Evolving Filter Criteria for Randomly Initialized Network Pruning in Image Classification

Xiangru Chen[a,b,1], Chenjing Liu[a,b,1], Peng Hu[a], Jie Lin[b], Yunhong Gong[a], Yingke Chen[d], Dezhong Peng[a,c,*] and Xue Geng[b,*]

[a]College of Computer Science, Sichuan University, 610065, Chengdu, China
[b]Institute for Infocomm Research, Agency for Science, Technology and Research, 138632, Singapore
[c]Sichuan Newstrong UHD Video Technology Co., Ltd., 610095, Chengdu, China
[d]Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne NE1 8ST, UK

ARTICLE INFO

Keywords:
genetic algorithm
evolving deep learning
network pruning
multi-objective evolutionary algorithm

ABSTRACT

Network pruning aims to enhance the performance of deep neural networks by eliminating redundant components from the model. However, existing pruning methods typically require a well-trained model and employ fixed, single pruning criteria throughout the pruning cycles. To address these limitations, we propose a novel method called Evolutionary Filter Criteria (EvoFC). This method enables the automated search for the network pruning ratio and criterion during a population-based heuristic search process. We introduce a unique encoding space that represents the chosen pruning criterion and ratio for each layer, facilitating the acquisition of optimal architecture configurations for candidate networks during iterations. Additionally, we devise a novel weight inheritance mechanism to mitigate the computational burden associated with the population-based nature of the method, resulting in a significant reduction in overall training time. We validate our method by applying it to randomly initialized networks and conducting empirical experiments on CIFAR-10/100, ILSVRC-2012 and Places365 datasets. The results demonstrate that our method effectively reduces the number of FLOPs while striking a fine balance between accuracy and computational efficiency. This underscores the practical value of our method in optimizing performance while efficiently utilizing computational resources, particularly when pruning networks starting from random initialization.

1. Introduction

The success of Deep Neural Networks (DNNs) has in fact propelled deep learning into a current paradigm for solving the problems in artificial intelligence [24], especially for the image classification [10] task. Although the performances of the DNNs are encouraging and promising, the bottlenecks of the huge computational complexity and extensive memory consumption are the dominant issues in deploying the DNNs on the practical applications. This problem is particularly prominent in some resource-constrained devices, such as the mobile phones, and embedded processors [4]. Hence designing efficient network architecture to alleviate the huge computational consumption while keeping the accuracy has attracted tremendous attention.

The community has taken many efforts for reducing the computational cost of the DNNs, such as Architecture Design [55], Low-rank Decomposition [21], Knowledge-distillation [48], Quantization [17] and Neural Architecture Search [15]. As a remedy, network pruning [29] is designed to remove the redundant weights to form a lighter network architecture. Due to the over-parameters which is widely-recognized as a property in very deep neural networks, pruning has been adopted as one of the efficient compression methods in reducing the network parameters and speeding up the inference [35]. In some scenarios [9, 26], it can achieve more than 90% redundancy removal.

A widely-used pruning procedure usually follows the next three steps: 1) Train the DNN model from scratch to prepare a well-trained model on some prevalent datasets (some weights of well-trained models are usually available

*Corresponding authors
cxrasdfg@gmail.com (X. Chen); cjliu1996@gmail.com (C. Liu); penghu.ml@gmail.com (P. Hu); jie.dellinger@gmail.com (J. Lin); gongyunhong@stu.scu.edu.cn (Y. Gong); yingke.chen@northumbria.ac.uk (Y. Chen); pengdz@scu.edu.cn (D. Peng); geng_xue@i2r.a-star.edu.sg (X. Geng)
ORCID(s):
[1]These authors contributed equally to this work

and can be downloaded directly from the public repositories). 2) Then, prune the larger trained model according to some criterion. 3) Lastly, fine-tune the pruned models to recover the original performances as well as possible.

A well-trained model is often necessary for the following pruning step [27, 51, 30], which means the extra training time should be appended to the whole procedure even when some small architecture modification on the original well-trained model. However, training a DNN model from scratch is always time-consuming, and it will increase the cost for pruning such a model. This is usually infeasible especially for those based on the hessian matrix, where the first order of Taylor expansion series is not zero for a randomly initialized models [53]. Fortunately, researches show that a randomly initialized neutral network can be pruned without pre-training, and it could gain competitive performances compared to the pruning method requiring pre-training, such as SNIP [25] and methods based on Lottery Ticket Hypothesis [7].

Another key factor of the network pruning is the selection of the pruning criterion. The pruning criterion is designed to identify which parameters in the model are useful or less useful. Nevertheless, the previous works manually specify the a single pruning criterion, and employ that criterion for all the different layers in the DNN model. But, recent advances showed that the different layers have diverse filter distribution [11]. Specifically, the shallow layers of the neural network only process some low-level visual notions such as edge, corner, line and etc. While the deep layers extract the high-level features (shapes and objects) [33]. Therefore, fixing one criterion to prune different layers is not suitable in the real application, which neglects the diversity of each layer and declines in performance.

In addition, the other limitation of current works is that only one single model is obtained when configured. When end users want to define a new compression goal (E.g., change the pruning coefficient ϵ) or deploy on the new target platform, they have to rerun the requested pruning methods to gain a new final pruned models [27, 25], which costs more time for adjustments.

In this paper, we proposed a novel method called Evolving Filter Criteria (EvoFC)[1]. to resolve the aforementioned problems. The presented EvoFC is a nouveau structured pruning method for removing the redundant filters of the target model on image classification, and it can automatically optimize the target pruning criterions and the corresponding pruning ratios. Compared with previous work, our method does not assume that the initial network is well trained on the target datasets. Therefore, for a custom network architecture without public weights for downloading, our method can directly perform the network pruning for the desired model. It is worth noting that, the combinations of the pruning criteria and the target pruning ratios are exponential, hence the traditional gradient-based method is not effective in this scenario. As a result, we decide to choose the meta-heuristic methods, i.e., the genetic algorithms as the optimization method due to its gradient-free merit and non-sensitivity to the local minimal [45, 8, 54]. Specifically, the core framework of the EvoFC is driven by the Multi-objective Evolutionary Algorithms (MOEAs) [3], and we explore the MOEA to solve such non-convex optimization problem in the criterion selections and pruning ratio assignment. The details of the MOEA will be described in the later sections. In order to prevent the hyper-parameters from manually tuning of previous works and introduce adaptive filter criteria in the network pruning procedure, we design a new encoding space to encode the network pruning criteria combined with the pruning ratios. This design will let the candidate pruning configurations automatically evolve from a randomly initialized population, and find a better hyper-parameter combination during each iteration. However, despite the advantages of evolutionary algorithms as mentioned above, population-based algorithms have a natural disadvantage in evolving deep learning that the evaluating of each candidate network will be very time-consuming, since each candidate solution is an independent neural network and must be trained one-by-one for evaluation. This incurs the large computational burden especially when increasing the population size and generations. As a remedy, we proposed a weight inheritance mechanism to alleviate this problem by sharing the weights of candidate networks, which largely reduces the total searching time. The main contributions of this paper are summarized as follow:

1. We proposed a new MOEA-based method for the network pruning of randomly initialized convolutional neural network (CNN), in which accuracy versus complexity of the Pareto-optimal solutions can be automatically obtained.
2. We designed a novel encoding space to encode the pruning criteria and pruning ratios, so that the proposed method can automatically search the optimal criterion for each layer in the target convolutional neural networks without pre-training.
3. We developed a novel weight inheritance mechanism to alleviate the computational complexity of the EA-based pruning methods, which significantly reduce the searching time.

[1] The source code for the implementation is available online: https://github.com/cxrasdfg/evofc

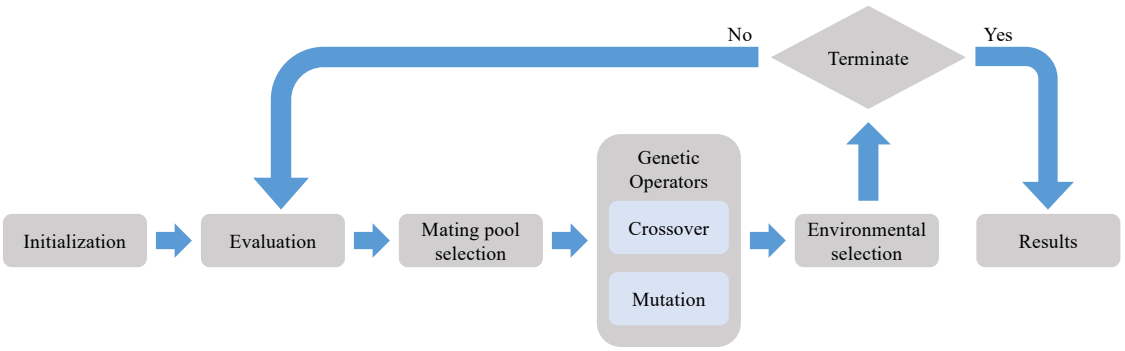


Figure 1: Basic flow chart of genetic algorithm.

4. The experiments show that the proposed methods can significantly compress the target network architecture under the randomly initialized condition on the prevalent CIFAR and ImageNet datasets without the human intervention.

The rest of this paper is organized as follows. Section 2 introduces the background and related work of this work. Then, in Section 3, the details of the proposed EvoFC will be presented. In Section 4, we elaborate the experimental settings and the final results of the EvoFC, which demonstrate the promising performances of the proposed method. Lastly, we will summarize our work in Section 5.

2. Background and Related Work

2.1. Genetic Algorithm

Genetic Algorithm (GA) [16] is a kind of heuristic population-based evolutionary algorithm, the other two are genetic programming (GP) [22], evolutionary strategy [20]. Base on initial population, GA can recombine the existing solutions to find better ones by employing crossover and mutation. Due to the gradient-free and in-sensitiveness to local minima, GA is a natural choice for non-convex and non-differentiable problems, where a high-quality solutions can commonly be obtained. A typical GA based algorithm usually includes the following steps:

1. Initialize the population according to a specific gene encoding strategy, each solution are encoded as an individual in the initial population.
2. Evaluate the current population to get the fitness of the newly individuals in the population, a designed evaluation mechanism is employed.
3. Select candidates to construct the mating pool, which will be used to selected pair individuals as parents.
4. Evolving on the selected parents to generate the offspring using two genetic operators: crossover and mutation.
5. Employ environmental selections to reform population for the next generation.
6. Repeat step 2 and step 5 until the stop criterion is satisfied.

Fig. 1 draws the flow chart of a typical GA.

Some methods only concern the model accuracy as the target metric [45], hence the single-objective GA is sufficient to solve the optimization problem. However, in most practical applications, users must consider extra objectives, in addition to the model accuracy, when they employed the DNN models on some resource-constrained devices. Therefore, the Multi-Objective Evolutionary Algorithms (MOEAs) are designed to solve the constrained problems under several objectives [36]. Compared with single-objective GA, multi-objective GA must resolve how to rank and select each candidates with multiple objectives [3].

2.2. Network Pruning

As a widely used method for network compression, network pruning can be generally categorized into unstructured and structured pruning [29]. The unstructured pruning methods often make the less useful weights to be zero, then the global weight would become very sparse matrices. Thus, the pruned model can be accelerated by employed the fast

sparse matrix operations. E.g., Zhang et al. proposed to prune the DNN model with the alternating direction method of multipliers (ADMM), where they converted the pruning problem of DNNs into a non-convex optimization problem. They employed the ADMM to resolve the non-convex obstacle, and pruned the model iteratively. Inspired by their work, Ren et al. proposed to combine ADMM with the quantization for the DNN compression. During pruning procedure, the proposed method also took both of the computation reduction and energy efficiency improvement into consideration. They achieved nearly $2,000\times$ model reduction on the Lenet-5 model. Yao et al. [50] proposed Balanced Sparsity, a novel fine-grained sparsity approach, for the compression of pretrained DNN models. The balanced sparsity can obtain a high compression ratio with considerable accuracy. However, the disadvantage of the unstructured pruning method is that it will destroy the consistence of regular structures in DNN models, such as the kernels, filters. This will incur extra cache or memory access problems [19]. It also requires the support of special hardware to maximize the efficiency of sparse operators [17].

Compared with unstructured pruning, structured pruning methods are more concerned with trade-offs at larger granularity, such as filters or channels in CNNs. Specifically, based on the various structures, structured pruning can often be classified as kernel-level pruning, filter-level pruning and channel-level pruning [39]. Taking filter-level pruning as an example, the minimum structure of pruning is not a single element of weights, but all elements of a filter as a whole are retained or removed. E.g., Wen et al. [49] used the ℓ_1-norm to regularize the structure in convolutional layers, and the learned structures of filters will be leveraged to form a more compact architecture for computational cost reduction. Besides, Zhou et al. [57] also considered the sparse property in deep CNNs. They imposed the sparse constraints on the objective function during training, and employed the forward-backward splitting method to resolve the sparse optimization problem for the filter removal. He et al. [12] proposed SFP to accelerate the deep CNNs, where the proposed method enabled the pruned filter to be updated during the fine-tuning procedure. You et al. [51] proposed GBN, an iterative global filter pruning method. GBN employed the channel-wise scaling factor to indicate which filter should be eliminated. To improve the final accuracy, they also proposed a *Tick-Tock* mechanism to perform the iterative pruning procedure. Lin et al. [30] developed Hrank, in which they pruned the DNN models via removing the low-rank feature maps, by considering that the low-rank features contain less useful information. In addition, some special structured pruning method do not take the common structure (i.e., kernel, filter and channel) as the base unit to be pruned. Meng et al. [38] proposed the pruning filter in filter method, which is an iterative pruning method that their pruning granularity is between the unstructured pruning and structured pruning. It has consistent hardware compatibility with structured pruning while having a large compression ratio.

However, most methods, whether structured or unstructured, assume an available model weight and prune the model based on the well trained weight at the beginning. Sometimes, this could be convenient when some common used architectures, such as VGGNet [44] or Resnet [10] pretrained on the ILSVRC-2012 [41], are decided to be pruned. But when extra tasks with diverse‘ datasets or different models with customer architectures are considered, this could be very arduous for training the model from scratch, such as more computational resources and training time.

Fortunately, recent works demonstrate that pruning the randomly models with randomly initialized weights are feasible. Lee et al. [25] proposed SNIP, where they sampled a simple batch from the training data to calculate the connection sensitivity of each element in the whole model. Then the top-κ elements with the highest sensitive scores will be retained for the pruned model. The whole procedure excludes the extra pre-training preliminary, so that almost any customer larger DNN models can be pruned without the public well-trained weight. Except the gradient-based SNIP, *Lottery Ticket Hypothesis* [7, 5, 2] is also recently proposed to prune the DNN models with random initialization. Frankle et al. [7] proved that randomly initialized dense networks contained subnetworks that, once found, could be trained to achieve comparable test accuracy to the original trained dense networks. Inspired by their work, Cunha et al. [2] demonstrated that it was feasible to approximate any CNNs by pruning a random CNN, which was possessed of a larger size by a logarithmic factor.

2.3. Network Pruning Criterion

Pruning criterion is the key component for pruning a DNN model. To be more specific, the pruning criterion indicates which parts of the model are less useful and should be removed for the computation reduction. According to the different methods of calculating importance scores, pruning criterion can be roughly divided into two categories, i.e., the weight-based criteria and activation-based criteria [11].

Activation-based criteria employ the activated features to determine which filter can be removed. Liu et al. [34] introduced a scaling factor, and the scaling factor was multiplied by each output channel. They compressed the target model by pruning channels with smaller scaling factors. Luo et al. [37] presented ThiNet, which was a pruning method

by considering the statistical information from next layer to guide the filter selection. Additionally, Molchanov et al. [40] proposed a novel importance estimation method, where the contribution of each elements to the final loss was approximated via the first and second order Taylor expansions.

Weight-based criteria use the weight of the filter to adjudicate the importance of filters. Some straightforward methods assume the larger elements should be considered as the more important ones to be remained. E.g., the magnitude-based methods perform the ℓ_1-norm [27] or ℓ_2-norm [12] criteria on the weight to determine which elements should be pruned. Furthermore, He et al. [13] presented a new weight-based criterion called FPGM. The proposed FPGM employed the Geometric Median in the filters as a reference for the following pruning. Specifically, they assumed the filters near the geometric median were redundant and should be dropped.

2.4. Preliminary & Motivation

For the simplicity of the following description, we firstly assume a L-layer plain CNN model m with different configurations of each layer, and let the whole weights in the model be $\mathcal{W} = \{W_1^f, W_2^f, \cdots, W_L^f\}$. For the weights in the i-th layer, we can denote them by a set of 3-D tensors $W_i^f = \{W_{i,1}^f, W_{i,2}^f, \cdots, W_{i,c_i}^f\} \in \mathbb{R}^{c_i \times c_{i-1} \times k_i \times k_i}$, where the symbol k_i denotes the kernel size, c_{i-1} is the input channel size, and c_i is the output channel size. Beside, we use $W_{i,j}^f \in \mathbb{R}^{c_{i-1} \times k_i \times k_i}$ to represent the weights of the j-th filter in the i-th layer. For the convolutional operation of the i-th layer, let the input features be $I_i \in \mathbb{R}^{c_{i-1} \times h_{i-1} \times w_{i-1}}$, and h, w be the height and width of the feature map, respectively. Then the output features O_i can be obtained by:

$$O_i = \text{cat}\left(I_i * W_{i,1}^f, \cdots, I_i * W_{i,c_i}^f\right) \in \mathbb{R}^{c_i \times h_i \times w_i}, \tag{1}$$

where $*$ denotes the convolution, and the operator 'cat' is to concatenate all the features along the channel axis. Please note that we omit the bias in the equation for the simplicity.

The objective of pruning is to remove the non-sensitive filters to the loss while keeping the final accuracy as much as possible. For the convenience of description, let $\mathcal{S}^f = \{S_i^f\}_{i=1}^L$ be the indices of all the filters, and vector $S_i^f = [0, 1, \cdots, c_i - 1] \in \mathbb{R}^{c_i}$ be the complete indices for the filters of the i-th layer. Afterwards, the complete index set of the filters $\mathcal{S}^f$ can be divided into two disjoint subsets $\mathcal{S}_r^f$ and $\mathcal{S}_p^f$. Specifically, $\mathcal{S}_r^f$ refers to the index set of remained filters, and $\mathcal{S}_p^f$ refers to that of the pruned filters. They satisfy the following formulas:

$$\begin{aligned} &\mathcal{S}_r^f = \{\hat{S}_i^f\}_{i=1}^L,\ \mathcal{S}_p^f = \{\overline{S}_i^f\}_{i=1}^L \\ &s.t.\ \hat{S}_i^f \cup \overline{S}_i^f = S_i^f,\ \hat{S}_i^f \cap \overline{S}_i^f = \emptyset \quad \forall i \in [1\,..\,L], \end{aligned} \tag{2}$$

To minimize the loss function on a given training dataset $\mathcal{D}_{trn} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ containing N pair samples, the generic objective function of pruning such a CNN model can be written as:

$$\begin{aligned} (\mathcal{S}_r^f)^* &= \min_{\mathcal{S}_r^f} \sum_{i=1}^N \ell(\boldsymbol{x}_i, y_i; \mathcal{W}, \mathcal{S}_r^f) \\ s.t.\quad & \frac{\sum_{l=1}^L |\hat{S}_{l-1}^f||\hat{S}_l^f|k_l^2}{\sum_{l=1}^L c_{l-1}c_l k_l^2} \le r, \end{aligned} \tag{3}$$

where ℓ denotes the loss function, and r denotes the compression ratio. Besides, we define $|\hat{S}_0^f| = c_0$, which denotes the channel size of the data samples.

Eqn. 3 is the generic objective function in most pruning methods, and how to find an adequate solution for the set $\mathcal{S}_r^f$ is the core problem in modern compression methods [30, 11]. To solve this problem, a particular pruning criterion is often adopted to guide the filter selection. Suppose a criterion $\mathbf{f}_i$:

$$\mathbf{f}_i : \mathbb{R}^{c_i \times c_{i-1} \times k_i \times k_i} \to \mathbb{R}^{c_i}, \tag{4}$$

which is a function to measure the sensitivity scores of the filters with respect to the final loss ℓ. Then we can simply sort the filters and select the Top-k filters through importance scores obtained by the criterion [53, 25]. However, most

exiting methods only consider one criterion to solve the Eqn. 3, and the compression rate r is also manually predefined for each layer. This hand-craft fixed strategy might be not optimal for the whole network due to the different filter distributions across the diverse layers [11]. Furthermore, when a different pruning rate is required, the entire algorithm needs to be re-run. To solve the limitations, and assign different criteria and pruning rates for each layer in the DNN model, we introduce a list of the criterion $F^c = [\mathbf{f}_1, \mathbf{f}_2, \cdots, \mathbf{f}_L]$ and a vector of the pruning ratio $\boldsymbol{r} \in \mathbb{R}^L$ into the original objective function. Then the Eqn. 3 can be rewritten as:

$$\begin{aligned} &\min_{F^c, \boldsymbol{r}} \sum_{i=1}^{N} \ell(\boldsymbol{x}_i, y_i; \mathcal{W}, F^c, \boldsymbol{r}) + \alpha \sum_{l=1}^{L} r_l \\ &s.t. \quad \frac{|\hat{S}_{l-1}^f||\hat{S}_l^f|k_l^2}{c_{l-1}c_l k_l^2} \leq r_l \quad \forall l \in [1 \,..\, L], \end{aligned} \tag{5}$$

where α is a scaling factor.

Please note that the set of the kept filters S_r^f is determined by the weight $\mathcal{W}$, the criteria F^c and the pruning rates $\boldsymbol{r}$:

$$S_r^f = \{\text{topk}(\mathbf{f}_l(W_l^f), c_l r_l) | l \in [1 \,..\, L]\}, \tag{6}$$

where the "topk($\boldsymbol{a}, n$)" function will return the indices of the Top-n largest values in the vector $\boldsymbol{a}$.

However, Eqn. 5 is non-differentiable and optimizing this problem over the given dataset $\mathcal{D}_{trn}$ is NP-hard [1, 31], which is very hard to be directly solved by the modern optimization tool, such as the gradient descend. Furthermore, Eqn. 5 is also a multi-objective optimization problem where we want to seek a fairly good compression configurations while keeping the accuracy as much as possible. Inspired by recent works where the evolutionary algorithms can be used to solve the non-convex optimization problem in DNN models [45], we propose to employ the MOEA to optimize the raised NP-hard problem by leveraging the good merit that EAs are insensitive to the local minima and gradient-free. What's more, by incorporating the MOEAs, we can also obtain various pruning configuration on the Pareto front only run the method once, which is convenient for the end-users with diverse hardware platforms. Please note that this time-saving feature is not always coming with most previous researches, such as [25, 51], because they must be rerun to obtain the new pruned models if new compression target is acquired.

But this will lead to a new performance bottle-neck when the EAs are employed due to its population-based mechanism. Specifically, to obtain the fitness of each individual in the population which represent the pruning configuration, each individual should be evaluated independently. That means a lot of models will be trained and evaluated. Although training a single model in the modern hardware could be in one GPU day [45]. But if 100 models need to be evaluated, it is still requires about 100 GPU days, which is infeasible in the real applications. This computational bottle-neck motivates us to develop a training mechanism, called Weight Inheritance, during the evolutionary loop, to accelerate the searching process. This mechanism will be elaborated in the next section.

What's more, another limitation of existing methods is that they often heavily rely on the pretrained models before optimization the Eqn. 3. As we mentioned in Subsection 2.2, whether structured or unstructured, most of the existing pruning-based methods must satisfy the condition that the model to be pruned is well pretrained on the target dataset [27, 51, 30, 37], which induces extra training consumption when customer network architecture is adopted. To resolve this concern, the proposed method is designed to be capable of pruning the CNN models with random initialization. Therefore, Eqn. 5 can be rewritten as:

$$\begin{aligned} &\min_{\mathcal{W}, F^c, \boldsymbol{r}} \sum_{i=1}^{N} \ell(\boldsymbol{x}_i, y_i; \mathcal{W}, F^c, \boldsymbol{r}) + \alpha \frac{\sum_{l=1}^{L} c_{l-1}c_l r_l k_l^2}{\sum_{l=1}^{L} c_{l-1}c_l k_l^2} \\ &s.t. \quad \frac{|\hat{S}_{l-1}^f||\hat{S}_l^f|k_l^2}{c_{l-1}c_l k_l^2} \leq r_l \quad \forall l \in [1 \,..\, L], \end{aligned} \tag{7}$$

it is worth noting that minimizing the rightmost term in Eqn. 5 is equivalent to minimizing the rightmost term in Eqn. 7. In order to obtain the final model compression ratio, we can use the global compression ratio to represent the metric of computational complexity for the pruning configuration.

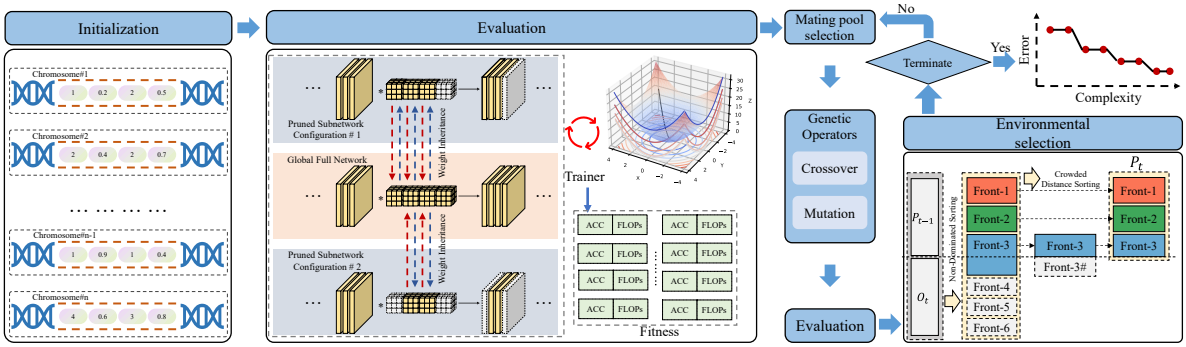


Figure 2: The framework of the proposed EvoFC. 1) Initialization: The pruning configuration parameters in the encoding space are randomly sampled to create initial individuals. 2) Evaluation: Each individual, which includes the pruning information, is evaluated on the corresponding dataset to determine the accuracy rate (ACC) and performance complexity (FLOPs). 3) Mating pool selection: Randomly paired parental chromosomes are selected from the current population to serve as parents for generating new individuals in the next generation. 4) Genetic operators: New offspring individuals are generated by applying crossover and mutation operations to the paired parents. Crossover is considered a global search operator, while mutation is considered a local search operator. 5) Environmental selection: The rank of the existing population and the newly generated individuals is calculated using non-dominated sorting and crowded distance sorting. This allows for the evolution and elimination of individuals to maintain a diverse and high-performing population.

By leveraging the meta-heuristic searching strategy, the proposed method will move toward to optimal direction with best criteria and pruning ratios by each loop step. Finally, we will obtain various well-trained networks with different pruning criteria and compression ratios. This whole procedure is completely evolved from a randomly initialized CNN model.

3. The Proposed Method

3.1. Overall algorithm

The developed EvoFC is a novel multi-objective evolutionary algorithm for pruning any CNN with random initialization. Compared with most existing methods that the models to be pruned are well trained on a specific dataset, our method can prune any customer CNN architectures from scratch. Afterwards, the proposed EvoFC will find a set of candidate solutions with different computational complexity and accuracies on the approximated Pareto front. This is beneficial to the end-users who own multiple platform to be deployed, and the optimal pruned models satisfying the corresponding hardware configurations can be obtained at only once run.

The overall algorithm of the proposed EvoFC is depicted in Algo. 1. Firstly, the population P_0 at the beginning will be initialized by sampling from the proposed encoding space (line 1). Immediately after, the initialized population P_0 will be trained and evaluated with the given model m which possesses of random weights to obtain the fitness F_0, the set of the decoded pruned models M_0 and the updated model m. Next, the algorithm will choose the parent individuals C_t by the binary tournament to form the mating pool (line 4). C_t will be employed to generate the offspring, the new pruning configurations containing the information of pruning criteria and pruning ratios, through the genetic operators (line 5). Similar to the initial population P_0, the new generated individuals O_t should also be evaluated to gain the corresponding fitness and decoded pruned models (line 6). Sequentially, the environmental selection will be performed on the combined set of the evaluated offspring and the population of the last generation to form the current population P_t (line 7). The whole process will be terminated when reaching the stop condition, and output the finally evolved population P_T and the set of well-trained models M_T. The desired pruned models with various hardware demands can be directly selected from the model set P_T without the continuous fine-tuning like the most methods before. To better illustrate the whole procedure of the proposed EvoFC, we draw the overall framework in the Fig. 2.

3.2. Encoding Space

As we mentioned in Section 2.4, Eqn. 5 is hard to be optimized by the gradient-based method. To solve it by the evolutionary algorithm, we should first determine a suitable search space to represent potential reasonable solutions.

Algorithm 1: Algorithm of EvoFC.

Input: The number of generations T, the randomly initialized DNN model m
Output: The candidate pruning solution P_T, the set of the well-trained models M_T

$P_0 \leftarrow$ Initialize the population from the proposed gene encoding space;
$F_0, M_0, m \leftarrow$ evaluate(P_0, m);
for $t = 1$ **to** T **do**
  $C_t \leftarrow$ Select the parent solutions via the binary tournament selection;
  $O_t \leftarrow$ Generate the offsprings with the genetic operators from C_t;
  $F_t, M_t, m \leftarrow$ evaluate(O_t, m);
  $P_t \leftarrow$ Environmental selection from $P_{t-1} \cup O_t$;
end
return P_T, M_T;

Hence, we proposed a novel encoding strategy which encodes the pruning criteria and pruning ratios at the same time. Therefore, we can use the proposed EvoFC to solve $F^c, \boldsymbol{r}$ in Eqn. 5.

In a nutshell, the gene in the criterion part will decide which criterion to choose for a particular layer, while that in the ratio part will determine the amount of the remaining filters in the layer. To define a criterion pool for the following encoding process, we select four criteria for the filter pruning in each layer. Firstly, motivated by the most existing work that they need a well-trained model, a pruning criteria which is not sensitive to the random weights should be introduced into this criterion pool. Inspired by the work presented in [25], the Connection Sensitive (CS) is adopted, since it has been shown to act as an effective pruning indicator for randomly initialized networks.

Specifically, to calculate the CS scores for the filters, we first sample a mini-batch data $\mathcal{D}_{trn}^b = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^b$ from the dataset $\mathcal{D}_{trn}$, where b denotes the sampled batch size for the CS scores. In order to calculate the impact of a filter on loss, we introduce an auxiliary variable $A_l \in \{0,1\}^{c_l \times c_{l-1} \times k_l \times k_l}$ for the filters in the l-th layer. In other words, that variable represents the connectivity of the weight W_l^f. Additionally, let the set of the auxiliary variables be $\mathcal{A}$. For a single criterion, the model loss function Ψ can be written as:

$$\begin{aligned} \min_{\mathcal{W},\mathcal{A}} \Psi(\mathcal{D}_{trn}^b; \mathcal{W}, \mathcal{A}) = \min_{\mathcal{W},\mathcal{A}} \frac{1}{b} \sum_{i=1}^{b} \ell(\boldsymbol{x}_i, y_i; \mathcal{W} \odot \mathcal{A}) \\ s.t. \quad ||\boldsymbol{a}_l||_0 \leq \kappa, \end{aligned} \tag{8}$$

where the symbol $\odot$ represents the Hadamard product. For the Hadamard product between the sets $\mathcal{A}$ and $\mathcal{W}$, define the results as the Hadamard product of the corresponding tensor elements in the two sets. That result can be expressed by:

$$\mathcal{W} \odot \mathcal{A} = \left\{ W_1^f \odot A_1, W_2^f \odot A_2, \cdots, W_L^f \odot A_L \right\}. \tag{9}$$

The connection sensitivity of any j-th element in l-th layer filters W_l^f can be measured by zeroing that element while leaving the rest elements unchanged. Therefore, the impact of the element on the loss is obtained by calculating the absolute value of the loss change before and after. Precisely, the change of the loss value on that element can be expressed by $\Delta\Psi_{l,j}$:

$$\Delta\Psi_{l,j} = \Psi(\mathcal{D}_{trn}^b; \mathcal{W}, \mathbb{1}) - \Psi(\mathcal{D}_{trn}^b; \mathcal{W}, \mathbb{1} - \mathcal{E}_{l,j}), \tag{10}$$

where the weight set represented by the symbol $\mathbb{1}$ is a set of weights that is consistent with the shape of $\mathcal{W}$ and comprises weight values of all ones. Similarly, the symbol $\mathcal{E}$ shares the same definition with $\mathbb{1}$, except that it represents a set of weights where the j-th element of the l-th layer is one, while all other elements are zero. It is important to note that computing all the $\Delta\Psi_{l,j}$ values is computationally expensive. Hence, to alleviate this computational burden, the binary constraint on the auxiliary variables $\mathcal{A}$ is relaxed, and $\Delta\Psi_{l,j}$ is approximated by taking the derivative of Ψ with respect to the j-th element in weight W_l^f. This approximation is denoted by $g_{l,j}(\mathcal{D}_{trn}^b; \mathcal{W})$.

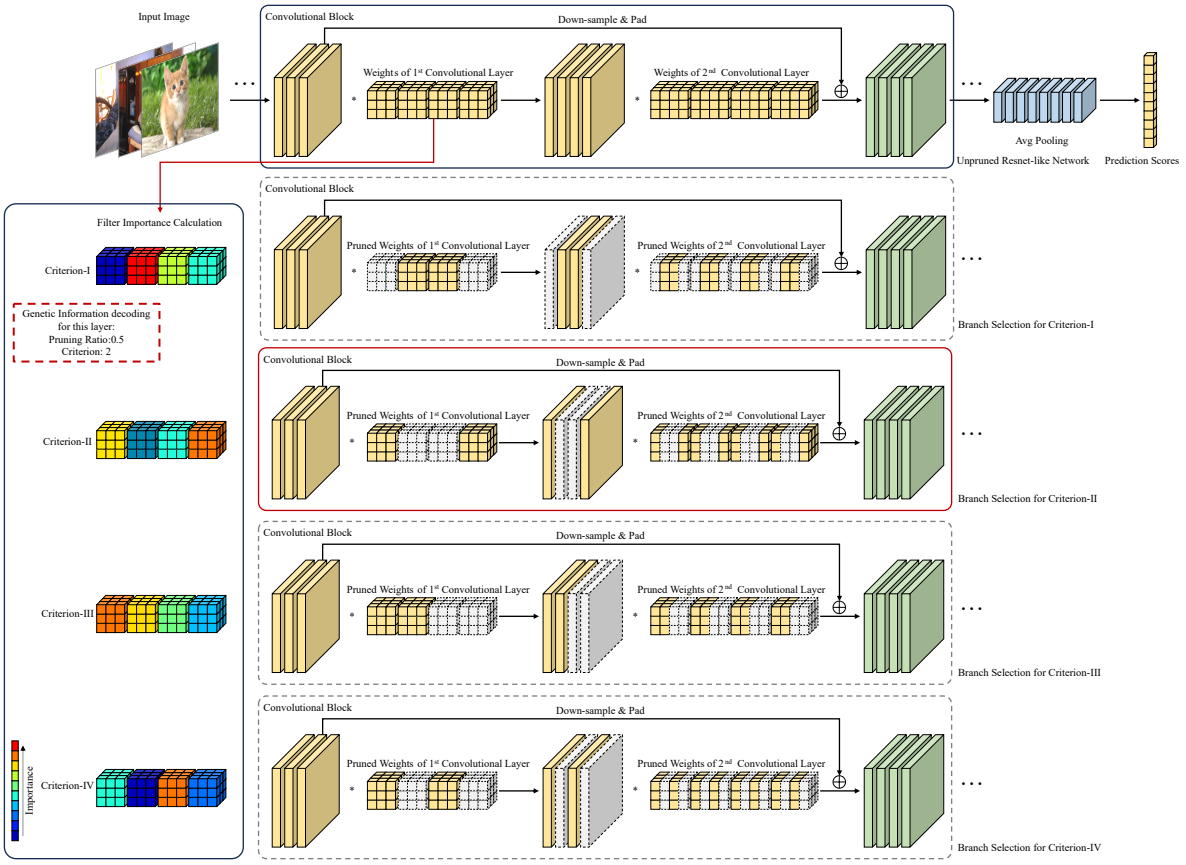


Figure 3: The way to decode the chromosome into a real resnet-like convolutional model. For the convolutional layer in each block, the evaluation process will first extract the gene information of the corresponding sequence of the individual to be evaluated, and convert it into the corresponding pruning rate and criteria information. After calculating the channel score of each pruning criterion, the filters that need to be retained are obtained according to the score under each pruning and rate. The figure shows the pruning process of a residual block with a rate of 50% and a criterion indexded by 2. The pruned block is marked with a red frame.

The gradient $g_{l,j}(\mathcal{D}^b_{trn}; \mathcal{W})$ can be efficiently computed by a single forward-backward process on the mini-batch data $\mathcal{D}^b_{trn}$. By considering all the elements in the weight of the filter, the CS importance score for the i-th filter can be obtained by summing the gradients, as follows:

$$\mathbf{f}_{cs}(W^f_{l,i}) = \sum_{j=1}^{c_{l-1}} \sum_{m=1}^{k_l} \sum_{n=1}^{k_l} \left| g(\mathcal{D}^b_{trn}; w^f_{l,i,j,m,n}) \right| \tag{11}$$

Notably, a higher CS importance score indicates that the filter is more critical to the network's performance. It is crucial to emphasize that this approach differs from that of [25] in that the importance score is calculated for coarser filters, rather than unstructured single elements. This modification is essential to adapt to the pruning of convolutional neural networks, which typically involves removing entire filters, rather than individual weights. Therefore, by evaluating the filter's importance as a whole, this approach provides a more accurate assessment of the filter's contribution to the network's overall performance.

Another criterion we employed is the Geometric Median (GM) [13] metric for the weight of the filter. The goal of the GM criterion is to identify the filters that best represent a given convolutional layer at the median position. To quantify the distance between each filter, we utilize the Euclidean Distance measure, which can be expressed as

follows:

$$\mathbf{f}_{gm}(W_{l,i}^{f}) = \text{sum}(\text{cdist}(\text{norm}(W_{l}^{f}))) \tag{12}$$

What's more, ℓ_1-norm, ℓ_2-norm are the supplement for the following strategies due to their popularity in pruning methods. Although the sensitivity of initialized weights cannot be ignored, it is imperative that the filters with greater importance are assigned larger values after training, as these values play a critical role in determining the final classification results:

$$\mathbf{f}_{\ell_p}(W_{l,i}^{f}) = \left(\sum_{j=1}^{c_{l-1}} \sum_{m=1}^{k_l} \sum_{n=1}^{k_l} |w_{l,i,j,m,n}^{f}|^{p} \right)^{1/p} \tag{13}$$

Fig. 3 illustrates the decoding processes for the chromosomes and the pruned models. Each chromosome fragment contains the pruning criterion and the pruning ratio for that layer.

3.3. Evaluation

Algorithm 2: Evaluate the given population without weight inheritance.

Input: The newly created population O_t, the DNN model m.
Output: The Fitness F_t, the set of the well-trained models M_t, and the auxiliary model m.

```
for i = 1 to population_size do
    m' ← deep copy the model m;
    F^c, r ← decode the i-th individual O_t^i;
    S_r^f ← obtain kept filters via Eqn. 6;
    for j = 1 to epochs do
        for D_trn^b in D_trn do
            Update the weights m' via:
            ∇_W Σ_{(x_i,y_i)∈D_trn^b} ℓ(x_i, y_i, S_r^f; W);
        end
    end
    ACC, FLOPs ← test m' on D_val^b;
    F_t ← F_t ∪ (ACC, FLOPs);
    M_t ← M_t ∪ m';
end
return F_t, M_t, m;
```

The rightmost term in Eqn. 7 represents the global compression ratio of the weight. But in practical situations, the proportion of weights may not necessarily reflect the actual changes of the computational amount. Therefore, in the practical application, we use the global floating point operations (FLOPs) of the pruned models to represent the amount of the computational complexity before and after model pruning. Hence the Eqn. 7 can be accordingly modified as:

$$\begin{aligned} \min_{\mathcal{W},F^c,\boldsymbol{r}} & \sum_{id=1}^{N} \underbrace{\ell(\boldsymbol{x}_i, y_i; \mathcal{W}, F^c, \boldsymbol{r})}_{Accuracy} + \alpha \underbrace{\text{FLOP}(\mathcal{W}, F^c, \boldsymbol{r})}_{Complexity} \\ s.t. \quad & \frac{|\hat{S}_{l-1}^{f}||\hat{S}_{l}^{f}|k_l^2}{c_{l-1}c_l k_l^2} \le r_l \quad \forall l \in [1 \,..\, L], \end{aligned} \tag{14}$$

where the FLOP(*) denotes the FLOPs of the pruned CNN model. The FLOPs of this CNN model are determined by the weight after pruning and the architecture of the CNN. The current population P_t contains the information of the

pruning criteria and pruning ratios. Once the model pruning criteria F^c and pruning ratios $\boldsymbol{r}$ are determined, the FLOPs of the pruned model can be directly obtained by counting the number of the floating point operations. Therefore, the only variables to be optimized are the weights $\mathcal{W}$ in the pruned models:

$$\min_{\mathcal{W}} \sum_{i=1}^{N} \ell(\boldsymbol{x}_i, y_i, F^c, \boldsymbol{r}; \mathcal{W}), \tag{15}$$

which can be directly solved by the stochastic gradient descend. Hence the fitness of an individual in the population is a bigram $(ACC, FLOPs)$. Besides, to overcome the over-fitting problem on the training dataset, a validation dataset which is disjoint from the training dataset is always created to obtain the reported accuracies.

Algorithm 2 provides the pseudo codes of the evaluation process for a population.

3.4. weight inheritance Mechanism

Algorithm 3: Evaluate the given population with weight inheritance

Input: The newly created population O_t, the DNN model m.
Output: The Fitness F_t, the set of the well-trained models M_t, and the auxiliary model m.

$iters = N \times epochs/(generation\_num \times population\_size \times batch\_size)$
for $i = 1$ **to** *population_size* **do**
  $m' \leftarrow$ deep copy the model m;
  Load the corresponding weight from m;
  $F^c, \boldsymbol{r} \leftarrow$ decode the i-th individual O_t^i;
  $\mathcal{S}_r^f \leftarrow$ obtain kept filters via Eqn. 6;
  for $j = 1$ **to** *iters* **do**
    Sample the mini-batch data $\mathcal{D}_{trn}^b$;
    Update the weights $\mathcal{W}$ in m' via its stochastic gradient:
    $\nabla_{\mathcal{W}} \sum_{(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{trn}^b} \ell(\boldsymbol{x}_i, y_i, \mathcal{S}_r^f; \mathcal{W})$;
  end
  $accuracy, flops \leftarrow$ evaluate the model on the $\mathcal{D}_{val}^b$;
  $F_t \leftarrow F_t \cup (accuracy, flops)$;
  $M_t \leftarrow M_t \cup m'$;
  Update the corresponding filters of m **via** m';
end
return F_t, M_t, m;

However, the evaluation of each individual based on algorithm 2 presents a formidable hurdle due to the extensive time commitment it demands. Drawing inspiration from darts [32], we propose a groundbreaking approach aimed at streamlining the calculation process by facilitating the sharing of weights among individuals. At its core, our approach entails treating the original network model as a supernet, while considering all potential pruned network models as sub-networks within this overarching supernet framework.

The crux of our strategy revolves around harnessing the supernet as the foundation for assessing the performance of the sub-networks. When a pruned model is subjected to evaluation, it seamlessly inherits the corresponding weights from the supernet, thereby obviating the need for repetitive training from scratch. This ingenious methodology effectively serves the purpose of acceleration, as the acquired weights from the supernet are judiciously employed to circumvent the redundancy associated with repeated training efforts.

Upon successful training of a specific subnetwork, the attained weights are seamlessly integrated back into the supernet. This iterative process ensures that, as the algorithm progresses, all subnetworks derive the benefit of shared weight inheritance. We have coined this mechanism as the weight inheritance mechanism, which forms the bedrock of our approach. Algorithm 3 delineates the pseudocode that comprehensively outlines the intricate workings of this weight inheritance mechanism.

Table 1
The hyper-parameter settings in the EvoFC.

Model Type	Dataset	Evolutionary Setting	Model Training Setting
VGG-16 Resnet-56 Googlenet Densenet-40	CIFAR-10	pop size : 10 generation: 20 train/val: 19	batch size: 128 training epochs: 300 learning rate: 0.1
VGG-19 Resnet-56 Googlenet Densenet-40	CIFAR-100	pop size : 10 generation: 20 train/val: 19	batch size: 128 training epochs: 300 learning rate: 0.1
Resnet-34	ILSVRC/Places365	pop size : 40 generation : 30 train/val: 19	batch size :128 epoch : 100 learning rate: 0.1
MobilenetV2	ILSVRC/Places365	pop size : 40 generation : 30 train/val: 199	batch size :128 epoch : 300 learning rate: 0.045

By integrating the weight inheritance mechanism into EvoFC, we achieve an unprecedented level of acceleration. The merits of our approach become strikingly apparent when juxtaposed against the conventional practice of training separate models. Under diverse tailoring configurations, our approach confers the remarkable advantage of simultaneously acquiring trained models, with virtually negligible additional computational overhead. Consequently, the arduous task of individually training each network is rendered obsolete, as the weight inheritance mechanism seamlessly facilitates the acquisition of trained models with minimal supplementary calculations.

In summary, our innovative approach, effectively tackles the formidable challenge of time-intensive evaluations when employing algorithm 2 on each individual. By treating the original network as a supernet and harnessing the weight inheritance mechanism, we achieve a substantial acceleration in the computation process. The resultant acceleration in EvoFC is truly remarkable, as it enables the simultaneous acquisition of trained models under diverse customization configurations, without necessitating separate training for each network.

4. Experiments

4.1. Datasets & Baseline Models

We evaluate the proposed EvoFC on the CIFAR-10/100 [23], ILSVRC-2012 [41] and Places365-standard [56] datasets .

CIFAR-10 is a popular image classification dataset that has been widely used to evaluate the performance of weight pruning algorithms. Weight pruning is a technique for reducing the number of parameters in a neural network by removing weights of small magnitude. The CIFAR-10 dataset contains 60,000 color images in 10 categories and is often used as a benchmark for testing the effectiveness of weight pruning methods.

CIFAR-100 expands upon CIFAR-10, a widely utilized benchmark dataset within the realm of computer vision. While CIFAR-10 comprises images categorized into 10 distinct classes, CIFAR-100 elevates this notion by incorporating a diverse spectrum through its 100 classes. Each class within CIFAR-100 represents a distinctive category, encompassing a broad array of objects, animals, and natural scenes. The CIFAR-100 dataset augmented collection of classes imparts greater complexity, fostering the endeavor to develop and refine algorithms, models, and techniques that can effectively discern and classify a wider range of visual concepts.

ImageNet-1K is a widely used image classification dataset that is also used to evaluate the performance of weight pruning algorithms. The large size and complexity of the dataset make it a challenging testbed for pruning methods. Weight pruning has been shown to be an effective technique to reduce the computational cost of neural networks trained on the ImageNet-1K dataset while maintaining high accuracy.

Places365-standard is an all-encompassing dataset for scene recognition that outshines the ImageNet dataset in both scale and diversity. Boasting a staggering count of approximately 1.8 million images, it offers an extensive collection that surpasses the 1.3 million images found in ImageNet. The cornerstone subset of Places365, aptly named Places365-Standard, encompasses a remarkable 365 distinctive scene categories. These categories span a wide

spectrum, capturing a myriad of indoor and outdoor environments, ranging from diverse landscapes and architectural structures to bustling urban settings and serene natural scenes.

The aim of this study is to investigate the effectiveness of single-branch and multi-branch convolutional neural networks for image classification tasks. Convolutional neural networks have achieved remarkable success in image classification, but their large size and complexity pose significant challenges for deployment on resource-constrained devices. Network pruning is a technique that can reduce the size and computational cost of convolutional neural networks by removing unimportant weights. Our goal is to compare the performance of single-branch and multi-branch networks on these datasets, both before and after weight pruning.

For the single-branch network, we adopt the VGG-16 [44] architecture, which is a widely used image classification network. We train the VGG-16 network on the complete dataset, while simultaneously pruning the weights to reduce the number of parameters from the beginning of the training process. We evaluate the performance of EvoFC on the test dataset and compare it to the unpruned network and other pruning methods. In addition, we also perform pruning on the VGG-19 model for reference using the CIFAR-100 dataset.

For multi-branch networks, we train Resnet [10], Densenet [18], Googlenet [46] and MobilenetV2 [43] on the CIFAR, ImageNet and Places365 datasets. On the dual CIFAR datasets, we use the ResNet-56 architecture, which has shown strong performance on this dataset in previous work. Additionally, we train Densenet-40 and Googlenet on the CIFAR datasets. On the ImageNet dataset, we use the Resnet-34 architecture, a variant of Resnet optimized for larger image sizes. MobileNetV2, designed for mobile and edge devices, is also chosen as the baseline model.

4.2. Details of Experimental Settings

The hyper-parameters of the proposed EvoFC contains two parts. The first is the evolutionary setting and the rest of the model training parameters. We list the details of the setting on the Table 1. Five different model types are listed in the table, including a VGG-16/Resnet-56/Densenet-40/Googlenet model trained on the CIFAR dataset, as well as Resnet-34 and MobilenetV2 trained on the ILSVRC and Places365 datasets. These models have diverse evolutionary settings and model training configurations.

In the case of the VGG-16/Resnet-56 model, the evolution settings include a population size of 10, evolution generations of 20. The 'train/val' represents the ratio of the training set to the validation set. The model training settings include a batch size of 128, training epochs of 300, and a learning rate of 0.1.

In the case of the Resnet-34 model, the evolution settings included a population size of 40, evolution generations of 30. The model training settings include a batch size of 128, training epochs of 100, and a learning rate of 0.1.

The experimental details for other models can also be seen from the Table 1.

4.3. Results on CIFAR-10

We first show the Error vs. FLOPs of the evolved pruning configurations on the target CIFAR-10 dataset in Fig. 4. Since EvoFC is a multi-objective evolutionary algorithm, in order to compare with other state-of-the-art methods, we select the largest, medium, and smallest models on the Pareto fronts for the subsequent analysis and comparison.

For the sake of simplicity, we use "EvoFC#L", "EvoFC#M", "EvoFC#S" to refer to the largest, medium, and smallest models, respectively. In order to demonstrate the relative superiority of the performance of our method, we choose the pruning method based on the pretrained model and the pruning method without pretraining. For the pretrained VGG-16 model, we selected L1 [27], ThiNet [37], SSS [19], GAL [31], Hinge [28] and HRank [30]. For methods based on non-pretrained models, we design new structured SNIP [25]-based (SSNIP) methods, as well as structured L1-based pruning (SL1), and random-based structured pruning (SRAND). SRAND can be regarded as the bottom baseline pruning method since it prunes the channels in each layer randomly. By manually setting the global compression rates, we ran the SSNIP, SL1, and SRAND methods at the compression rates of 0.1, 0.2, 0.5, 0.7, and 0.95. We reported the final experimental results. It should be noted that all the above methods are manually set the coefficient to control the final compression rate. The proposed EvoFC does not need to manually specify the pruning rate, but automatically finds the balance between the automatically reassigned compression hyper-parameters and classification accuracy based on a heuristic algorithm.

The results of VGG-16 model on CIFAR-10 is shown in Table 2. Compared with non-pretrained methods that manually set compression parameters, EvoFC has demonstrated significant advantages in automatic parameter setting and effects. For example, EvoFC can reduce 60% of FLOPs and 70% of parameter amounts on the largest model. The intermediate-scale model "EvoFC#M" has better performance than SSNIP, SL1, and SRAND under 0.5, with an average accuracy improvement of 1.5%. Even smaller models, such as "EvoFC#S", can achieve a 90% reduction in

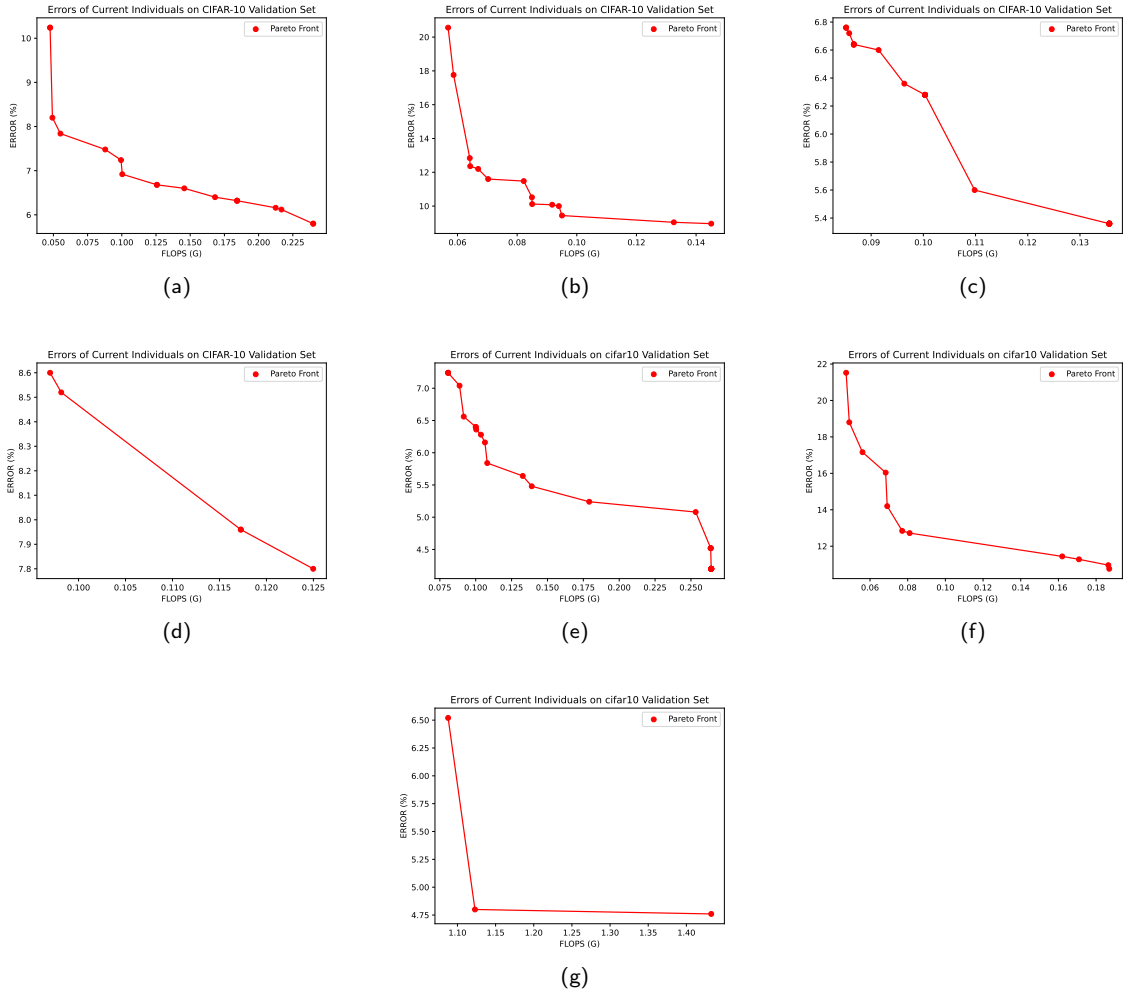


Figure 4: The Pareto curve of Error vs. FLOPs on the CIFAR-10 dataset. a) EvoFC pruned the non-pretrained VGG-16 model on CIFAR-10 without weight inheritance. b) EvoFC pruned the non-pretrained VGG-16 model on CIFAR-10 with weight inheritance. c) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-10 without weight inheritance. d) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-10 with weight inheritance. e) EvoFC pruned the non-pretrained Densenet-40 model on CIFAR-10 without weight inheritance. f) EvoFC pruned the non-pretrained Densenet-40 model on CIFAR-10 with weight inheritance. g) EvoFC pruned the non-pretrained Googlenet model on CIFAR-10 with weight inheritance.

computation and higher accuracy than SSNIP. EvoFC can search for a better pruning rate and the corresponding mixed criterion configuration. This shows that compared with a single criterion strategy, this adaptive criterion method has advantages and can select a more suitable criterion and corresponding pruning rate for different layers of the network. Furthermore, under a larger pruning rate, such as 99.7% pruning rate, the accuracy loss of SSNIP will reach 25%. In this case, the SRAND can achieve better results.

Although EvoFC was originally designed for non-pretrained models, the EvoFC algorithm still shows relatively large advantages compared with traditional pretrained model-based pruning, especially in terms of computational complexity reduction. For example, compared with ThiNet, “EvoFC#L” achieves lower accuracy loss under similar FLOPs. On the other hand, “EvoFC” achieves lower accuracy loss than the GAL algorithm at a parameter drop rate of about 80%, with lower FLOPs and higher classification accuracy.

Additionally, compared with the result of “EvoFC+WS” with the added weight inheritance mechanism, there is a greater performance loss in the accuracy of the final classification, but in exchange for faster search speeds. In some

Table 2
Pruning the VGG-16 model on the CIFAR-10 dataset. The base accuracy of the unpruned model is 93.25%, base FLOPs is 0.63G, number of parameters is 14.99M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
L1	pretrained	manually	34.3	64	-0.15
ThiNet		manually	64.02	63.95	2.49
SSS		manually	41.6	73.8	0.23
GAL		manually	39.6	77.6	1.22
Hinge		manually	39.07	80.05	-0.34
HRank		manually	53.5	82.9	-0.18
SSNIP	non-pretrained	manually (0.1)	18.4	18.7	1.45
SL1					1.04
SRAND					1.24
SSNIP	non-pretrained	manually (0.2)	35.5	36.6	1.65
SL1					1.35
SRAND					1.76
SSNIP	non-pretrained	manually (0.5)	74.8	74.5	2.65
SL1					2.95
SRAND					2.84
SSNIP	non-pretrained	manually (0.7)	92.0	90.5	5.55
SL1					5.40
SRAND					6.21
SSNIP	non-pretrained	manually (0.95)	99.7	99.6	25.95
SL1					24.35
SRAND					24.21
EvoFC#L	non-pretrained	automatically	61.8	73.3	0.35
EvoFC+WS#L			76.9	88.3	3.39
EvoFC#M			76.8	82.9	1.17
EvoFC+WS#M			86.5	93.4	4.54
EvoFC#S			92.4	91.9	5.08
EvoFC+WS#S			90.9	94.9	13.11

cases, “EvoFC+WS” can achieve relatively high results. For example, “EvoFC+WS#M” achieves higher classification accuracy than “EvoFC#S” when the performance of FLOPs and the number of parameters are comparable.

The overall pruning rate of EvoFC is higher than other algorithms. For example, EvoFC can automatically find pruning configurations with close to 90% FLOPs reduction, whether in shared or non-shared weight configurations. Compared with manually set pruning methods based on pretrained models, the reduction in FLOPs is more obvious. In particular, the reduction in FLOPs of the pruning method based on pretrained models is generally much lower than the reduction in the number of parameters. For example, in the Hinge method, although 80% of the model parameters are reduced, the relative FLOPs are only reduced by about 40%. However, the proposed EvoFC algorithm has a relatively balanced reduction in the number of parameters and FLOPs, maintaining a ratio of 1:1.

For the Resnet-56 model, we chose L1 [27], CP [14], NISP [52], DCP [58], IR [47], C-SGD [6], GBN [51], HRANK [30], PFIF [38] as peer competitors. CP does not need to manually specify the pruning rate, but is adaptively obtained by the algorithm itself. Unlike some other pruning methods, such as those requiring a pretrained model as a hypothesis, PFIF can achieve network pruning with high accuracy without such a requirement. However, it is worth noting that the granularity of pruning achieved with PFIF falls somewhere between structured and unstructured pruning.

The experimental results are shown in Table 3. At a compression rate of about 50%, “EvoFC#L” gains a certain improvement in the classification accuracy compared with SSNIP. At a FLOPs reduction rate of about 65%, “EvoFC#S” has also been compared with SSNIP and SL1 that gains better classification performance. It is worth noting that the performance of “EvoFC#M” is relatively lower than that of “EvoFC#S” despite having a larger amount of computation. We think this may be due to the overfitting problem of the current individuals on the validation set, resulting in the performance of the M-type. Although the accuracy of the validation set is higher than that of the S-type, it is not as good as the small model on the test set. In addition, when the compression rate reaches about 90%, the classification performance of Resnet-56 is greatly reduced. That is, the accuracy drop of SSNIP at 0.95 is 4.6% larger than that at 0.9.

Table 3
Pruning results for the Resnet-56 model on the CIFAR-10 dataset. The accuracy of the unpruned baseline model is 93.1%, the base FLOPs is 0.25G, and the number of parameters is 0.85M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
L1	pretrained	manually	27.6	13.7	-0.02
CP	pretrained	automatically	50	-	1.00
NISP	non-pretrained/pretrained	manually	43.6	42.6	0.03
DCP	pretrained	manually	47.1	70.3	-0.01
IR	pretrained	manually	67.7	-	0.4
C-SGD	pretrained	manually	60.8	-	-0.23
GBN	pretrained	automatically	70.3	66.7	0.03
HRank	pretrained	manually	74.1	68.1	2.38
PFIF	non-pretrained	manually	75.6	77.7	0.12
SSNIP	non-pretrained	manually (0.1)	8.3	9.4	-0.2
SL1					-0.1
SRAND					0.2
SSNIP	non-pretrained	manually (0.5)	49.7	49.4	1.3
SL1					0.8
SRAND					0.7
SSNIP	non-pretrained	manually (0.6)	57.9	58.8	1.2
SL1					2.1
SRAND					1.3
SSNIP	non-pretrained	manually (0.7)	68.3	68.2	1.5
SL1					2.9
SRAND					1.7
SSNIP	non-pretrained	manually (0.75)	74.5	74.1	2.3
SL1					3.6
SRAND					2.3
SSNIP	non-pretrained	manually (0.8)	77.0	78.8	3.6
SL1					4.3
SRAND					2.8
SSNIP	non-pretrained	manually (0.9)	87.4	88.2	4.9
SL1					8.9
SRAND					5.1
SSNIP	non-pretrained	manually (0.95)	93.1	92.9	9.5
SL1					13.5
SRAND					8.7
EvoFC#L	non-pretrained	automatically	46.4	43.0	0.79
EvoFC+WS#L			50.6	52.6	3.42
EvoFC#M			60.3	63.5	1.80
EvoFC+WS#M			53.6	53.0	3.27
EvoFC#S			66.3	65.5	1.42
EvoFC+WS#S			61.7	62.2	4.08

Compared with the pretrained methods, the overall EvoFC is not obvious on Resnet-56. Take the NISP and CP as examples, the proposed EvoFC can automatically search for pruning configurations with lower computational load, but the corresponding accuracy is not as good. In addition, under the condition of the maximum compression rate, the model FLOPs obtained by EvoFC is also higher than that of GBN, HRank, and PFIF. PFIF is relatively one of the most SOTA algorithms at present, so its effect is better than EvoFC, although it does not require the pruned model to be pretrained. It is worth noting that PFIF is a hybrid granularity pruning method.

After adding the weight inheritance mechanism, although better search speed is obtained, the accuracy of classification is also affected. For example, in the case of the smallest models, even though "EvoFC+WS#S" has a larger computational complexity than "EvoFC#S", the accuracy is about 2.5% lower. In addition, it can be seen that the compression rate of EvoFC is between 40% and 70%. We think this is affected by the population size of the EA algorithm.

For the Densenet-40 model, we chose GAL [31], Hinge [28], C-SGD [6], HRank [30] and Slimming [34] as the comparative methods. The experimental results about the "EvoFC" and "EvoFC+WS" are reported in Table 4. From the table, it is evident that the proposed EvoFC achieves a considerable accuracy loss compared to traditional well-trained pruning methods while significantly reducing computational consumption. For instance, the computational FLOPs of

Table 4
Pruning results for the Densenet-40 model on the CIFAR-10 dataset. The accuracy of the unpruned baseline model is 94.13%, the base FLOPs is 0.57G, and the number of parameters is 1.06M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
GAL	pretrained	manually	71.4	75.0	1.58
Hinge	pretrained	manually	44.4	27.5	0.07
C-SGD	pretrained	manually	60.1	-	-0.75
HRank	pretrained	manually	61.0	53.8	1.13
Slimming	pretrained	manually	55.0	65.2	-0.5
EvoFC#L	non-pretrained	automatically	53.9	53.3	0.9
EvoFC+WS#L			67.4	75.3	6.9
EvoFC#M			81.2	79.6	1.99
EvoFC+WS#M			86.5	83.1	9.17
EvoFC#S			85.9	84.4	2.26
EvoFC+WS#S			91.7	91.7	17.5

Table 5
Pruning results for the Googlenet model on the CIFAR-10 dataset. The accuracy of the unpruned baseline model is 94.51%, the base FLOPs is 3.07G, and the number of the parameters is 6.17M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
GAL	pretrained	manually	38.2	49.3	1.12
L1	pretrained	manually	32.9	41.8	0.51
HRank	pretrained	manually	54.9	55.4	0.52
EvoFC+WS#L	non-pretrained	automatically	53.3	44.1	0.23
EvoFC+WS#M			63.4	52.2	0.97
EvoFC+WS#S			64.5	52.5	1.41

the "EvoFC#WS" outperforms all selected pretrained-based methods. However, the accuracy of EvoFC experiences a significant drop when employing the weight inheritance mechanism.

For the Googlenet model, we chose GAL [31], L1 [27] and HRank [30] as the peer competitors. Due to the large size of the Googlenet model, we applied the proposed weight inheritance mechanism for pruning and present the results in Table 5. The results for Googlenet demonstrate the superiority of our algorithm over the selected competitors, even when pruning from a random initialization. Specifically, EvoFC achieves higher accuracy than GAL and L1, while reducing FLOPs by 53.3%. Additionally, compared to HRank, EvoFC exhibits a smaller accuracy drop while maintaining similar FLOPs.

Overall, the proposed EvoFC is an automated network pruning method with good optimization and pruning effects for non-pretrained models on the CIFAR-10 dataset. It is able to effectively reduce model computation and parameters while maintaining model performance. Compared with the manually set non-pretrained algorithm, the superiority of EvoFC is more obvious. This is because during the training process, EvoFC can automatically search for the optimal hyper-parameter, which better balances model complexity, parameters and performance, and does not require artificial selection of pruning rates or the corresponding hyperparameters. When conducting experiments with conventional pretrained techniques, EvoFC consistently demonstrates the superiority of automated search. It conveniently and flexibly acquires diverse weight files across various configurations, enabling the direct acquisition of models with different compression ratios without the need for subsequent training. This characteristic makes it highly suitable for swift deployment on a wide range of devices.

4.4. Results on CIFAR-100

Fig. 5 shows the Error vs. FLOPs Curve of the EvoFC on the CIFAR-100 dataset. The quantitative results on the CIFAR-100 dataset are documented in Tables 6, 7, 8 and 9.

Table 6 presents the pruning results for the VGG19 model. For comparison, we selected Slimming [34] as the only peer competitor. Slimming achieves a lower reduction in FLOPs compared to its significant parameter reduction. In contrast, the proposed EvoFC achieves higher FLOPs reduction while maintaining similar parameter reductions. However, EvoFC does suffer from a significant accuracy loss when compared to pretrained-based method. By enabling

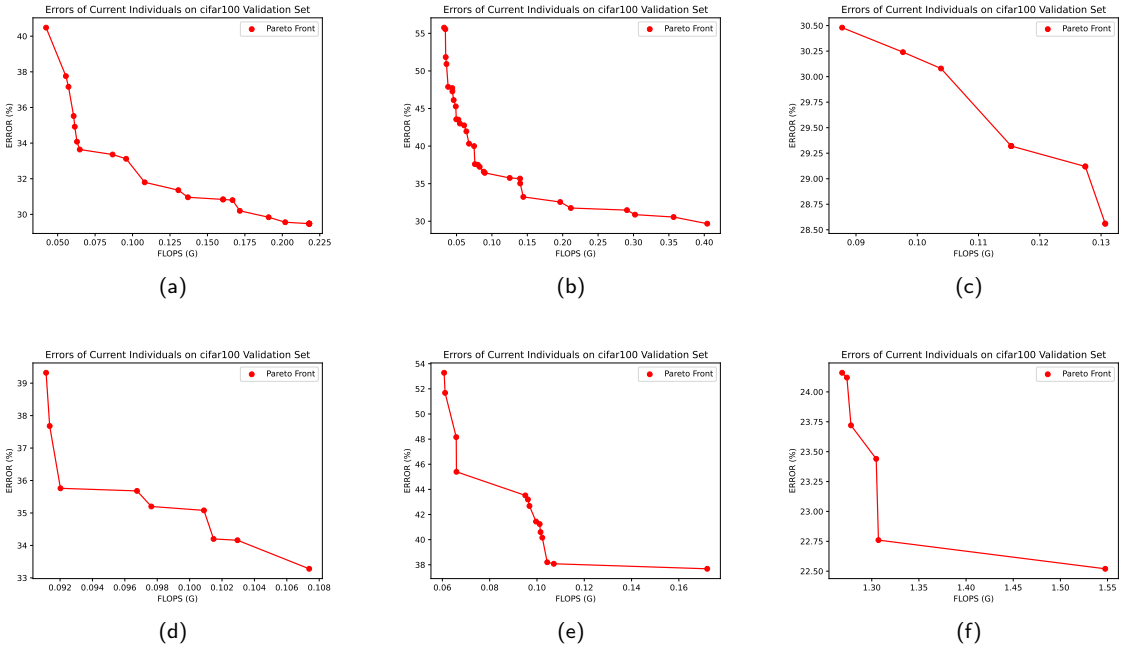


Figure 5: The Pareto curve of Error vs. FLOPs on the CIFAR-100 dataset. a) EvoFC pruned the non-pretrained VGG-19 model on CIFAR-100 without weight inheritance. b) EvoFC pruned the non-pretrained VGG-19 model on CIFAR-100 with weight inheritance. c) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-100 without weight inheritance. d) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-100 with weight inheritance. e) EvoFC pruned the non-pretrained Densenet-40 model on CIFAR-100 with weight inheritance. f) EvoFC pruned the non-pretrained Googlenet model on CIFAR-100 with weight inheritance.

Table 6
Pruning results for the VGG-19 model on the CIFAR-100 dataset. The accuracy of the unpruned baseline model is 71.81%, the base FLOPs is 0.80G, and the number of parameters is 20.34M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
Slimming	pretrained	manually	37.1	75.1	-0.22
EvoFC#L	non-pretrained	automatically	72.7	85.7	2.99
EvoFC+WS#L			49.3	58.9	1.72
EvoFC#M			86.5	91.3	5.15
EvoFC+WS#M			90.6	90.0	10.26
EvoFC#S			94.7	95.0	12.74
EvoFC+WS#S			95.9	96.5	27.28

the WS mechanism, EvoFC adopts a more aggressive FLOPs reduction strategy, but this results in a substantial drop in accuracy.

The pruned model for the Resnet-56 model is presented in Table 7. FPGM [13] and SFP [12] were chosen as peer competitors representing manually-based pruning methods. Without the weight inheritance strategy, EvoFC outperforms the pretrained SFP by 0.56% with a higher acceleration rate. When compared to the pretrained FPGM, its result is slightly better than SFP. Furthermore, in comparison to the results on CIFAR-10, EvoFC exhibits similar compression capability but with a significant accuracy loss.

For the Densenet-40 and Googlenet models, given the considerable training resources required, we opted to train them solely with the weight inheritance technique for rapid verification. The corresponding results can be found in Tables 8 and 9. Notably, EvoFC demonstrates superior performance for the Googlenet when compared to the Densenet-40. Surprisingly, enabling the weight inheritance strategy for the Densenet-40 yields higher accuracy drops than anticipated, akin to the results observed on the CIFAR-10 dataset. We intend to explore this phenomenon in greater

Table 7
Pruning results for the Resnet-56 model on the CIFAR-100 dataset. The accuracy of the unpruned baseline model is 70.83%, the base FLOPs is 0.25G, and the number of parameters is 0.86M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
FPGM	pretrained	manually	52.6	-	1.75
SFP	non-pretrained	manually	52.6	-	2.61
EvoFC#L	non-pretrained	automatically	48.3	54.1	1.45
EvoFC+WS#L			57.5	50.4	5.23
EvoFC#M			54.3	51.4	2.05
EvoFC+WS#M			61.3	58.5	5.86
EvoFC#S			65.3	59.1	2.56
EvoFC+WS#S			63.9	63.8	7.59

Table 8
Pruning results for the Densenet-40 model on the CIFAR-100 dataset. The accuracy of the unpruned baseline model is 73.11%, the base FLOPs is 0.57G, and the number of parameters is 1.1M.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
Slimming	pretrained	manually	47.1	54.6	0.36
EvoFC+WS#L	non-pretrained	automatically	70.0	62.6	11.01
EvoFC+WS#M			82.6	69.0	14.14
EvoFC+WS#S			89.4	89.8	25.71

Table 9
Pruning results for the Googlenet model on the CIFAR-100 dataset. The accuracy of the unpruned baseline model is 78.02%, the base FLOPs is 3.07G, and the number of parameters is 6.26M.

Pruned Method	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
EvoFC+WS#L	49.5	36.7	2.14
EvoFC+WS#M	57.5	45.9	2.30
EvoFC+WS#S	58.6	50.4	2.67

Table 10
EvoFC on the ImageNet-1k dataset for the Resnet-34 model. The accuracy and FLOPs of the base model are 73.31%, 7.34G, separately.

Pruned Method	Input Model	Compression Setting Assignment	FLOPS Drop (%)	PARAM Drop (%)	Top-1 ACC Drop (%)
L1	pretrained	manually	7.5	7.2	0.75
FPGM	pretrained	manually	41.1	-	1.29
SFP	non-pretrained	manually	41.1	-	2.09
NISP	non-pretrained/pretrained	manually	43.8	43.7	0.92
Edropout	non-pretrained	automatically	-	55.8	1.49
EvoFC+WS#L	non-pretrained	automatically	52.9	39.2	8.16
EvoFC+WS#M			71.3	54.3	10.21
EvoFC+WS#S			79.6	77.4	14.53

depth in future research endeavors. Despite the substantial reduction in computational burden, achieving nearly 90% FLOPs reduction for the Densenet-40 model, the Googlenet model only attains a reduction of 58.6%. However, it is worth noting that the accuracy drop for the Googlenet is significantly less pronounced compared to that of the Densenet-40.

Overall, CIFAR-100 presents a more challenging dataset for pruning methods compared to CIFAR-10, as evidenced by the average classification accuracy of the pruned models. However, the proposed EvoFC demonstrates its ability to compress the target network architecture with a higher reduction in FLOPs, while also demonstrating that accuracy loss is not inevitable.

4.5. Results on ILSVRC-2012

We also trained EvoFC on the ImageNet-1k dataset, because the cost of training on ImageNet using the non-shared mechanism is too high, for example, for evaluating 400 individuals, it may take more than 20 months in the case of two

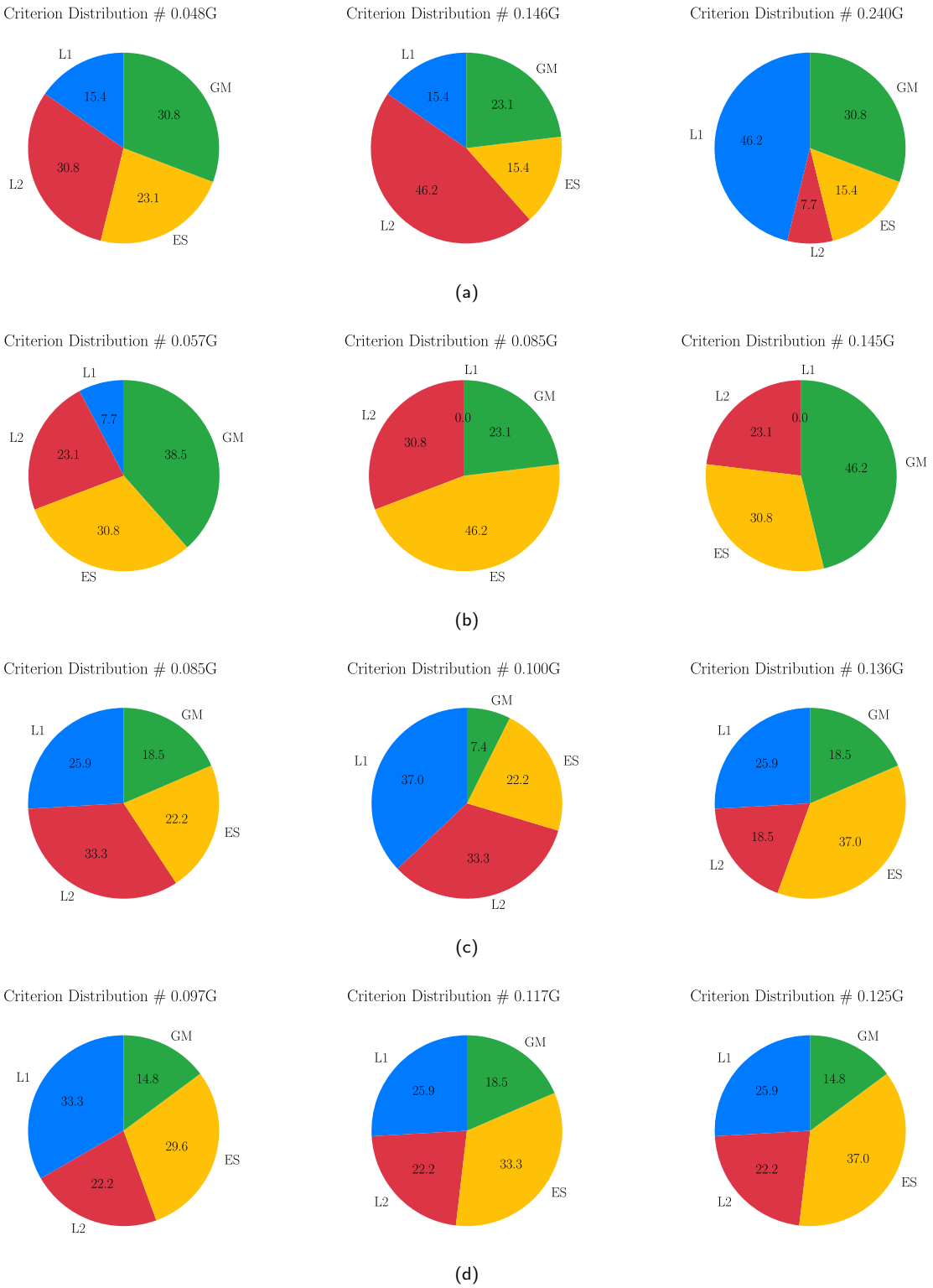


Figure 6: Visualize distribution plots of the different criteria for the smallest, medium, and largest models. a) EvoFC pruned the non-pretrained VGG-16 model on CIFAR-10 without weight inheritance. b) EvoFC pruned the non-pretrained VGG-16 model on CIFAR-10 with weight inheritance. c) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-10 without weight inheritance. d) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-10 with weight inheritance.

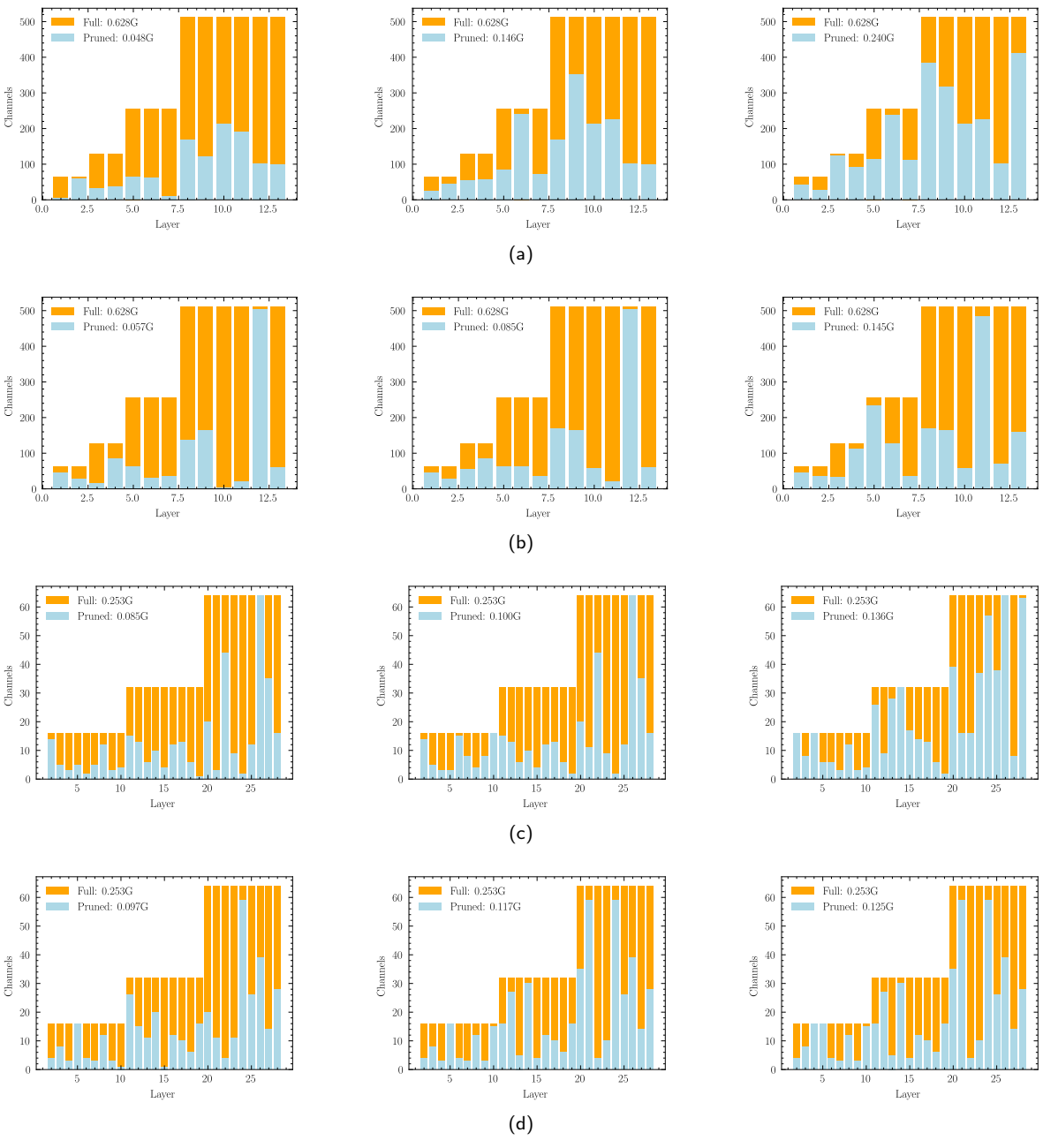
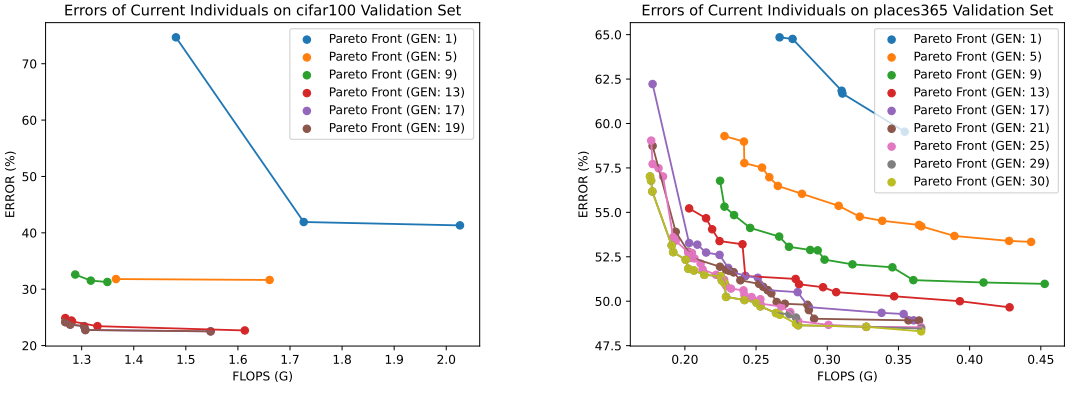


Figure 7: The number of channels vs. layers for the smallest, medium, and largest models before and after pruning, to analyze the impact of pruning on network architecture. a) EvoFC pruned the non-pretrained VGG-16 model on CIFAR-10 without weight inheritance. b) EvoFC pruned the non-pretrained VGG-16 model on CIFAR-10 with weight inheritance. c) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-10 without weight inheritance. d) EvoFC pruned the non-pretrained Resnet-56 model on CIFAR-10 with weight inheritance.

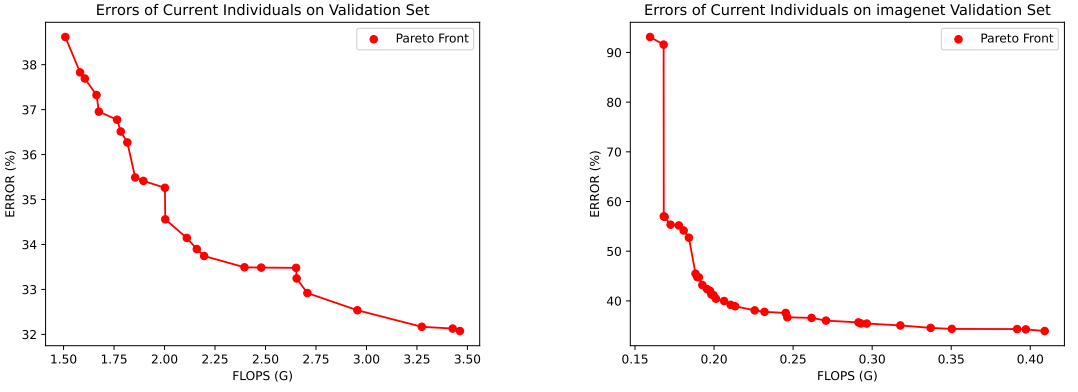
NVIDIA RTX3090s. Therefore, this training resource is unacceptable. We can only train an EvoFC based on weight inheritance for testing and analysis. The setting of the EvoFC for the ImageNet-1k dataset is shown in Table 1.

The resulting Pareto fronts for the Resnet-34 and MobilenetV2 are shown in Fig. 9. For the Resnet-34, we selected methods L1 [27], FPGM [13], SFP [12], NISP [52] and Edropout [42] for comparison. It is worth noting that although EvoFC has a large gap with the selected method in terms of final accuracy, EvoFC's reduction in FLOPs is significantly



(a) (b)

Figure 8: a) Pareto curves illustrating the pruning of Googlenet on the CIFAR-100 dataset across generations. b) Pareto curves depicting the pruning of MobilenetV2 on the Places365 dataset across generations.



(a) (b)

Figure 9: a) The relationship between the Error and FLOPs on the ImageNet-1k dataset for the Resnet-34 model, represented by the Pareto curve. b) The relationship between the Error and FLOPs on the ImageNet-1k dataset for the MobilenetV2 model.

Table 11
Pruning results for the MobilenetV2 model on the ImageNet-1k dataset. The accuracy of the unpruned baseline model is 71.878% inherited from the pytorch official model. The base FLOPs of the MobilenetV2 model is 0.62G.

Pruned Method	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
EvoFC+WS#L	34.1	10.6	10.25
EvoFC+WS#M	66.8	31.6	15.90
EvoFC+WS#S	74.3	44.5	65.37

higher than that of selected SOTA pruning methods. In the process of automatically searching the pruning rate, the FLOPs of the model obtained by EvoFC can be reduced by approximately 80%. It is particularly important to note that these conventional pruning methods usually require retraining when changing the compression configuration, while EvoFC does not require such a process of re-configuration or retraining. That is, the model represented by any point on the frontier in Fig. 9 can be directly deployed on edge devices where computing resources are not sufficient, such

Table 12
Pruning results for the Resnet-34 model on the Places365 dataset. The accuracy of the unpruned baseline model is 54.81%. The base FLOPs of the Resnet-34 model is 7.34G, and the base number of parameters 21.47M.

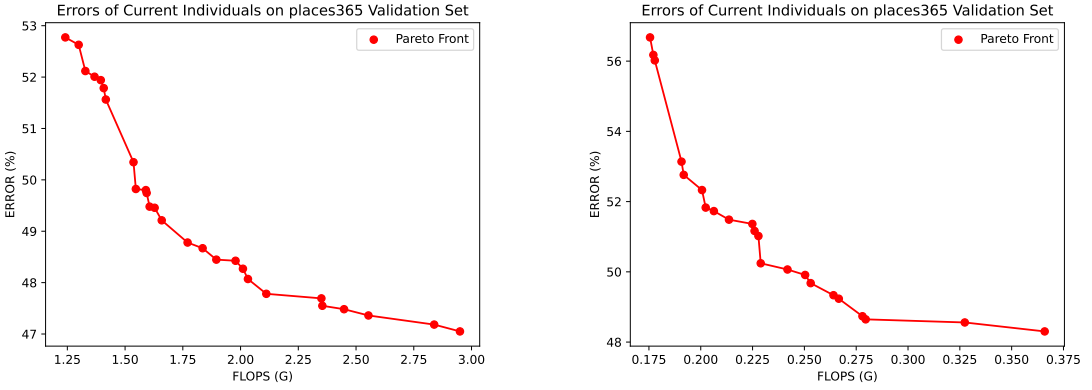
Pruned Method	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
EvoFC+WS#L	59.8	27.8	2.39
EvoFC+WS#M	77.4	62.2	4.86
EvoFC+WS#S	83.1	78.0	8.09

as mobile phones. Edropout is another method that does not require well-trained weights for the target network and demonstrates high accuracy compared to our proposed EvoFC method. Even with approximately 50% reduction in FLOPs, Edropout achieves superior accuracy when compared to our method.

The results for MobilenetV2 are presented in Table 11. While EvoFC achieves higher results, the accuracy drop exhibited for the dataset indicates a significant compression rate, surpassing our initial expectations. Notably, when the FLOPs reduction reaches 74.3%, the corresponding accuracy drop amounts to 65%. We posit that this outcome may be attributed to the inherent compactness of MobilenetV2 on the ILSVRC dataset.

It is important to highlight that while the Resnet-34/MobilenetV2 model we employed for comparison benefits from pre-trained weights on ImageNet, our approach does not rely on the assumption of pre-trained network weights. EvoFC enables model compression without the need for pre-training, making it applicable to any custom network with a fundamental convolutional architecture. This allows for direct deployment of models tailored to different computational complexities. Should users desire enhanced accuracy, they can further refine and optimize the pruned and trained model through fine-tuning, thereby achieving superior performance on the target dataset.

4.6. Results on Places365



(a) (b)

Figure 10: a) The relationship between the Error and FLOPs on the Places365-standard dataset for the Resnet-34 model, represented by the Pareto curve. b) The relationship between the Error and FLOPs on the Places365-standard dataset for the MobilenetV2 model.

On the Places365-standard dataset, we conducted pruning experiments on the original Resnet-34 and MobilenetV2 models. The Pareto front curves illustrating their performance are depicted in Fig. 10, and the corresponding results are reported in Tables 12 and 13. Given the scarcity of papers reporting network pruning performance specifically on the Places365 dataset, we chose to compare the pruned models solely with the original full networks. Remarkably, on the Places365 dataset, the Resnet-34 model exhibits the ability to achieve up to an 83% reduction in FLOPs, while the corresponding accuracy drop is not as pronounced as observed on the ImageNet-1k dataset.

Table 13
Pruning results for the MobilenetV2 model on the Places365 dataset. The accuracy of the unpruned baseline model is 53.9%. The base FLOPs of the MobilenetV2 model is 0.62G, and the number of parameters is 3.5M.

Pruned Method	FLOPS Drop (%)	PARAM Drop (%)	ACC Drop (%)
EvoFC+WS#L	40.9	18.9	2.84
EvoFC+WS#M	63.2	37.5	4.96
EvoFC+WS#S	71.7	55.5	10.72

Moreover, the pruning results for MobilenetV2 on this dataset are notably more favorable. The largest accuracy drop observed is a mere 10.72%, a considerable improvement when compared to the results obtained on the ImageNet dataset.

These results on the Places365 dataset effectively demonstrate the generalizability of the proposed EvoFC approach. It showcases its capacity to compress models with larger compression ratios while yielding substantial improvements in accuracy drop.

4.7. Visualization of Pruning

In order to intuitively illustrate the effect of EvoFC on the searched criteria, we visualized the corresponding criterion configurations of the searched models and analyzed them. For simplicity, we still only analyze the largest, medium, and smallest models on the Pareto front.

We first counted the distribution of the criterion of each pruning model and displayed it in Fig. 6. For the VGG-16 model, "EvoFC#S" and "EvoFC#M" account for the largest criterion are both L2, which are 30.8% and 46.2%. The second place is GM criterion, which accounted for 30.8% and 23.1% respectively. For the largest "EvoFC#L", the largest proportion is the L1 criterion, reaching 46.2%. However, when the weight inheritance mechanism is applied, the L1 criterion is suppressed and becomes the smallest criterion. Especially On "EvoFC+WS#M" and "EvoFC+WS#L", the search algorithm completely discards the L1 criterion. i.e., its proportion is 0%. In addition, the other three criteria are relatively balanced. Specifically, the GM criterion has a large proportion on "EvoFC+WS#S" and "EvoFC+WS#L"; while the ES criterion has the largest proportion on "EvoFC+WS#M", eached 46.2%. Meanwhile, for the two largest models, the GM criterion occupies the largest proportion. In general, L1-norm is not so important for VGG-16 model from the figure, although it accounts for 46.2% in "EvoFC#L". This also shows that weight sharing has a significant impact on the choice of criterion.

For the pruning of Resnet-56 model, the GM criterion is always the lowest proportion. For example, in "EvoFC#M", the GM criterion only accounts for 7.4%, which shows that in the Resnet-56 model. This indicates that the sensitivity of the Resnet-56 model to the GM criterion is not high, which is different from the VGG-16 model. In addition, L2, L1 and ES are the most used criteria in "EvoFC#S", "EvoFC#M", "EvoFC#L", respectively. With the introduction of the weight inheritance mechanism, L1, ES has become the most popular criterion.

The proportion of different criteria and the size of the model is not a simple linear relationship. E.g., for the Resnet-56 model, as the model size increases, the proportion of the L1 criterion will first increase and then decrease. While for the ES criterion, the proportions of "EvoFC" and "EvoFC+WS" gradually increase with the size of the model.

Additionally, we show the number of channels reserved for each convolutional layer/block of VGG-16 and Resnet-56 on CIFAR-10 in Fig. 7. First take the VGG-16 model as an example. On the largest model obtained by EvoFC, the layers with the largest pruning rates come from deeper layer. With the reduction of network complexity, the first few layers of the VGG-16 network also began to be pruned forcibly. For the network in the middle part, more channels are reserved. For EvoFC with the applied weight inheritance mechanism, the network pruning rates of the last layer of the three models are all relatively high. Second, around at layer 10, all three models exhibit very high rates of channel removal.

For the Resnet-56 model, it can be clearly found that the proposed EvoFC is biased towards pruning the layer in the middle of the network. In addition, from the perspective of different stages, the overall trend of the number of channels reserved at the network layer in stage-1 and stage-2 is getting lower and lower. But in stage-3, there is a low pruning rate in the next few layers. That is, at the network layer in the penultimate block, there is almost no pruning operation in the models of the three scales, which means that this layer of network might have a greater impact on the performance of the network, so the evolutionary algorithm for this layer of pruning is relatively conservative, and it less inclined to do more pruning on this layer. In the case of EvoFC with the weight inheritance mechanism, the models of the three scales

Table 14
Ablation study of the Weight Initialization Mechanism in terms of training time cost.

Model	Dataset	Evaluation Mechanism	Population Size	Generation	Search Time Cost (GPU Days)
VGG-16	CIFAR-10	Vanilla	10	20	10.21
VGG-16	CIFAR-10	Weight Inheritance	10	20	0.05
VGG19	CIFAR-100	Vanilla	10	20	10.64
VGG19	CIFAR-100	Weight Inheritance	10	20	0.67
Resnet-56	CIFAR-10	Vanilla	10	20	22.88
Resnet-56	CIFAR-10	Weight Inheritance	10	20	0.12
Resnet-56	CIFAR-100	Vanilla	10	20	25.84
Resnet-56	CIFAR-100	Weight Inheritance	10	20	0.17
Densenet-40	CIFAR-10	Vanilla	10	20	43.40
Densenet-40	CIFAR-10	Weight Inheritance	10	20	0.09
Densenet-40	CIFAR-100	Vanilla	10	20	-
Densenet-40	CIFAR-100	Weight Inheritance	10	20	0.08
Googlenet	CIFAR-10	Vanilla	10	20	-
Googlenet	CIFAR-10	Weight Inheritance	10	20	0.23
Googlenet	CIFAR-100	Vanilla	10	20	-
Googlenet	CIFAR-100	Weight Inheritance	10	20	0.23
Resnet-34	ILSVRC-2012	Vanilla	40	30	-
Resnet-34	ILSVRC-2012	Weight Inheritance	40	30	6.98
Resnet-34	Places365	Vanilla	40	30	-
Resnet-34	Places365	Weight Inheritance	40	30	3.33
MobilenetV2	ILSVRC-2012	Vanilla	40	30	-
MobilenetV2	ILSVRC-2012	Weight Inheritance	40	30	5.94
MobilenetV2	Places365	Vanilla	40	30	-
MobilenetV2	Places365	Weight Inheritance	40	30	4.99

are relatively similar in term of the the overall trend of channel retention. For the two models "EvoFC+WS#L" and "EvoFC+WS#M", the channel retention rates of stage-2 and stage-3 are almost the same. The main difference comes from the stage-1, which is due to the difference in computational complexity caused by "EvoFC+WS#L" keeping more intermediate channels in the stage-1. The results of the "EvoFC+WS#S" and "EvoFC+WS#M" are similar in stage-1, except for the last block. In addition, the two models have certain differences in the shallow layers in stage-2 and stage-3.

In addition, to provide insights into the behavior of the EvoFC during the pruning process, we include the visualization of the Pareto front for each generation. This allows us to demonstrate how the algorithm strives to approximate the Pareto optimality in terms of both computational complexity and classification accuracy. For the purpose of simplicity, we have chosen to present figures specifically for Googlenet on the CIFAR-100 dataset and MobilenetV2 on the Places365 dataset. These figures can be found in Figures 8.

Overall, EvoFC can automatically assign different pruning ratios to different layers of the non-pretrained network, thereby reducing the computational complexity of the model.

4.8. Ablation Study

We conducted ablation experiments on weight inheritance to assess its impact on training efficiency. To measure the search time cost, we adopted the GPU days metric introduced by Sun [45]. Table 14 presents the search time cost for different models on various datasets, comparing the performance of Vanilla and Weight Inheritance-based evaluation mechanisms in terms of search time cost.

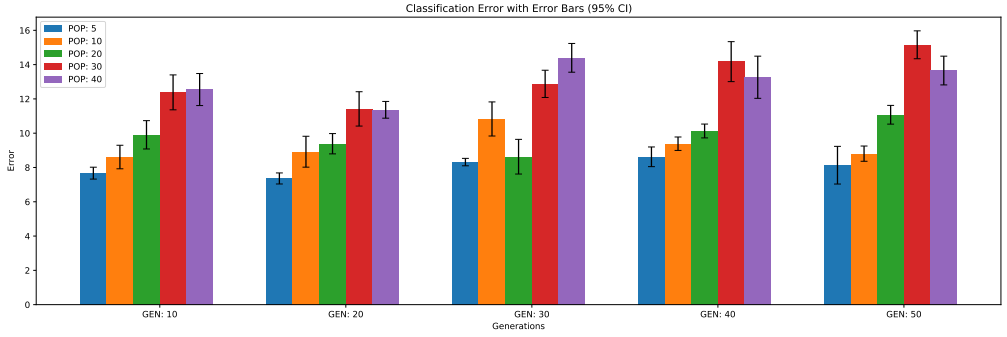
Regarding the evaluation of the VGG-16 model on the CIFAR-10 dataset, the search time cost using the conventional evaluation amounted to 10.21 GPU days, whereas employing the weight inheritance mechanism reduced the search time cost to a mere 0.05 GPU days.

Similarly, for the evaluation of the Resnet-56 model on the CIFAR-10 dataset, the search time cost was 22.88 GPU days when adopting the traditional evaluation mechanism, whereas utilizing the weight inheritance mechanism

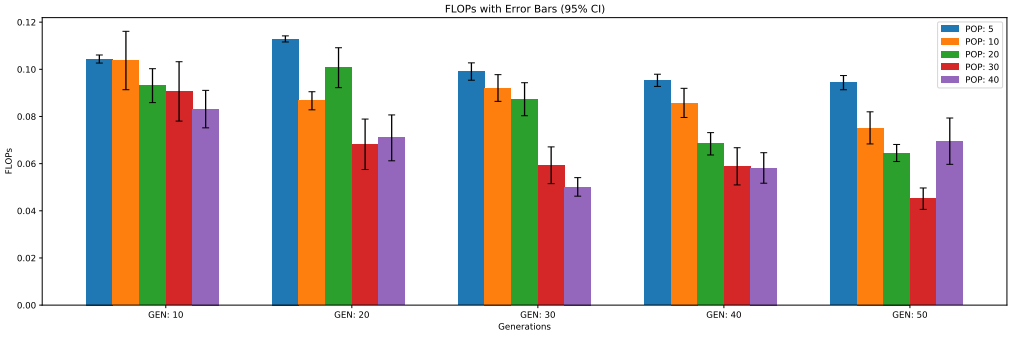
resulted in a significantly lower search time cost of 0.12 GPU days. The training time required for the two strategies on the Densenet-40 demonstrates significantly larger ratios, with weight inheritance achieving a remarkable acceleration of approximately 400 times.

For the Resnet-34 model evaluated on the ILSVRC-2012 dataset, a direct comparison was unfeasible due to the extensive computational resources required. However, it was theoretically estimated that the Vanilla evaluation mechanism would exceed ten months of search time. In contrast, the weight inheritance mechanism demonstrated a considerably reduced search time cost of 6.98 GPU days. These observations highlight the effectiveness of the weight inheritance mechanism in substantially reducing search time costs, even in the context of large-scale datasets. In a similar manner, for the MobilenetV2 and Googlenet, we exclusively provide the search time cost while utilizing weight inheritance, as the associated training cost proves to be excessively burdensome.

4.9. Sensitivity Analysis



(a) Error with CI.

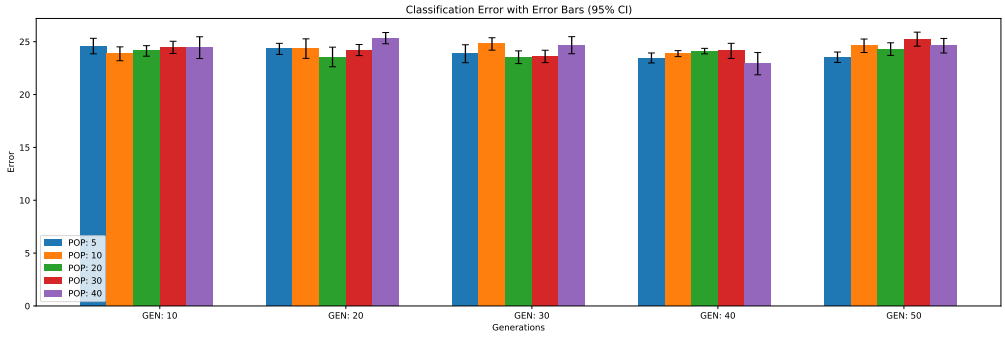


(b) FLOPs with CI.

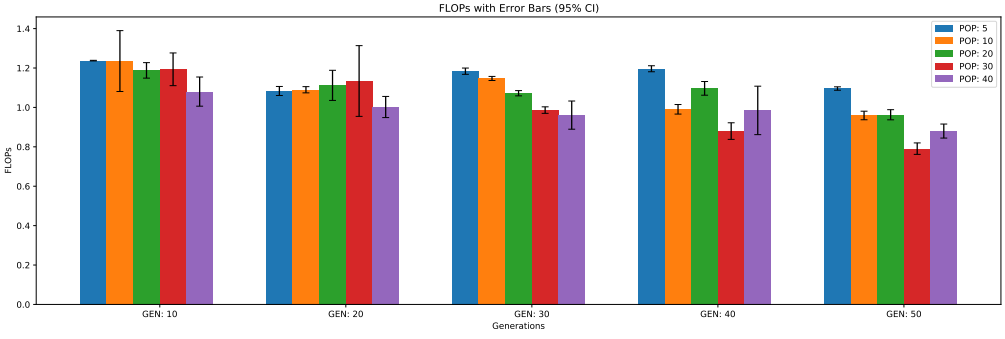
Figure 11: Sensitivity analysis for population sizes and generations for Resnet-56 on the CIFAR-10 dataset.

The number of generations and population size play pivotal roles as hyper-parameters in our proposed EvoFC, particularly with the WI mechanism enabled. Population size affects the initial setup of GA. A larger population may have an advantage over smaller ones, as it can explore a wider range of promising model pruning configurations early on thanks to its expanded sampling domain. Boosting the number of generations can likewise enhance the end results. Nonetheless, given the WI mechanism, the quantity of samples contributing to select individuals reduces with bigger pop sizes and more generations, which in turn could affect the final results. To examine this effect, we'll carry out a sensitivity analysis in this part, unraveling how these two factors impact the network pruning strategy.

For the sake of simplicity, we conduct sensitivity analyses on two models, i.e., Resnet-56 and Googlenet. Specifically, Resnet-56 is experimented on the CIFAR-10 dataset, while Googlenet is experimented on the CIFAR-100



(a) Error with CI.



(b) FLOPs with CI.

Figure 12: Sensitivity analysis for population sizes and generations for Googlenet on the CIFAR-100 dataset.

dataset. We apply the 95% Confidence Interval (CI) to the dual metrics of the final individuals in the Pareto optimal set. For each hyper-parameter, we compute the mean and margin of error then plot them in the Figures 11 and 12.

For the Resnet-56 model on the CIFAR-10 dataset, we notice that as the population grows, the errors in the validation set increase. This makes sense because each individual receives fewer training iterations. Additionally, the average computational complexity decreases over generations. This indicates that as the evolutionary process progresses, the search algorithm uncovers a more condensed architecture.

In the case of Googlenet, its error variation on the validation set appears to be relatively stable compared to Resnet-56. This implies that, in terms of classification errors, the impact of population size and evolution generations on the EvoFC pruning of Googlenet is relatively minor. Conversely, in the confidence interval plot of FLOPs, similar conclusions can be drawn as those for the Resnet-56 model. That is, larger numbers of generations and population sizes lead to a greater capacity for the algorithm to explore more compact network architectures.

4.10. Batch Size of Criteria

To demonstrate the impact of batch size on the proposed EvoFC, we selected batch sizes for the criterion from the set $[8, 16, 32, 64, 128, 256, 512, 1024]$. We compared the obtained Pareto fronts for each batch size, and the results are presented in Figure 13. As depicted in the figure, the batch size exhibits diverse influences, aligning with our expectations before conducting the experiments. The Pareto curve varies with the batch size, indicating that there is no optimal setting evident from the figure. Each batch size offers its own advantages in approximating the optimal Pareto front. Notably, even the largest batch size of 1024 does not demonstrate significant superiority compared to the smaller batch sizes. This finding contradicts our initial expectation of more dominant results with larger batch sizes for filter decision-making prior to conducting the experiments.

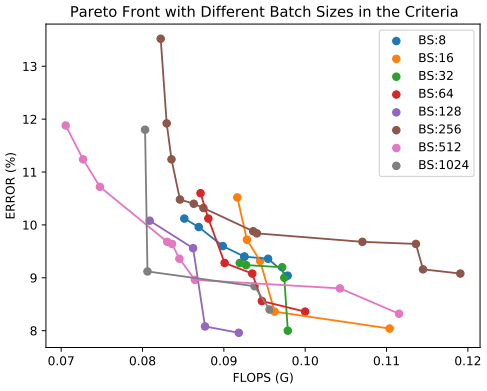


Figure 13: The correlation between the Pareto curve and batch size in criterion calculation, observed with the Resnet-56 model on the CIFAR-10 dataset.

5. Conclusion

In this study, we introduce EvoFC, a novel network pruning method that leverages multi-objective evolutionary computation to prune randomly initialized networks. Our approach aims to achieve automatic network pruning without the need for pre-training, by designing a unique encoding space that enables the exploration of various pruning criteria and rates. This enables the discovery of a range of pruning configurations without human intervention. Furthermore, to enhance computational efficiency, we propose a weight inheritance mechanism for acceleration. Compared to traditional vanilla evaluation, our proposed approach yields remarkably faster computation times, achieving speed improvements of hundreds of times. Moreover, it enables the simultaneous acquisition of all trained networks under different pruning configurations. Through extensive experiments conducted on VGG, Resnet, Googlenet, Densenet and Mobilenet architectures, we demonstrate that EvoFC can effectively prune untrained networks and identify the Pareto optimal solution set in terms of Error versus FLOPs.

Moving forward, our research will be dedicated to enhancing the sharing mechanism, aiming to minimize any performance degradation and further optimize algorithmic efficiency, especially in scenarios involving higher compression ratios. By diligently addressing these areas, our primary goal is to achieve superior algorithmic performance and yield significant overall improvements in the results obtained.

Acknowledgements

This work was supported by the National Natural Science Foundation of China (62372315), Sichuan Science and Technology Program (24ZDZX0007, 2023ZYD0143). This work was also financially supported by the China Scholarship Council (202106240095, 202106240096). This work was also supported by the Agency for Science, Technology, and Research (A*STAR) under its AME Programmatic Funds A1892b0026.

References

[1] Chen, S., Lin, L., Zhang, Z., Gen, M., 2019. Evolutionary netarchitecture search for deep neural networks pruning, in: Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence, pp. 189–196.

[2] da Cunha, A., Natale, E., Viennot, L., 2022. Proving the lottery ticket hypothesis for convolutional neural networks, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=Vjki79-619-.

[3] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T., 2002. A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE transactions on evolutionary computation 6, 182–197.

[4] Deng, L., Li, G., Han, S., Shi, L., Xie, Y., 2020. Model compression and hardware acceleration for neural networks: A comprehensive survey. Proceedings of the IEEE 108, 485–532.

[5] Diffenderfer, J., Kailkhura, B., 2021. Multi-prize lottery ticket hypothesis: Finding accurate binary neural networks by pruning a randomly weighted network, in: International Conference on Learning Representations.

[6] Ding, X., Ding, G., Guo, Y., Han, J., 2019. Centripetal sgd for pruning very deep convolutional networks with complicated structure, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 4943–4953.

[7] Frankle, J., Carbin, M., 2019. The lottery ticket hypothesis: Finding sparse, trainable neural networks, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=rJl-b3RcF7.

[8] Gong, Y., Sun, Y., Peng, D., Chen, P., Yan, Z., Yang, K., 2021. Analyze covid-19 ct images based on evolutionary algorithm with dynamic searching space. Complex & Intelligent Systems 7, 3195–3209.

[9] Han, S., Mao, H., Dally, W.J., 2016. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding, in: Bengio, Y., LeCun, Y. (Eds.), 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings. URL: http://arxiv.org/abs/1510.00149.

[10] 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.

[11] He, Y., Ding, Y., Liu, P., Zhu, L., Zhang, H., Yang, Y., 2020. Learning filter pruning criteria for deep convolutional neural networks acceleration, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 2009–2018.

[12] He, Y., Kang, G., Dong, X., Fu, Y., Yang, Y., 2018. Soft filter pruning for accelerating deep convolutional neural networks, in: IJCAI, pp. 2234–2240. URL: https://doi.org/10.24963/ijcai.2018/309.

[13] He, Y., Liu, P., Wang, Z., Hu, Z., Yang, Y., 2019. 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.

[14] 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.

[15] Hirsch, L., Katz, G., 2022. Multi-objective pruning of dense neural networks using deep reinforcement learning. Information Sciences 610, 381–400.

[16] Holland, J.H., 1992. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence.

[17] Hu, P., Peng, X., Zhu, H., Aly, M.M.S., Lin, J., 2021. Opq: Compressing deep neural networks with one-shot pruning-quantization, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 7780–7788.

[18] Huang, G., Liu, Z., Van Der Maaten, L., Weinberger, K.Q., 2017. Densely connected convolutional networks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4700–4708.

[19] Huang, Z., Wang, N., 2018. Data-driven sparse structure selection for deep neural networks, in: Proceedings of the European conference on computer vision (ECCV), pp. 304–320.

[20] Janis, C., 1976. The evolutionary strategy of the equidae and the origins of rumen and cecal digestion. Evolution , 757–774.

[21] Kim, H., Khan, M.U.K., Kyung, C.M., 2019. Efficient neural network compression, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 12569–12577.

[22] Koza, J.R., et al., 1994. Genetic programming II. volume 17. MIT press Cambridge.

[23] Krizhevsky, A., Hinton, G., et al., 2009. Learning multiple layers of features from tiny images .

[24] LeCun, Y., Bengio, Y., Hinton, G., 2015. Deep learning. nature 521, 436.

[25] Lee, N., Ajanthan, T., Torr, P., 2019. SNIP: SINGLE-SHOT NETWORK PRUNING BASED ON CONNECTION SENSITIVITY, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=B1VZqjAcYX.

[26] Leng, C., Dou, Z., Li, H., Zhu, S., Jin, R., 2018. Extremely low bit neural network: Squeeze the last bit out with admm, in: Thirty-Second AAAI Conference on Artificial Intelligence.

[27] Li, H., Kadav, A., Durdanovic, I., Samet, H., Graf, H.P., 2017. Pruning filters for efficient convnets, in: 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings, OpenReview.net. URL: https://openreview.net/forum?id=rJqFGTslg.

[28] Li, Y., Gu, S., Mayer, C., Gool, L.V., Timofte, R., 2020. Group sparsity: The hinge between filter pruning and decomposition for network compression, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 8018–8027.

[29] Liang, T., Glossner, J., Wang, L., Shi, S., Zhang, X., 2021. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing 461, 370–403. URL: https://www.sciencedirect.com/science/article/pii/S0925231221010894, doi:https://doi.org/10.1016/j.neucom.2021.07.045.

[30] Lin, M., Ji, R., Wang, Y., Zhang, Y., Zhang, B., Tian, Y., Shao, L., 2020. Hrank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 1529–1538.

[31] Lin, S., Ji, R., Yan, C., Zhang, B., Cao, L., Ye, Q., Huang, F., Doermann, D., 2019. 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.

[32] Liu, H., Simonyan, K., Yang, Y., 2019a. DARTS: Differentiable architecture search, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=S1eYHoC5FX.

[33] Liu, Y., Fan, B., Xiang, S., Pan, C., 2019b. Relation-shape convolutional neural network for point cloud analysis, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 8895–8904.

[34] 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.

[35] Liu, Z., Sun, M., Zhou, T., Huang, G., Darrell, T., 2019c. Rethinking the value of network pruning, in: International Conference on Learning Representations. URL: https://openreview.net/forum?id=rJlnB3C5Ym.

[36] Lu, Z., Whalen, I., Boddeti, V., Dhebar, Y., Deb, K., Goodman, E., Banzhaf, W., 2019. Nsga-net: neural architecture search using multi-objective genetic algorithm, in: Proceedings of the genetic and evolutionary computation conference, pp. 419–427.

[37] Luo, J.H., 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.

[38] Meng, F., Cheng, H., Li, K., Luo, H., Guo, X., Lu, G., Sun, X., 2020. Pruning filter in filter. Advances in Neural Information Processing Systems 33, 17629–17640.

[39] Mi, J.X., Feng, J., Huang, K.Y., 2022. Designing efficient convolutional neural network structure: A survey. Neurocomputing .

[40] Molchanov, P., Mallya, A., Tyree, S., Frosio, I., Kautz, J., 2019. Importance estimation for neural network pruning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 11264–11272.

[41] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV) 115, 211–252. doi:10.1007/s11263-015-0816-y.

[42] Salehinejad, H., Valaee, S., 2021. Edropout: Energy-based dropout and pruning of deep neural networks. IEEE Transactions on Neural Networks and Learning Systems 33, 5279–5292.

[43] Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., Chen, L.C., 2018. Mobilenetv2: Inverted residuals and linear bottlenecks, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 4510–4520.

[44] Simonyan, K., Zisserman, A., 2015. Very deep convolutional networks for large-scale image recognition, in: Bengio, Y., LeCun, Y. (Eds.), 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings. URL: http://arxiv.org/abs/1409.1556.

[45] Sun, Y., Xue, B., Zhang, M., Yen, G.G., 2019. Evolving deep convolutional neural networks for image classification. IEEE Transactions on Evolutionary Computation 24, 394–407.

[46] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z., 2016. Rethinking the inception architecture for computer vision, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 2818–2826.

[47] Wang, H., Zhang, Q., Wang, Y., Yu, L., Hu, H., 2019a. Structured pruning for efficient convnets via incremental regularization, in: 2019 International Joint Conference on Neural Networks (IJCNN), IEEE. pp. 1–8.

[48] Wang, J., Bao, W., Sun, L., Zhu, X., Cao, B., Philip, S.Y., 2019b. Private model compression via knowledge distillation, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1190–1197.

[49] Wen, W., Wu, C., Wang, Y., Chen, Y., Li, H., 2016. Learning structured sparsity in deep neural networks. Advances in neural information processing systems 29.

[50] Yao, Z., Cao, S., Xiao, W., Zhang, C., Nie, L., 2019. Balanced sparsity for efficient dnn inference on gpu, in: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 5676–5683.

[51] You, Z., Yan, K., Ye, J., Ma, M., Wang, P., 2019. Gate decorator: Global filter pruning method for accelerating deep convolutional neural networks. Advances in Neural Information Processing Systems 32.

[52] Yu, R., Li, A., Chen, C.F., Lai, J.H., Morariu, V.I., Han, X., Gao, M., Lin, C.Y., Davis, L.S., 2018. Nisp: Pruning networks using neuron importance score propagation, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 9194–9203.

[53] Yu, S., Yao, Z., Gholami, A., Dong, Z., Kim, S., Mahoney, M.W., Keutzer, K., 2022. Hessian-aware pruning and optimal neural implant, in: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pp. 3880–3891.

[54] Zhan, Z.H., Li, J.Y., Zhang, J., 2022. Evolutionary deep learning: A survey. Neurocomputing 483, 42–58.

[55] Zhang, X., Zhou, X., Lin, M., Sun, J., 2018. Shufflenet: An extremely efficient convolutional neural network for mobile devices, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 6848–6856.

[56] Zhou, B., Lapedriza, A., Khosla, A., Oliva, A., Torralba, A., 2017. Places: A 10 million image database for scene recognition. IEEE transactions on pattern analysis and machine intelligence 40, 1452–1464.

[57] Zhou, H., Alvarez, J.M., Porikli, F., 2016. Less is more: Towards compact cnns, in: European conference on computer vision, Springer. pp. 662–677.

[58] Zhuang, Z., Tan, M., Zhuang, B., Liu, J., Guo, Y., Wu, Q., Huang, J., Zhu, J., 2018. Discrimination-aware channel pruning for deep neural networks. Advances in neural information processing systems 31.