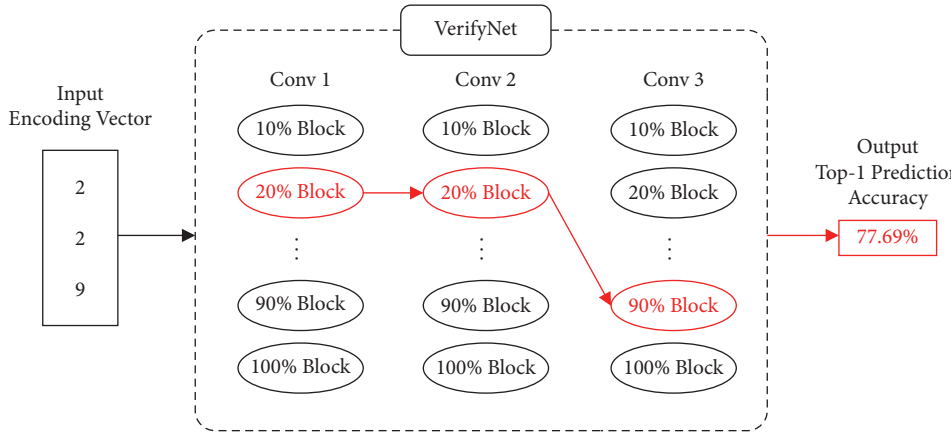

FIGURE 2: The structure of VerifyNet with three convolutional layers. Each layer of VerifyNet consists of 10 blocks. The number of channels in α% block is α% of the original number of channels in the convolution kernel. Only the block corresponding to the encoding vector is activated. Given a data set and input the encoding vector, we can predict the Top-1 accuracy of the network structure corresponding to the encoding vector on the data set.

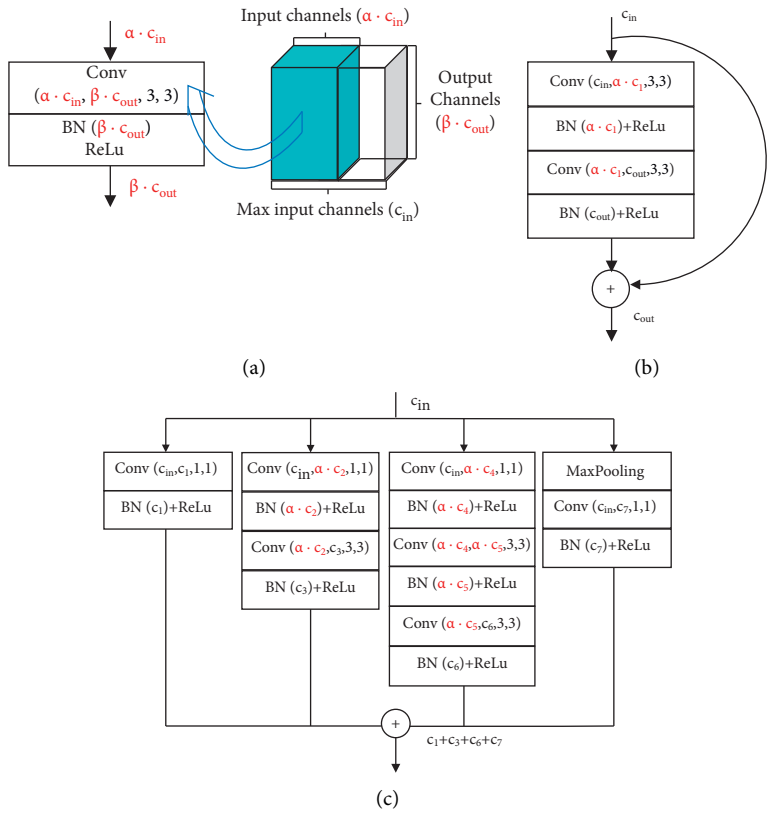

FIGURE 3: Three blocks of the VerifyNet. The red font represents that part which needs to be pruned. We use block (a) to prune VGG16, block (b) to prune ResNet, and block (c) to prune Googlenet. For ResNet and Googlenet, we only prune the middle layer and do not change the channel number of input and output of the block.

velocity V_i representing the update direction of the particle. Then, we calculate the fitness function value of each particle according to equation (4) and update the local optimal particle P_i (the optimal particle found by the i-th particle during the algorithm operation) and the global optimal particle G (the optimal particle found during the algorithm operation).

$$\text{fit}(a) = \mu \cdot \text{acc}(a) + (1 - \mu) \cdot \text{Para}(a), \tag{4}$$

where $\mu \in (0\%, 100\%)$ is a preset constant. It represents the proportion of the network performance when calculating the fitness function. Finally, the velocity and position of each particle are updated according to equation (5), and the algorithm is executed iteratively.

$$\begin{cases} V_i = w \cdot V_i + a_1 \cdot r \cdot (h_i - C_i) + a_2 \cdot r \cdot (G - C_i), \\ C_i = C_i + V, \end{cases} \tag{5}$$

where $r \in (0, 1)$ is a random number.

The standard PSO algorithm fixes the size of w, and the particles cannot obtain a balance between the global search and the local search. This will reduce the diversity of the model, and the particles cannot search for new regions and eventually fall into a local optimal solution. According to equation (6), in order to solve this problem, we dynamically change the size of w when updating the model speed to help the particles expand the search space and jump out of the local optimal solution.

$$w = 0.1 \cdot \cos\left(\frac{\pi}{N} \cdot n\right) + 0.2, \tag{6}$$

where N is the maximum number of iterations of the PSO and n is the current number of iterations of the PSO.

In addition, we have introduced the concept of detection particles. When a particle has not been updated for a long time, we think that the particle has fallen into a local optimum. In this case, a detective particle will be generated to replace the particle that has fallen into the local optimum. By introducing detection particles, the search space can be well expanded, which helps the particles to jump out of the local optimal solution and avoid the premature phenomenon.

After the algorithm is executed, the optimal network structure we searched for is the neural network represented by the global optimal particles. SuperPruner evaluates the searched network structure performance through VerifgNet which has been trained before searching. Therefore, after performance evaluation and structure search are completely decoupled, we can easily search for the optimal network structure under arbitrary hardware constraints by modifying the fitness function in PSO.

The weight of the searched optimal network structure is inherited from VerifyNet. We only need to fine-tune a few steps on the training set to get the pruned network model. More details of the improved PSO algorithm are shown in Algorithm 1.

4. Results and Discussion

We conducted experiments on object recognition and image segmentation tasks to verify the effectiveness of the SuperPruner. The pruned network model includes VGG, GoogLeNet, ResNet, and UNet. All experiments run on one NVIDIA Tesla P40 GPU, implemented with Pytorch.

4.1. Experimental Settings

Datasets. On object recognition task, we evaluated our method on CIFAR-10 and CIFAR-100. The CIFAR-10 has 10 classes, and each class has 6K images. There are 50K training images and 10K test images. CIFAR-100 is similar to CIFAR-10 but is divided into 100 classes, each with 600 images. We randomly divide the original training set into two parts: 10% of images are used as the validation set and the remaining as the training set. The divided validation set is used for predicting performance, which network structure searches for VerifyNet, ensuring the generalization of the network.

Training Strategy. VerifyNet plays a very important role in quickly predicting network structure performance. For CIFAR dataset, we use the stochastic gradient descent (SGD) algorithm with a momentum of 0.9 and a weight decay of 0.0001. We train each VerifyNet by 2K epochs with the initial learning rate of 0.1, which is scaled by 0.25 over 500 epochs. The batch size is set to 256. When training the optimal network structure, we reduce the epochs from 2K to 150 and the learning rate is divided by 10 every 50 epochs.

PSO Parameter. In order to find the optimal network structure, we experimentally set $M = 20$, $N = 100$, and $T = 10$ in Algorithm 1. The value of m changes according to the network structure such as m is set to 16 for VGG16 and set to 27 for ResNet56. The value of μ belongs to $\{10\%, 20\%, \ldots, 100\%\}$, and we can freely choose the value of the μ according to actual application. The influence of μ on the optimal network structure will be discussed in chapter *D*.

4.2. Results on Object Recognition Task

4.2.1. VGG16. VGG16 has 13-conv and 3-fc without shortcut, and the baseline can achieve 93.45% accuracy on CIFAR-10. Using SuperPruner to prune VGG16, we can remove 74.19% FLOPs and 89.25% parameters, but the accuracy can still be kept at 93.18%. As seen from Table 1, compared with other methods, such as GAL [41] and ABCpruner [19], our method is superior in FLOPs and parameters pruning ratio, with almost no reduction in accuracy. For example, our method can reach the higher pruning rate of FLOPs (74.19% vs. 45.26% by GAL and 73.68% by ABCpruner) and parameter (89.25% vs. 82.22% by GAL and 88.68% by ABCpruner) by the less accuracy loss (−0.3% vs. −0.54% by GAL and −0.37% by ABCpruner). Based on further analysis, Figure 4 shows that SuperPruner retains more channels and parameters for the first few layers of VGG16, and the parameter pruning rate is significantly improved starting from Conv6. This is because each layer of the network has different sensitivities to pruning, resulting in different pruning rates. The first few layers of VGG16 are mainly used for feature extraction, and retaining more channels and parameters helps the compressed model maintain high accuracy. Therefore, SuperPruner can automatically learn network structure information through particle swarms in the search to obtain the optimal pruner model.

4.2.2. GoogLeNet. For GoogLenet, our experimental results can be obtained from Table 2. We can remove 55.27% parameters, and the accuracy is only 1.37% lower than baseline. At the same time, SuperPruner can achieve 55.29% FLOPs pruning rate. The comparison results of SuperPruner and other algorithms are shown in Table 2. With the same FLOPs

Hyperparameter: number of particles: M, dimension of Particles: m, the maximum number of iterations: N, impact factor: μ, max time: T, input: VerifyNet: **Verify Net**, validation set: val, and output: the optimal pruned network structure: a^*

(1) Initialize the particle position $\{C_i\}_1^m$, particle velocity $\{V_i\}_1^m$, historical best position $\{P_i\}_1^m$, global best position G, and no updated time t_{i1}^m
(2) for $i = 0$: N do
(3) for $j = 0 : M$ do
(4) Calculate $\text{fit}(C_j)$ based on equation (4)
(5) Update V_j and C_j based on equation (5)
(6) if $\text{fit}(C_j) > \text{fit}(P_j)$, then
(7) $P_j = C_j$, $t_j = 0$
(8) else $t_j = t_j + 1$
(9) end if
(10) if $\text{fit}(C_j) > \text{fit}(G)$, then
(11) $G = C_j$
(12) end if
(13) if $t_j > T$, then
(14) randomly initialize m one-dimensional particles C_j
(15) $t_j = 0$
(16) end if
(17) end for
(18) Calculate w based on equation (6)
(19) end for
(20) return G

Algorithm 1: The improved PSO algorithm.

Table 1: Accuracy and pruning ratio of VGG16 on CIFAR-10.

Method	Prune Acc (%)	Acc drop (%)	FLOPs/PR (%)	Parameters/PR (%)
L1[16]	93.40	0.56	206.00 M/34.30	5.40 M/64.00
GAL-0.1 [41]	93.42	0.54	171.89 M/45.20	2.67 M/82.20
ABCpruner-80% [19]	93.08	−0.06	82.81 M/73.68	1.67 M/88.68
SuperPruner-50%	93.18	0.27	81.19 M/74.19	1.64 M/89.25

Acc: accuracy; PR: pruning rate.

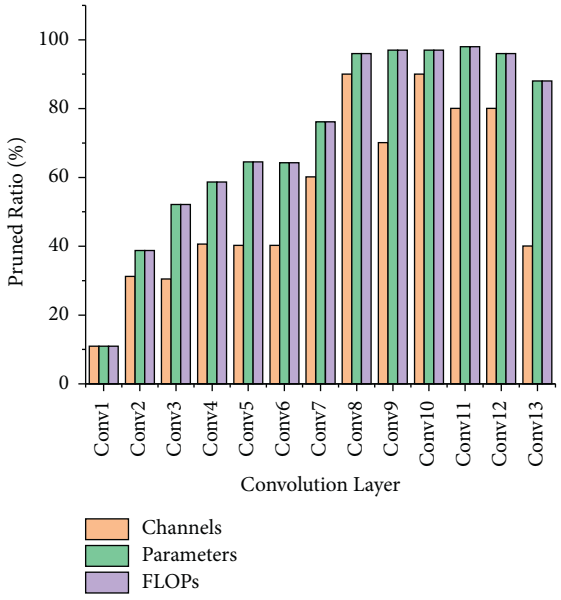

Figure 4: The pruned ratio of each layer for VGG16 on CIFAR-10.

Table 2: Accuracy and pruning ratio of GoogLeNet on CIFAR-10.

Method	Prune Acc (%)	Acc drop (%)	FLOPs/PR (%)	Parameters/PR (%)
L1 [16]	94.54	0.51	1.04 GM/31.93	3.51 M/43.11
GAL-0.5 [41]	94.56	0.49	0.94 GM/38.20	3.12 M/49.30
ABCpruner-30% [19]	94.84	0.21	0.51 GM/66.56	2.46 M/60.14
SuperPruner-50%	93.78	1.27	0.68 GM/55.29	2.76 M/55.27

pruning ratio and parameters pruning ratio, SuperPruner is better than L1 [16] in pruning rate (55.29% vs. 31.39% for FLOPs and 55.27% vs. 43.11% for parameters) and below L1 in accuracy (93.78% vs. 94.54%). Comparing with ABCpruner [19] and GAL [41], our algorithm has a slightly lower accuracy than ABCpruner but the pruning rate exceeds that of GAL.

4.2.3. ResNet. VGG16 is a simple network that focuses on building convolutional layers, and there is no Short-Block. In order to verify the effect of SuperPruner on the Short-Block model, we prune ResNet56 on CIFAR-10 and CIFAR-100. We construct VerifyNet on ResNets as shown in Figure 3(b), which does not change the input and output of the block and only trims the middle part. We summarize the pruning results of ResNet56 in Table 3. On CIFAR-10, we set μ to 80%, our algorithm can reach 80.74% FLOPs pruning rate, but the accuracy declines by 1.10%. On CIFAR-100, SuperPruner can achieve 57.30% FLOPs pruning rate, and the accuracy is 2.23% lower than the unpruned network. This is because ResNet56 has lots of redundant connections during the design, and SuperPruner can automatically find these redundant connections and prune them. Removing these connections can effectively prevent over-fitting and will not affect network performance. Compared with other methods, our model can achieve competitive results. Compared with GAL, SuperPruner can achieve better results. The accuracy is increased from 91.58% to 92.17%, and the FLOPs pruning rate is increased from 60.20% to 80.74%. Even compared with the state-of-the-art algorithm TAS [21], SuperPruner still reaches the higher pruning rate of FLOPs (80.74% vs. 52.70%), with a slightly accuracy loss (92.17% vs. 92.81%). For CIFAR-100, the performance of most algorithms has declined, but our algorithm can still achieve the highest FLOPs pruning rate (57.30%) with a small loss of accuracy (68.97%).

4.3. Results on Semantic Segmentation Task. This paper presents results of pruning of UNet trained for semantic segmentation on the Carvana data based on Kaggle's Carvana Image Masking Challenge from high definition images. After the challenge, we can only get all the images and corresponding masks of the training set. In order to obtain the pruned model with the best effect and generalization ability, we redivide the original training set into training set, validation set, and test set according to the ratio of 6 : 2 : 2. Consistent with the competition, we evaluated the pruned network on the mean dice coefficient. The dice is defined in our experiment as follows:

$$\text{dice} = \frac{2^*|X \cap Y|}{|X| + |Y|}, \tag{7}$$

where X is the predicted segmentation set of pixels and Y is the ground truth. The dice coefficient is defined to be 1 when both X and Y are exactly the same.

We use the block (b) to construct VerifyNet according to the structure of UNet. The VerifyNet was trained from scratch with 4096 images (no data augmentation) and trained 20 rounds in total. We set the initial learning rate to 0.0001, batch size to 1, and the remaining parameters are the same as above. The mean of the dice coefficients for each image in the validation set is used as the network performance evaluation index. After the training, the PSO algorithm is used to iteratively search for the optimal network structure. Finally, the optimal network structure on the origin training set is retrained to obtain the pruned UNet model. We compared the performance indicators of the pruned UNet and the original UNet, and the results are shown in Table 4. When μ is set to 30%, we can remove 78.34% FLOPs and 75.1% parameters still keep the dice score at 0.9945, even 0.002 higher than the original model. Figure 5 shows the segmentation results of 30%SuperPruner-UNet on the test image.

Apart from this, we tested the speed of the model. The input picture resolution is 959×640, the original UNet divides 249 pictures per second, but 30%SuperPruner-UNet can split 311 pictures. The pruned network is about 20% faster than the original network. Through the above analysis, the pruned network has achieved better results on the test set than baseline. Our proposed algorithm is also effective in semantic segmentation tasks.

5. Ablation Studies

To further illustrate the efficiency of SuperPruner in searching for the optimal network structure, we choose VGG16 for an ablation experiments.

5.1. Effect of the VerifyNet. We designed two sets of experiments. A set of experiments does not train VerifyNet and directly uses the PSO to search for the optimal network structure. We retrained the searched network structure for four epochs to obtain network performance. We define this experiment as PSOPruner. Another set of experiments trains VerifyNet and uses the PSO to search for the optimal network structure. VerifyNet is used to predict network performance of the searched network structure. We define this experiment as SuperPruner. To ensure the fairness of comparison, all the parameters of PSO are the same. We

Table 3: Accuracy and pruning ratio of ResNet56 on CIFAR.

Method	CIFAR-10			CIFAR-100		
	Prune Acc (%)	Acc drop (%)	FLOPs/PR (%)	Prune Acc (%)	Acc drop	FLOPs/PR
FPGM [42]	93.49	0.42	59.40M/52.60	69.66	1.75	59.40M/52.60
GAL-0.8 [41]	91.58	1.68	49.99M/60.20	—	—	—
TAS [21]	92.81	1.54	59.50M/52.70	72.25	0.93	61.20M/51.30
ABCpruner-70% [19]	93.23	0.03	58.54M/54.13	—	—	—
SuperPruner-80%	92.17	1.10	24.17M/80.74	68.97	2.35	50.00M/57.30

Table 4: Dice score, pruning ratio, and inference time of UNet on the Carvana data. Dice score is the mean of the dice coefficients for each image in the test set.

Model	Dice score	FLOPs/PR	Parameters/PR	Time (s)
UNet	0.9928	374.47G/-	17.27M/-	0.24
30%SuperPruner-UNet	0.9944	81.11G/78.34%	4.3M/75.1%	0.19
50%SuperPruner-UNet	0.9896	162.23G/56.68%	5.88M/65.95%	0.21
70%SuperPruner-UNet	0.9926	237.26G/36.64%	12.74M/26.23%	0.22

Figure 5: Example semantic segmentation result using 30%SuperPruner-UNet.

assume $M = 10$ and $N = 100$, and the results are shown in Table 5. Compared with PSOPruner, SuperPruner saves twice the time spent. The time saved will increase with the increase in M and N or retraining time. This is because SuperPruner only needs to train VerifyNet once (2000 rounds), and only one forward propagation is needed to get network performance. And PSOPruner needs four rounds of retraining to get network performance for every fitness calculation of a path ($M{\cdot}N{\cdot}$ 4 rounds in total). For SuperPruner, network evaluation and structure search are decoupled. If the hardware requirements change, it only needs eighteen minutes to get the optimal network structure under the new constraints. However, the PSOPruner algorithm needs to start training from zero to get the final result, and it takes about 20 h.

5.2. Effect of μ. We compare the pruned network with different μ. The experimental results are shown in Figure 6. It can be easily found from the figure that as μ increases, the prune rate of FLOPs and parameters will decrease, but the accuracy will increase. We conjecture this because as μ rises, more and more convolution kernel channel will be saved, and the prune model obtained by the SuperPruner is sufficient for image feature extraction. Hence, we can change μ or customize the fitness function to get pruned model that satisfies constraints.

5.3. Comparison with Other Methods on Time Consumption. We also analyzed ABCpruner [19] and Metapruner [25] on time consumption. The experimental results are shown in Table 5. ABCpruner directly uses the ABC algorithm to search for the optimal network structure. Retraining is also used when calculating fitness. If the ABCpruner and PSOPruner parameter settings are the same, the time overhead of the two algorithms is basically the same. Metapruner trains an auxiliary network PrunerNet, which is used to accelerate the calculation of the fitness of the searched network. In the training time of the auxiliary network, PrunerNet training 2000 rounds requires 25 h, while VerifyNet only requires 10 h. This is because the output of PrunerNet is the weight of the network structure. The output of VerifyNet is the accuracy of the network structure on a given dataset. In contrast, VerifyNet has a simpler structure and fewer parameters. It takes less time during training. In the calculation of fitness, Metapruner needs 3.01 s for one calculation and SuperPruner needs 2.20 s. This is because Metapruner requires PrunerNet to perform a forward propagation to predict the weight of the network structure. Through the predicted weights, the forward propagation is performed again to get the accuracy on the given dataset. However, VerifyNet only needs to perform forward propagation once to obtain the accuracy of the network structure on a given dataset. Through the above analysis, it can be concluded that

Table 5: Comparison results of SuperPruner and automatic pruning algorithm in time.

Experiments	Train time (h)	Count one fitness time (s)	Search time (h)	Total running time (h)
PSOPruner	0	74.2	20.6	20.6
SuperPruner	10	2.2	0.6	10.6
ABCpruner [19]	0	74.2	20.6	20.6
Metapruner [25]	25	3.01	0.8	25.8

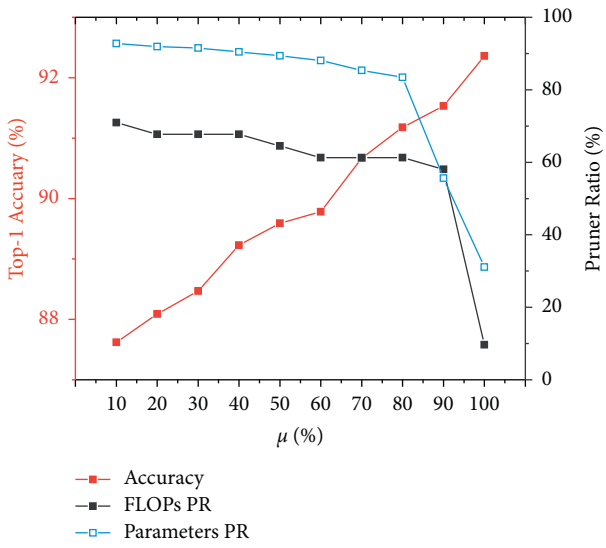

Figure 6: The effect of μ for VGG16 on CIFAR-10. There is no fine-tuning for Top-1 accuracy.

SuperPruner is significantly better than Metapruner and ABCpruner in running time.

6. Conclusions

In this paper, we introduce an efficient automatic pruning algorithm, named SuperPruner. SuperPruner introduces VerifyNet to predict the performance of the network structure, speeding up the search for the optimal network structure. On multiple datasets, SuperPruner is able to achieve higher pruning rate than other state-of-the-art method with little loss of accuracy. Compared with the automatic pruning algorithm, our proposed algorithm has a significant improvement in the pruning speed. Even if the hardware limitation changes, the pruning model can be obtained quickly. More importantly, SuperPruner can be efficiently applied in multiple fields, such as object recognition and semantic segmentation.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This work was supported by the State Grid Zhejiang Electric Power Co., Ltd Innovation Project (No. 5211NB1900VN).

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[2] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2014, https://arxiv.org/abs/1409.1556.

[3] C. Szegedy, W. Liu, Y. Jia et al., "Going deeper with convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, Boston, MA, USA, June 2015.

[4] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *InProceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, Las Vegas, NV, USA, June 2016.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *Advances in Neural Information Processing Systems*, vol. 28, pp. 91–99, 2015.

[6] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: unified, real-time object detection," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 779–788, Las Vegas, NV, USA, June 2016.

[7] A. Wong, M. Famuori, M. J. Shafiee, F. Li, B. Chwyl, and J. Chung, "Yolo nano: a highly compact you only look once convolutional neural network for object Detection," 2019, https://arxiv.org/abs/1910.01271.

[8] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, Boston, MA, USA, June 2015.

[9] O. Ronneberger, P. Fischer, and T. Brox, "U-net: convolutional networks for biomedical image segmentation," in *Proceedigs of the International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 234–241, Springer, Munich, Germany, October 2015.

[10] P. Li, H. Zhao, P. Liu, and F. Cao, "Rtm3d: real-time monocular 3d detection from object keypoints for autonomous driving," 2020, https://arxiv.org/abs/2001.03343.

[11] M. Jaderberg, A. Vedaldi, and A. Zisserman, "Speeding up convolutional neural networks with low rank Expansions," 2014, https://arxiv.org/abs/1405.3866.

[12] Y.-D. Kim, E. Park, S. Yoo, T. Choi, L. Yang, and D. Shin, "Compression of deep convolutional neural networks for fast and low power mobile applications," 2015, https://arxiv.org/abs/1511.06530.

[13] M. Courbariaux and Y. Bengio, "Binarynet: training deep neural networks with weights and activations Constrained To+ 1 or- 1," 2016, https://arxiv.org/abs/1602.02830.

[14] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: training deep neural networks with binary weights during propagations," in *Proceeedings of the Advances in Neural Information Processing Systems*, pp. 3123–3131, Montreal, Canada, December 2015.

[15] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: imagenet classification using binary convolutional neural networks," in *Proceedings of the European Conference on Computer Vision*, pp. 525–542, Springer, Amsterdam, Netherlands, October 2016.

[16] S. Han, J. Pool, J. Tran, and W. J. Dally, "Learning both weights and connections for efficient neural networks," 2015, https://arxiv.org/abs/1506.02626.

[17] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: a data-driven neuron pruning approach towards efficient deep architectures," 2016, https://arxiv.org/abs/1607.03250.

[18] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "Amc: automl for model compression and acceleration on mobile devices," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 784–800, Munich, Germany, September 2018.

[19] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, and Y. Tian, "Channel pruning via automatic structure search," 2020, https://arxiv.org/abs/2001.08565.

[20] S. Lin, R. Ji, Y. Li, Y. Wu, F. Huang, and B. Zhang, "Accelerating convolutional networks via global & dynamic filter pruning," *In IJCAI*, vol. 2, no. 7, p. 8, 2018.

[21] X. Dong and Y. Yang, "Network pruning via transformable architecture search," in *Proceedings of the Neural Information Processing Systems (NeurIPS)*, pp. 760–771, Vancouver, Canada, December 2019.

[22] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," 2015, https://arxiv.org/abs/1503.02531.

[23] S. Lin, R. Ji, C. Chen, D. Tao, and J. Luo, "Holistic cnn compression via low-rank decomposition with knowledge transfer," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 41, no. 12, pp. 2889–2905, 2018.

[24] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning Filters for Efficient Convnets," 2016, https://arxiv.org/abs/1608.08710.

[25] Z. Liu, H. Mu, X. Zhang et al., "Metapruning: meta learning for automatic neural network channel pruning," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 3296–3305, Seoul, South Korea, April 2019.

[26] J.-H. Luo and J. Wu, "AutoPruner: an end-to-end trainable filter pruning method for efficient deep model inference," *Pattern Recognition*, vol. 107, Article ID 107461, 2020.

[27] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," 2018, https://arxiv.org/abs/1810.05270.

[28] Y. Wang, X. Zhang, L. Xie et al., "Pruning from scratch," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 7, pp. 12 273–12280, 2020.

[29] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," 2016, https://arxiv.org/abs/1611.01578.

[30] H. Pham, M. Guan, B. Zoph, Q. Le, and J. Dean, "Efficient neural architecture search via parameters sharing," in *Proceedings of the International Conference on Machine Learning. PMLR*, pp. 4095–4104, Stockholm, Sweden, July 2018.

[31] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model architecture search through hypernetworks," 2017, https://arxiv.org/abs/1708.05344.

[32] Z. Guo, X. Zhang, H. Mu et al., "Single path one-shot neural architecture search with uniform sampling," in *Proceedings of the European Conference on Computer Vision*, pp. 544–560, Springer, Glasgow, UK, August 2020.

[33] L. Li and A. Talwalkar, "Random search and reproducibility for neural architecture search," pp. 367–377, Uncertainty in Artificial Intelligence. PMLR, 2020.

[34] S. Han, H. Mao, and W. J. Dally, "Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding," 2015, https://arxiv.org/abs/1510.00149.

[35] T. Zhang, S. Ye, K. Zhang et al., "A systematic dnn weight pruning framework using alternating direction method of multipliers," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 184–199, Munich, Germany, September 2018.

[36] J.-H. Luo, J. Wu, and W. Lin, "Thinet: a filter level pruning method for deep neural network compression," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5058–5066, Venice, Italy, October 2017.

[37] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1389–1397, Venice, Italy, October 2017.

[38] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," 2016, https://arxiv.org/abs/1608.03665.

[39] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, Venice, Italy, October 2017.

[40] T. P. Lillicrap, J. J. Hunt, A. Pritzel et al., "Continuous control with deep reinforcement learning," 2015, https://arxiv.org/abs/1509.02971.

[41] S. Lin, R. Ji, C. Yan et al., "Towards optimal structured cnn pruning via generative adversarial learning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2790–2799, Long Beach, CA, USA, June 2019.

[42] Y. He, P. Liu, Z. Wang, Z. Hu, and Y. Yang, "Filter pruning via geometric median for deep convolutional neural networks acceleration," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4340–4349, Long Beach, CA, June 2019.