Efficient Network Compression via Gradient-Score Aware Pruning

Qiuying Li[a], Zhixiang Chen[b], Yu Li[a], Zhiyuan Jiang[a] and Shan Cao[a,*]

[a]School of Communication and Information Engineering, Shanghai University, Shanghai 200444, China
[b]The University of Sheffield, Sheffield S10 2TN, UK

ARTICLE INFO

Keywords:
Pruning
Zero-shot Learning
Neural Architecture Search (NAS)
Genetic Algorithm (GA)

ABSTRACT

Convolutional neural networks (CNNs) have demonstrated significant achievements in the field of computer vision, yet their high computational demands restrict practical application. Current pruning methods seek to mitigate this issue, which however often rely on heuristic manual approaches, encountering challenges in maintaining both significant model compression and accuracy. To address the above issues, a fast neural architecture search pruning (FNP) technique is proposed in this paper. Firstly, an importance matrix (IM) based preprocessing stage efficiently removes redundant structures by considering both weight importance and computational complexity, providing a compact baseline for subsequent pruning. Secondly, we adapt fast genetic algorithms (FGA) to identify optimally pruned model configurations. Furthermore, to accelerate the search process, we utilize a zero-shot learning approach to estimate model performance with the score of the frame (SoF), which is a gradient-based score. Compared with state-of-the-art (SOTA) pruning techniques, FNP demonstrates superior performance in terms of search duration and compression ratio. On the CIFAR-10 dataset, our method removes 95.24% of the parameters in VGG-16 while achieving a 0.72% accuracy improvement compared with the baseline. On the ImageNet dataset, we prune 68.98% of the parameters in ResNet-50 and obtain a 1.2% accuracy improvement compared with state-of-the-art (SOTA) approaches, while reducing the search time by 98.94%. The code is available at https://github.com/aqiu/FNP.git

1. Introduction

As artificial intelligence application technologies advance, the scale and complexity of convolutional neural networks (CNNs) have expanded rapidly. Deep learning models often contain numerous redundant parameters, resulting in inefficient use of storage and processing power. To address these issues, several model compression techniques have been introduced, such as quantization [1], pruning [2], knowledge distillation [3], and decomposition [4]. Among them, channel pruning is notable due to its minimal dependence on hardware, strong maintenance of accuracy, and significant improvement in inference speed. As a result, it has been widely used in accelerators [5, 6, 7] and has become an essential method to implement deep models in edge devices [8].

Current channel pruning methodologies typically face three persistent challenges that limit their effectiveness. Firstly, the allocation of pruning ratios constitutes a pivotal but challenging aspect, as improper pruning ratio settings frequently induce inefficient pruning and excessive computational costs. Conventional approaches typically employ fixed pruning ratios, which create rigid compression patterns and require laborious trial-and-error procedures to identify viable ratios. As demonstrated by [9], suboptimal pruning ratio allocation can lead to substantial accuracy degradation, particularly in deep neural architectures where layer sensitivity varies considerably. Secondly, prevailing pruning frameworks primarily rely on manually designed paradigms guided by empirical hyperparameters, such as global sparsity factors, which typically require extensive tuning. Although this manual process has been extensively investigated over prolonged research cycles, an effective solution to this challenge remains elusive. Thirdly, existing approaches struggle to achieve a reliable accuracy-compression trade-off. Extensive previous research has demonstrated that aggressive parameter reduction often leads to significant performance degradation [10].

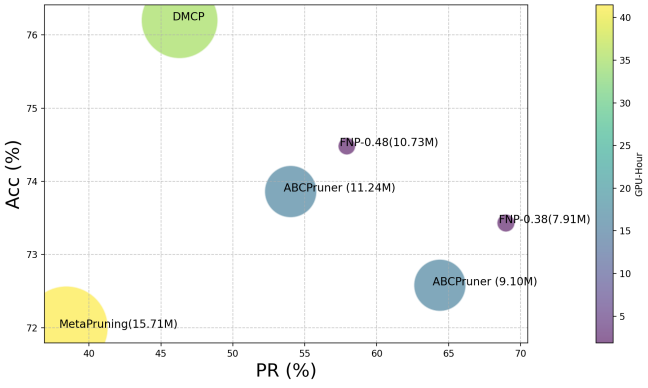

Figure 1: Comparison of AutoML based Pruning methods on ImageNet. We compare the pruning ratio (PR) of networks in the x-axis, and test accuracies (Acc) for the selected networks in the y-axis. The GPU-Hours are visualized by the circle size and color. FNP achieves the best balance among compression ratio, accuracy, and time efficiency, delivering consistent improvements across all metrics.

To address these challenges, many emerging automated machine learning (AutoML) based approaches have been proposed, in which Neural architecture search (NAS) plays

*Corresponding author: Shan Cao (E-mail: cshan@shu.edu.cn).
This work was supported in part by the National Natural Science Foundation of China (NSFC) under Grants 62271300 and 12141107, in part by the Shanghai Science and Technology Plan Project under Grants 24DP1501100 and 24DP1500600.

a key role. NAS is crucial for automating the design of pruning strategies and effectively searching for optimal pruning configurations that balance accuracy and compression. Although NAS-based methods theoretically eliminate manual heuristic design, their practical implementation introduces prohibitive computational costs. Typical NAS-driven pruning frameworks require iterative sampling and evaluation of numerous candidate architectures within the combinatorial search space. For example, recent work on automated model compression [11] demonstrates that even optimized search strategies still require resource-intensive exploration processes, severely limiting their scalability to complex network architectures.

Recently, two studies [12, 13] revealed that the performance of zero-shot learning methods remains robust when utilizing properly designed semantic proxies, regardless of whether they are derived from pre-trained feature extractors or constructed from scratch. This finding implies that the critical factor in zero-shot learning recognition lies in establishing effective proxy representations that bridge seen and unseen classes. Based on this insight, these approaches reduce the complexity of time. However, zero-shot learning has not been integrated into NAS-based pruning methods until now.

In this paper, we propose Fast Neural Architecture Search Pruning (FNP), an AutoML-based channel pruning method. FNP significantly reduces the cost of inefficient training time and maintains high accuracy at a high compression rate. It consists of a three-stage pipeline: an initial preprocessing phase, an FGA-based search stage, and a final fine-tuning to optimize performance. The main contributions of FNP are summarized as follows:

- The **importance matrix (IM)** is applied in the preprocessing stage to address the challenge of high computational cost in pruning. It is designed to automatically estimate the initial pruning ratio for each layer. Based on this guide, the preprocessing stage significantly accelerates the entire pruning process while maintaining competitive accuracy.
- In the search stage, a **fast genetic algorithm (FGA)** is proposed to address the problem of heavy dependence on manual expertise. FGA performs an automatic small-scale search on the PreprocessedNet. As a result, it can discover highly compressed models in a short time.
- In FGA, to address the limitations of compression-accuracy trade-off, **the score of the frame (SoF)** is defined. We use the SoF to evaluate the model based on the model architecture. The high-accuracy model with a high compression rate was selected using the SoF-based zero-shot learning approach.

The graph shown in Figure 1 offers a comprehensive overview of the performance characteristics of various ResNet-50 architectural variants. By integrating the proposed AutoML-based methods, the FNP-0.48 and FNP-0.38 configurations represent two distinct pruning regimes, achieving pruning ratios of up to nearly 70%. The accuracies are still higher than those of the approximate parameter methods. These results provide further evidence to support these claims.

2. Related Works

To better position our work, we review previous studies on model pruning, zero-shot learning, and AutoML, where AutoML includs NAS and other comprehensive frameworks.

2.1. Model Pruning

Model pruning is an essential technique for reducing the computational cost and memory footprint of deep neural networks while maintaining performance. Combined with recent review research[14, 15], network pruning is primarily categorized into weight pruning and channel pruning, both of which fall under structured pruning techniques. Magnitude-based pruning [7] is a conventional method that removes weights with the smallest absolute values to evaluate channel importance. Includes weight-based, gradient-based, and impact-based approaches. Weight-based methods measure importance using the ℓ_1 normalization and ℓ_2 normalization of the weights [16], where channels with lower values are considered less significant. Gradient-based methods [17] leverage information such as the rank matrix or sensitivity analysis to assess the impact of the channel on the loss function. Impact-based methods[18] evaluate importance by analyzing performance degradation when a channel is removed or masked. More advanced approaches, including structured pruning [19] and lottery ticket hypothesis[20], aim to find optimal sparse subnets. Recent studies integrate pruning with knowledge distillation and dynamic sparsification to improve efficiency without significant accuracy loss [21]. Despite these advancements, pruning often requires iterative fine-tuning, which increases the overall training cost.

2.2. Zero-shot Learning

Zero-shot learning [12] aims to recognize categories that are not present in the training data by leveraging auxiliary semantic information, such as attributes or word embeddings. Instead of relying on labeled instances for each class, it enables the model to infer unseen classes by transferring knowledge from seen classes through shared semantic representations. Earlier strategies rely on attribute-based embeddings [22] and semantic mappings [23] to bridge the gap between visible and unseen classes. More recent developments incorporate generative models, such as zero-shot learning based on GAN [24] and contrastive learning [25, 26] to create features for classes that have not been observed.

2.3. AutoML

AutoML aims to reduce the dependence on human expertise in the design and optimization of machine learning models by automating key tasks such as model selection, hyperparameter tuning, and architecture design [27, 28]. Among the various AutoML techniques, NAS [29, 30] automatically explores optimal network architectures. In addition, several AutoML-based methods have been specifically developed for model compression [11, 31, 32], aiming to automate the selection of efficient subnet structures and pruning configurations, thereby enhancing compression performance with minimal manual effort.

2.3.1. Conventional NAS Methods

With the development of automation, some recent studies focus on the search for the best channel configuration in each layer, rather than evaluating the importance of filters. NAS is achieved by reinforcement learning (RL), genetic algorithm (GA)[33], and gradient-based mode. More recent approaches, such as differentiable NAS DARTS [30], reduce search space and training time by gradient-based optimization. AACP [34] first trains and searches for efficient architectures within the one-shot model. ABCPruner [10] finds an optimal pruned structure using a reinforcement learning-based artificial bee colony algorithm. DMCP [21] makes the search process differentiable by modeling it as a factorized Markov process. TAS [35] exploits NAS to find the depth and width of a network to obtain the pruned networks. NPAS [36] introduces a compiler-based joint network pruning and architecture search, determining the filter type, the pruning scheme, and the ratio for each layer. However, despite their success, most NAS-based pruning methods still suffer from prohibitively high computational costs. Architecture evaluation is typically iterative, often involving repeated training or inference. This repetition consumes significant time and computational resources. Consequently, these methods struggle to scale and are less feasible for deployment on edge or resource-limited devices.

2.3.2. Zero-shot NAS Methods

Zero-shot NAS aims to efficiently select high-performing neural network architectures by rapidly evaluating and ranking candidate networks without training, using proxy metrics that estimate their potential performance. In recent years, numerous training-free proxies have been developed to improve the accuracy and generalizability of the evaluation. SNIP [17] calculates the scores by estimating how weight pruning affects loss. GraSP [37] introduces the preservation of the gradient signal to account for optimization behavior. SynFlow [38] uses a data-agnostic signal propagation method to calculate global importance scores in a stable way. GFP [39] evaluates the importance of the group-level filter by structural grouping and gradient flow. While these methods focus on measuring the importance of parameters or structures without supervision, recent work such as ZiCo [40] explores a different direction. ZiCo incorporates few-shot supervision to enhance zero-cost proxies, which significantly improves the correlation with actual model performance, making it a promising approach to achieve accurate and efficient model evaluation.

2.3.3. Other AutoML-Based Methods

In addition to being used in NAS and zero-shot NAS, AutoML has demonstrated its effectiveness in many other areas. Bayesian optimization [41], genetic algorithms [42], and gradient-based pruning [21] are commonly used for hyperparameter optimization. Meta-learning [43] and transfer learning [44] further improve efficiency by leveraging prior knowledge. Although AutoML frameworks such as Auto-sklearn [45] and Google AutoML [46], demonstrated promising results, challenges remain in interpretability, scalability, and computational cost.

3. Proposed FNP Method

While channel pruning and AutoML have achieved promising results, existing methods frequently incur substantial computational costs and require extensive human effort for fine-tuning. A notable advancement by Liu et al. [11] highlights that the effectiveness of channel pruning is largely determined by the resulting network structure, rather than by the specific selection strategy. Similarly, our FNP method is a lightweight and training-free framework that emphasizes structure-aware pruning guided by adaptive search.

3.1. The Framework of FNP

In this section, we introduce our FNP approach for automatically pruning channels in deep neural networks, which achieves a pruned network that efficiently meets both high compression and high accuracy. To achieve this, we propose a multi-stage automatic pruning method, as shown in Figure 2: First, we define the original network as the UnprunedNet, and preprocess it to obtain the PreprocessedNet, which incorporates an accurate and adaptive channel-wise pruning ratio without human intervention. Second, we further compress the PreprocessedNet by conducting a local search using FGA, evaluating candidates through a comparison of their SoF to find the optimal PrunedNet. Third, we apply fine-tuning to restore the accuracy of the optimal PrunedNet, and the final model is defined as the FinetunedNet.

We formulate the channel pruning problem as an optimization task, where a large-scale model with a substantial number of channels as input, and the objective is to generate a PrunedNet that maintains performance while reducing computational complexity. First, the UnprunedNet is represented as $\mathcal{N}(\mathcal{V})$, where $\mathcal{V} = [C_1, C_2, \dots, C_i, \dots, C_L]$ denotes the channel configuration of the L-layer network. C_i represents the number of channels in the i-th layer. Then, after preprocessing, the configuration of the PreprocessedNet is represented as a new vector, named channel vector $\mathcal{V}^* = [C_1^*, C_2^*, \dots, C_i^*, \dots, C_L^*]$, where C_i^* represents the number of retained channels in the i-th layer after pruning. Finally, during the search stage, the s-th PrunedNet in the

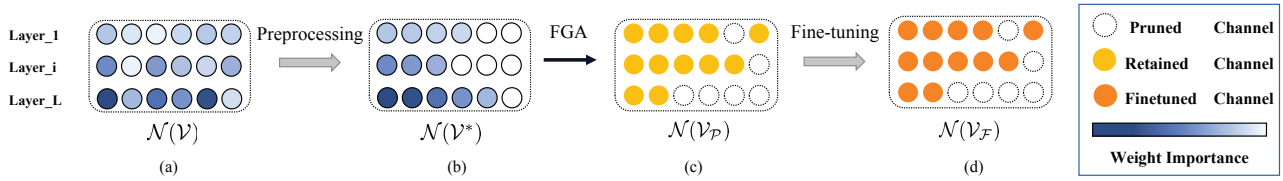

Figure 2: The overall pipeline of FNP. (a) The UnprunedNet $\mathcal{N}(\mathcal{V})$. (b) The PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$. (c) The optimal PrunedNet $\mathcal{N}(\mathcal{V}_\mathcal{P})$. (d) The FinetunedNet $\mathcal{N}(\mathcal{V}_\mathcal{F})$.

n-th iteration, denoted $\mathcal{N}(\mathcal{V}^*_{ns})$, is obtained. We formulate the channel pruning problem as:

$$\max \textbf{Score}\left(\mathcal{N}(\mathcal{V}^*_{ns}), \theta, \theta_b\right) \quad \text{s.t.} \quad \min \theta \tag{1}$$

Our objective is to identify the PrunedNet that achieves the highest **Score** (i.e., accuracy) while minimizing the number of parameters θ without training or inference. θ_b is the initial parameters. The details of this formula are provided in Section 3.2.2. We will perform experimental verification in Section 4, and the methods are described in detail below.

3.2. UnprunedNet Preprocessing

The original channel pruning method takes an inordinate amount of time to pre-train. The pruning configuration is obtained by ranking the importance of the weights during the training process and combining it with a global pruning ratio. However, the granularity of the specific larger-wise pruning ratio assigned to each layer, which is guided by human-designed heuristics, tends to be relatively coarse.

Our method FNP incorporates IM-based preprocessing to determine an initial layer-wise pruning ratio. In the first stage, the UnprunedNet $\mathcal{N}(\mathcal{V})$ undergoes a preliminary pruning process using IM, which serves as a preprocessing step to accelerate the subsequent search procedure.

Recent works suggest designing the pruning ratio based on the model rather than relying on predefined heuristics [7]. Li et al. [47] proposed an adaptive method to determine the pruning ratio based on the weight properties of the network. Compared with traditional heuristic-based pruning strategies, this approach significantly enhances model compression performance. Similarly, we introduce a compound weight metric, IM, to evaluate the importance of each channel. This metric is designed to comprehensively consider the importance of weights, computational complexity, and the global pruning ratio, enabling the identification and removal of redundant filters in each layer of the network during the preprocessing stage. Figure 3 and Algorithm 1 illustrate an overview of the preprocessing pipeline from different angles. Figure 3 also illustrates that the workflow consists of four sequential steps: ranking the weights, reshaping their dimensions, determining the threshold, and selecting the optimal channel configuration. The following presents a set of relevant equations.

Algorithm 1: Preprocessing Pipeline

Input: Pre-trained weights of the UnprunedNet $\mathcal{N}(\mathcal{V})$, per-layer computational complexity, pruning ratio γ

Output: Retained weights N_{IM}, threshold $thre$, the PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$

1: Compute the importance weight matrix W_{IM} ;
2: Count important weights as N_{IM} (Eq. 2);
3: Rank the weights of $\mathcal{N}(\mathcal{V})$;
4: Reshape $\mathcal{N}(\mathcal{V}')$ to one-dimension
5: Compute retained weights N_{m} (Eq. 3);
6: Define the threshold $thre$ as the $(N_{\text{m}})^{th}$ element in W_{m}(Eq. 4);
7: Select weights that satisfy the threshold (Eq. 5);
8: Construct the initial channel configuration (Eq. 6-Eq. 7);
9: **return** the PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$;

$$N_{\text{IM}} = \#\left(\frac{|(\mathbf{k}_i^j)_q|}{\left(\#FLOPs_i\right)^\lambda}\right) \tag{2}$$

First, as described in Eq. 2, the importance weight W_{IM} is computed using pre-trained weights $|(\mathbf{k}_i^j)_q|$ and per-layer computational complexity $\#FLOPs_i$. The number of elements in W_{IM} is counted and denoted as N_{IM}, where # denotes the statistical count of its input. We use $(\mathbf{k}_i^j)_q$ as the q-th weight element in the j-th kernel of $\mathbf{k}_i$ in the i-th layer, where $|\cdot|$ returns the absolute value of its input. $\#FLOPs_i$ returns the computational complexity count in the i-th layer, and $\lambda \geq 0$ is a hyperparameter shared across the UnprunedNet $\mathcal{N}(\mathcal{V})$. λ acts as a trade-off parameter that balances two competing objectives. On the one hand, the algorithm tries to maximize the accuracy of the model, and on the other hand, it seeks to reduce model complexity. A larger λ places more emphasis on the impact of FLOPs, making layers or weights with higher computational cost more likely to be pruned, while a smaller λ favors traditional magnitude-based pruning. Adjusting λ, the algorithm can control the balance between efficiency and performance.

Subsequently, integrating this formula with the global pruning ratio γ, we significantly accelerate the calculation of the pruning ratio p_i for each layer during the preprocessing phase. The number of retained weights N_m is calculated using γ through Eq. 3:

$$N_m = N_{\text{IM}} * (1 - \gamma) \tag{3}$$

After ranking and reshaping all the weights, the top N_m weights are selected to form the retained weight set W_m. Then the threshold *thre* is selected as shown in Eq. 4:

$$thre = W_m\{N_m - 1\} \tag{4}$$

Finally, weight pruning converts each filter $(\mathbf{k}_i^j)_q$ into a sparse tensor $(\hat{\mathbf{k}}_i^j)_q$, whose elements are defined by Eq. 5.

$$(\hat{\mathbf{k}}_i^j)_q = \begin{cases} 0, & (\mathbf{k}_i^j)_q < thre \\ (\mathbf{k}_i^j)_q, & \text{otherwise.} \end{cases} \tag{5}$$

Consequently, this leads to the number of channels $\#(\hat{\mathbf{k}}_i^j)_q$ for each layer and the pruning ratio p_i for each layer in Eq. 6, where # denotes the statistical count of its input. The number of channels in the PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$ is calculated by Eq. 7.

$$p_i = \frac{\#(\hat{\mathbf{k}}_i^j)_q}{\#(\mathbf{k}_i^j)_q} \tag{6}$$

$$C_i^* = (1 - p_i)C_i \tag{7}$$

Based on this pre-optimization method, we use the IM characteristics of the UnprunedNet $\mathcal{N}(\mathcal{V})$ to construct a local search space in the preprocessing stage. This method allows us to optimize the design of the layer-wise pruning ratio without relying on human expertise.

3.3. Preprocessed Network Fast Search

To further enhance model compression, we efficiently explore the optimal architecture using NAS, ensuring an optimal trade-off between compactness and performance. Typical NAS-driven pruning frameworks require iterative sampling and evaluation of numerous candidate architectures within the combinatorial search space, which is a huge time-consuming process. To save time, we use two strategies in our FGA method. First, we incorporate automation-based methods to perform GA, transforming the global search into a localized search guided by adaptive preprocessing. Second, during the search process, the accuracy of PrunedNet is evaluated based on the SoF, eliminating the time-consuming inference step. Furthermore, considering that pruning methods inherently involve a trade-off between compression and accuracy, our ultimate objective is to achieve a higher compression structure under the same global pruning ratio setting by leveraging FGA.

3.3.1. Fast Genetic Algorithms

After the PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$ is identified, we obtain an initial set of reserved channels $\mathcal{V}^*$. Since traditional NAS methods based on GA operate as a global search process, the compressed model found during the search often suffers from accuracy degradation.

To maintain performance while further compressing, we adopt our variant of FGA for fine-grained search optimization. Details of the FGA are illustrated in Figure 4. (a).

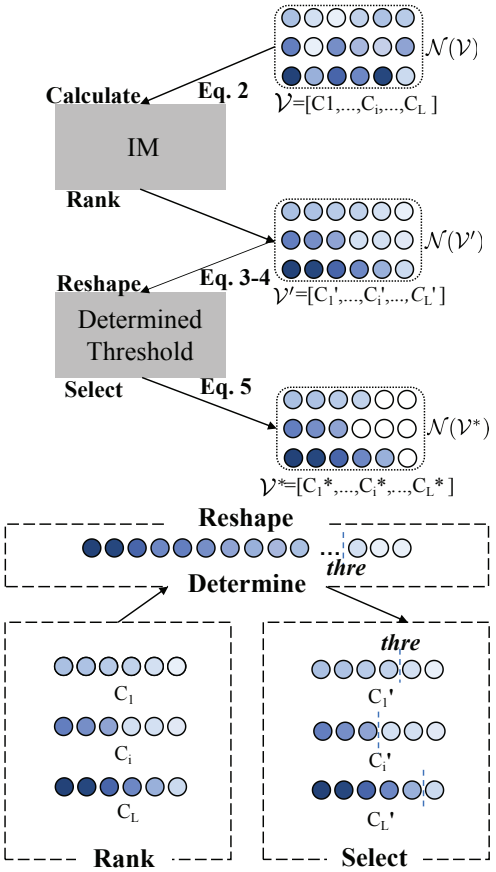

Figure 3: Preprocessing: The pipeline of the procedural framework from computing the Importance Matrix (IM) to the PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$ through a sequence of analytical operations.

The PreprocessedNet $\mathcal{N}(\mathcal{V}^*)$ is used as input. The search stage involves two modes: mutation, and crossover. In the first search stage, $\mathcal{N}(\mathcal{V}^*)$ are randomly assigned to different structures. Then, in each subsequent generation, the PrunedNets $\mathcal{N}(\mathcal{V}_{ns}^*)$ will undergo mutation and crossover. After computing the SoF for each candidate, we rank them accordingly for selection. The top K PrunedNets with the highest scores are selected as parents for the next generation. Mutation generation utilizes a perturbation factor mechanism to determine the number of channel variations in each layer. Figure 4. (b) mainly shows the first iteration. Figure 4. (c) mainly shows the final iteration, and the PrunedNet with the highest score is selected as the optimal PrunedNet $\mathcal{N}(\mathcal{V}_p)$. The detailed FGA algorithm is presented in Algorithm 2.

Mutation is a random perturbation mechanism that modifies the channel vector of the network. This mechanism generates a perturbation vector within a specified range, enabling random selection of channel configurations within the search space while ensuring compliance with channel constraints. This stochastic variation process ensures model diversity in the search space and provides high-quality initial candidates for subsequent iterations. Crossover involves randomly selecting two parent models and recombining their channel vectors to generate offspring.

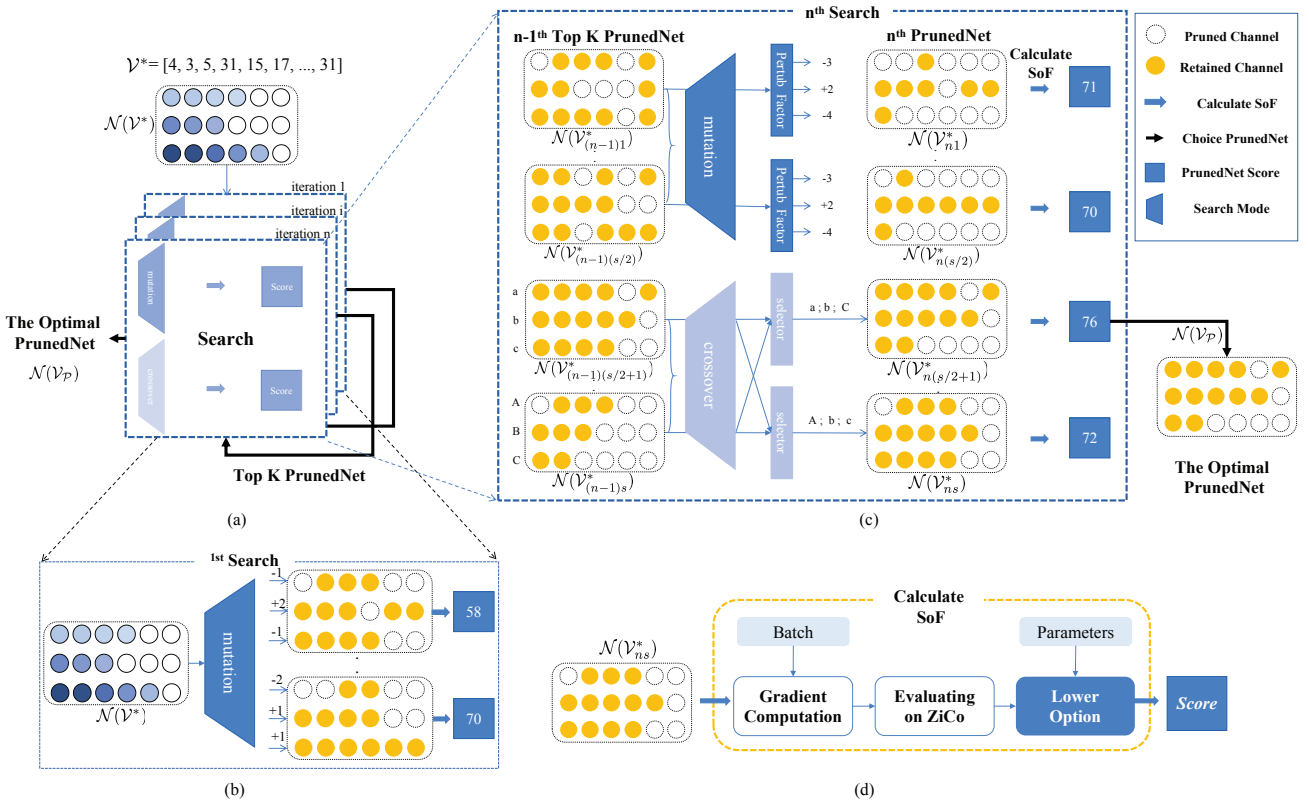

Figure 4: Framework of FGA. (a) The overall pipeline of FGA. (b) The illustrations of the first iteration of searching. (c) The illustrations of the perturbation factors mechanism in the mutation and selection in the crossover. (d) The processing pipeline from model loading to SoF computation.

The PrunedNets of each evolutionary iteration are subsequently evaluated and scored to assess the performance of the model. After evaluating and ranking all candidates according to their fitness scores, the current iteration provides the top K offspring for subsequent generations. This process is repeated for $n-1$ generations. Ultimately, the highest-scoring pruned network, as the optimal PrunedNet $\mathcal{N}(\mathcal{V}_\mathcal{P})$ that meets the specified fewest constraints, is obtained in the last iteration.

3.3.2. *Score of the Frame (SoF)*

In the conventional GA framework, randomly generated networks are typically evaluated through inference or training. However, the above-mentioned evaluation process is subject to uncertainty due to its heavy dependence on the choice of network weights. Training is time-consuming, and the inherent variability in inference introduces randomness, making it difficult to accurately reflect the precision.

Recently, ZiCo [40] found that optimizing for both a high absolute mean and low variance in gradients is crucial for achieving efficient training and strong generalization. Based on this insight, they introduced a strategy to shorten the search duration by advancing the concept of zero-shot NAS. This approach centers on the development of a training-free proxy metric. The SoF aims to estimate the test performance of a specific architecture without requiring full training. It introduces a zero-shot metric for evaluating neural network architectures without the need for training.

Our method gains the ability of the SoF to characterize model performance by computing the ratio between the expected absolute gradient magnitude and the standard deviation. A logarithmic transformation is applied to the computed ratio to improve numerical stability. This method serves as a training-free proxy to reflect the effectiveness of the architecture without any expenditure of training time. A higher SoF value indicates a more favorable architecture, as it implies more stable and effective parameter updates with a flatter loss landscape, which facilitates better optimization and generalization. Notably, the SoF is architecture-agnostic and requires only two input batches ($B = 2$) to achieve the performance of state-of-the-art (SOTA) methods among existing accuracy predictors. As a result, it serves as an efficient and reliable zero-shot NAS proxy, allowing rapid selection of superior architectures without the need for expensive training.

The specific calculation details of the score are shown in Figure 4. (d). To compute the SoF for the model, we combine the batch of the model with the gradients of the model. Without selecting or loading any weights, the framework evaluates model performance using just one backpropagation step. Compared with training over several epochs, this approach significantly reduces the time required, and

the immediate scoring of these architectures enhances the predictability and accelerates the efficiency of the search process. By combining the proxy score and parameter count in a composite formulation, we obtain an optimal model with a higher compression ratio. First, the layer L and the standard deviation σ_ω are defined as follows:

$$L = L(\mathbf{X}_k, y_k; \theta), \quad k \in \{1, ..., B\} \tag{8}$$

$$\sigma_\omega = \sqrt{\mathrm{Var}(|\nabla_\omega L|)} \tag{9}$$

Here, B denotes the number of training batches used to compute SoF; θ denotes the initial parameters of a given L-layer network; ω represents each element in the parameters θ_i of the $i-$th layer of the network; X_k and y_k are the k-th input batch and corresponding labels from the training set, respectively. Then, the evaluation of the performance of PrunedNet is conducted through the following scoring calculation, in which the logarithmic transformation serves to stabilize the computation and compress the scale of the sensitivity metrics:

$$Score_{ZiCo} = \sum_{i=1}^{L} \log\left(\sum_{\omega\in\theta_i} \frac{\mathbb{E}[|\nabla_\omega L|]}{\sigma_\omega}\right) \tag{10}$$

We first consider the original proxy score as the baseline evaluation. To encourage the search process to favor more compact models, we introduce a parameter penalty term that explicitly accounts for the number of parameters. This penalty guides the optimization toward smaller networks while maintaining accuracy. Furthermore, we apply a logarithmic transformation to the penalty term, which regularizes the effect of parameters by preventing extremely large or small values from disproportionately dominating the score. The resulting formulation is defined as follows:

$$\textbf{Score}\left(\mathcal{N}(\mathcal{V}^*_{ns}), \theta, \theta_b\right) = Score_{ZiCo} - \log\left(1 + \frac{\theta}{\theta_b}\right) \tag{11}$$

In summary, by leveraging the SoF-guided FGA approach, we optimize the efficiency of the compression search process, which leads to a substantial reduction in parameters and significantly shortens the time needed for fine-tuning.

Algorithm 2: Fast Genetic Algorithm

Input: PreprocessedNet: $\mathcal{N}(\mathcal{V}^*)$;
Perturb Factor: $\mathcal{F}$;
Constraints: θ;
Hyperparameters: Population Size g, Max Iterations n, Number of 1^{st} Iteration Gene s, Number of Mutation Genes per Iteration $\frac{s}{2}$, Number of Crossover genes per Iteration $\frac{s}{2}$, Candidates Generated per Iteration V.
Output: Optimal pruned structure: $\mathcal{V}_{top1}$

$\mathcal{V}_{topK} = \emptyset$;
for $i = 1$ *to* s **do**
  PrunedNet($\mathcal{V}_i$) = Mutation($\mathcal{V}^*$,1), s.t. $\theta, \mathcal{F}$;
  Compute the SoF of PrunedNet in Eq. 8-Eq. 11:
    $\{\mathcal{V}_i, Score_{ZiCo}\}$ = Calculate(PrunedNet($\mathcal{V}_i$));
    $\{\mathcal{V}_i, \textbf{Score}\}$ = Calculate($Score_{ZiCo}, \theta$);
end
$\{\mathcal{V}_{topK}, Score_{topK}\}$ = TopK($\{V, \textbf{Score}\}$);
for $j = 1$ *to* n **do**
  $V_{mutation}$ = Mutation($V_j, \frac{s}{2}$), s.t. $\theta, \mathcal{F}$;
  $V_{crossover}$ = Crossover($V_j, \frac{s}{2}$), $s.t.\theta$;
  $V_j = V_{mutation} + V_{crossover}$;
  Compute the SoF of PrunedNet in Eq. 8-Eq. 11:
  $\{\mathcal{V}_{topK}, Score_{topK}\}$ = TopK($\{V_j, \textbf{Score}\}$);
end
$\{\mathcal{V}_{top1}, Score_{top1}\}$ = Top1($\{V_{topK}, Score_{topK}\}$);
return $\mathcal{V}_{top1}$;

3.4. Fine-tuning

After conducting the FGA-based search under high compression, we obtain the optimal PrunedNet $\mathcal{N}(\mathcal{V}_\mathcal{P})$. We apply a fine-tuning process to ensure that $\mathcal{N}(\mathcal{V}_\mathcal{P})$ maintains high performance while maintaining computational efficiency. Drawing inspiration from contemporary research, we employ a K-Nearest Neighbors (KNN) approach to select crucial weights for the fine-tuning process. This methodology deviates from the conventional approach by assessing the collective significance of a filter subset of size, rather than evaluating the importance of individual filters. It prioritizes the selection of the most representative filters within the entire network architecture. This approach reduces the duration of the fine-tuning phase and enables the pruned network to regain the accuracy of the original model at a faster rate.

4. Experiments

As outlined in Section 3, the FNP pruning pipeline involves three critical stages: preprocessing, searching, and fine-tuning. To systematically assess the impact of each phase, we conducted a series of controlled experiments. This section details our experimental design, including dataset specifications, hyperparameter settings, and evaluation protocols. Then, we present a comparative analysis of the performance of our method, demonstrating the advantages in both efficiency and effectiveness. First, we describe the details of FNP implementation in Section 4.1. Then we compare our results with SOTA channel pruning methods and AutoML-based methods in Sections 4.2 and 4.3, respectively, including comparisons of VGGNet, GoogLeNet, and ResNet-56/110 on CIFAR-10, and ResNet-50 on ImageNet. Finally, to study the effectiveness of the main component in our methods, we conduct an ablation study in Section 4.4.

4.1. Implementation Details

Preprocessing & Searching. The preprocessing and searching are both conducted on Nvidia RTX 3090 GPUs with a 256-batch size. In the preprocessing stage, we set an initial pruning starting point by applying different ranges

Table 1
Pruning Results of VGGNet-16 on CIFAR-10.

Model	Top-1	Parameters	Pruned
Baseline	93.02%	14.73M	-
SPCRC[48]	93.90%±0.05%	2.97M	82.19%
GAL-0.1[49]	93.42%	2.67M	82.20%
FICP[50]	93.42%	2.67M	82.20%
CHIP [51]	93.72%	2.08M	85.90%
FNP-0.7	**93.93%**	**1.81M**	**87.74%**
HRank[9]	91.23%	1.78M	87.92%
GCN[52]	93.08%	1.06M	92.80%
CLR-RNF[47]	93.32%	0.74M	95.00%
FNP-0.86	**93.67%**	**0.70M**	**95.24%**

Table 2
Pruning Results of ResNet-56 on CIFAR-10.

Model	Top-1	Parameters	Pruned
Baseline	93.26%	0.85M	-
CFD[53]	93.81%	0.60M	27.59%
FNP-0.4	**93.92%**	**0.50M**	**40.80%**
SPCRC[48]	91.65±0.01%	0.30M	64.71%
GAL-0.8[49]	91.58%	0.29M	65.88%
HRank [9]	90.72%	0.27M	68.24%
CLR-RNF [47]	92.32%	0.26M	69.41%
GCN [52]	92.75%	0.25M	70.58%
FNP-0.7	**93.10%**	**0.25M**	**70.58%**

Table 3
Pruning Results of ResNet-110 on CIFAR-10.

Model	Top-1	parameters	Pruned
Baseline	93.50%	1.72M	-
CFD [53]	94.48%	0.85M	50.70%
FNP-0.5	**94.49%**	**0.85M**	**50.70%**
HRank [9]	92.65%	0.53M	68.70%
CHIP[51]	93.50%	0.54M	68.30%
CLR-RNF[47]	93.71%	0.53M	69.14%
FNP-0.7	**93.95%**	**0.49M**	**71.51%**

of the preprocessing pruning ratio γ to different models. The specific λ settings details are in the appendix. During the search process, 50 genes are generated in each iteration through two modes: mutation, and crossover. The number of mutation genes and crossover genes of each iteration is 25. The perturbation factor is set within a fluctuation range of 5 to expand the search space. We select the top 10 genes for the next iteration. The probability of mutation is set to 0.1.

Fine-tuning Strategy. We use Stochastic Gradient Descent (SGD) for fine-tuning, with momentum set to 0.9 and a 256-batch size. On CIFAR-10, the weight decay is set to 5e-3 for fine-tuning. We fine-tune the optimal PrunedNet $\mathcal{N}(\mathcal{V}_\mathcal{P})$ with a learning rate of 0.01, which is reduced by a factor of 10 every 50 training epochs on VGGNet. The initial learning rate for the UnprunedNet $\mathcal{N}(\mathcal{V})$ is 0.2, which is reduced to 0.02 by the cosine scheduler for GoogLeNet and ResNet-56/110. On ImageNet, the weight decay is set to 1e-4 for fine-tuning. The learning rate is set to 0.1 and reduced to 0.02 using the cosine scheduler.

4.2. Comparison with the Conventional methods

4.2.1. Results on CIFAR-10

On CIFAR-10, we compare our FNP method with various traditional SOTA pruning methods. More detailed analyses are provided below.

Results on VGG16: For VGGNet, we apply our FNP to prune the 16-layer VGGNet model, a widely utilized sequential CNN architecture for object detection and semantic segmentation tasks. As shown in Table 1, under a preprocessing ratio γ of 0.7. We remove 87.74% of the channels. Our FNP can achieve 1.81M parameters while still keeping the accuracy at 93.93%. Under a preprocessing ratio γ of 0.86. We remove 95.24% of the channels. Our FNP can achieve 0.70M parameters while still keeping the accuracy at 93.67%, better than most SOTA methods.

Results on ResNet: In our study, we have selected two distinct depths of ResNets, namely ResNet-56, and ResNet-110, to showcase the proficiency of our FNP technique in optimizing networks that incorporate residual blocks. Specifically, for ResNet-56, as shown in Table 2, under a preprocessing ratio γ of 0.4, the channel reduction is enhanced to 40.80%; the amount of parameters is reduced from 0.85M to 0.50M, with the accuracy reaching 93.92%, notably improved over the baseline, under a preprocessing ratio γ of 0.7, the channel reduction is enhanced to 70.58%; the amount of parameters is reduced from 0.85M to 0.25M, with the accuracy reaching 93.10%, which is a notable improvement over the same compression level achieved by the GCN method[52]. For ResNet-110, as shown in Table 3, under a preprocessing ratio γ of 0.5, the channel reduction is enhanced to 50.70%, the amount of parameters is reduced from 1.72M to 0.85M, with a 94.49% accuracy. Under a preprocessing ratio γ of 0.7, the channel reduction is enhanced to 71.51%, the amount of parameters is reduced from 1.72M to 0.49M, with an accuracy of 93.95, which is improved over the baseline. This superior performance underscores the effectiveness of FNP in maintaining network accuracy while achieving significant compression. The FNP approach adaptively identifies and eliminates redundancies, thereby streamlining the network architecture.

Results on GoogLeNet: GoogLeNet is the representative network with a multi-branch structure. As we can see from Table 4, under a significantly high preprocessing ratio γ of 0.95, we remove 65.69% of inefficient channels with only a 0.02% performance drop in accuracy. Compared with the best SOTA (e.g., FICP, HRank, GAL), it achieves higher accuracy performance while significantly reducing the number and parameters.

4.2.2. Results on ImageNet

We further perform our method on large-scale ImageNet for ResnNet-50. We present two different pruning ratios for ResNet-50 in the preprocessing stage. The results are reported in Table 5. The ResNet-50 is a network with a compact design, which may contain fewer redundant parameters. For ResNet-50, a representative deep shortcut architecture,

Table 4
Pruning Results of GoogLeNet on CIFAR-10.

Model	Top-1	Parameters	Pruned
Baseline	95.05%	6.15M	-
GAL-0.05[49]	94.56%	3.12M	49.30%
GAL-ApoZ[49]	92.11%	2.85M	53.70%
FICP [50]	94.75%	2.77M	55.00%
HRank [9]	94.53%	2.74M	55.40%
CLR-RNF[47]	94.85%	2.18M	64.70%
FNP-0.95	**95.03%**	**2.11M**	**65.69%**

Table 5
Pruning Results of ResNet-50 on ImageNet.

Model	Top-1	Parameters	Pruned
Baseline	76.60%	25.50M	-
GCN[52]	59.26%	14.63M	41.92%
HRank [9]	71.98%	13.77M	46.00%
SPCRC [48]	75.00±0.1%	-	-
FNP-0.25	**75.10%**	**15.09M**	**40.82%**
GCN[52]	57.67%	10.03M	61.16%
CLR-RNF [47]	73.34%	9.00M	64.77%
CHIP [51]	73.30%	8.01M	68.60%
FNP-0.38	**74.50%**	**10.82M**	**57.57%**
FNP-0.48	**73.85%**	**7.99M**	**68.67%**

both the training and search processes are time-consuming. As can be observed from Table 5, under a preprocessing ratio γ of 0.25, our model achieves an accuracy of 75.10% while achieving a compression ratio of 40.82%, under a preprocessing ratio γ of 0.38, the ultimate compression ratio reaches 57.57%, and our model outperforms HRank by 2.52% in accuracy. We further compress the model to enhance effectiveness, under a higher preprocessing ratio γ of 0.48, we have removed 68.67% of the inefficient channels. In contrast to the SOTA method (e.g., CHIP), our accuracy still surpasses those with a similar compression ratio by 0.55%.

4.3. Comparison with AutoML-based methods

4.3.1. Result on CIFAR-10

In this section, we present the experimental results that compare our proposed FNP method with various SOTA AutoML-based approaches on the CIFAR-10 dataset with a focus on compressing computational complexity. Compared with Artificial-Bee-colony-Algorithm-based method ABCPruner[10] and RL-based pruning method AACP [34], we can find a more compressed space, further compression based on maintaining accuracy.

In Table 6, with the FNP approach, under a preprocessing ratio γ of 0.86, VGGNet-16 achieves an accuracy of 93.67% while achieving a compression ratio of 75.25%, effectively and significantly pruning the VGGNet-16 architecture while maintaining the accuracy of the model. Under the FNP method, ResNet-56 with a preprocessing ratio γ of 0.56 achieves an accuracy of 93.62%, which is significantly higher than that of the vast majority of other

Table 6
Compared with SOTA AutoML-based methods on CIFAR-10.

Model	Top-1	FLOPs	Pruned
VGGNet-16	93.02%	314.59M	-
CCEP [54]	94.74%	115.77M	63.20%
AACP [34]	93.57±0.12%	94.38M	70.00%
ABCPruner[10]	93.13%	82.81M	73.68%
FNP-0.86	**93.67%**	**77.86M**	**75.25%**
ResNet-56	93.26%	127.62M	-
DAIS[55]	93.53%	61.90M	51.50%
MFP [56]	93.56%	60.49M	52.60%
AACP[34]	93.31±0.28%	63.81M	50.00%
ABCPruner[10]	93.23%	58.54M	54.13%
CCEP [54]	93.48%	46.68M	63.42%
FNP-0.56	**93.62%**	**56.82M**	**55.48%**
GoogLeNet	95.05%	1534.55M	-
ABCPruner[10]	94.84%	513.19M	66.56%
FNP-0.95	**95.03%**	**511.64M**	**66.66%**

AutoML-based approaches, while simultaneously maintaining lower computational cost in terms of computational complexity. GoogLeNet with the highest preprocessing ratio γ of 0.95 under the FNP method outperforms the SOTA (e.g., ABCPruner[10]), which also employs a genetic algorithm, achieving a 0.19% higher accuracy in terms of compression efficiency. Compared with the baseline, it just experiences a slight decrease of 0.02% in accuracy.

In summary, the proposed FNP method significantly reduces computational complexity while preserving accuracy. Experiments on models like VGGNet-16, ResNet-56, and GoogLeNet confirm its efficiency and competitive performance compared to other SOTA AutoML-based methods.

4.3.2. Result on ImageNet

We further compare FNP with AutoML-based pruning methods on ImageNet, including the Markov-Chain-based method DMCP [21], GA-based method MetaPruning [11], and Artificial-Bee-colony-Algorithm-based method ABCPruner [10].

In Table 7 we find FNP has both high effectiveness and efficiency, which is superior to other AutoML-based methods. Compared with MetaPruning [11], the superiority of FNP can be attributed to its scoring design, where the accuracy of the pruned architecture is directly measured within the framework of the network without any inference. We evaluated ResNet-50 with preprocessing ratios γ of 0.25 and 0.38. FNP also advances in the adaptive search structure. It effectively verifies the finding in ABCPruner [10] that the optimal PrunedNet $\mathcal{N}(\mathcal{V}_{\mathcal{P}})$ is more important in the channel pruning methods, rather than selecting "important" channels. Compared with the compression and accuracy achieved by other SOTA AutoML-based methods, FNP demonstrates significant reductions in both parameters and computational complexity. For instance, ResNet-50 with a preprocessing ratio γ of 0.25, computational complexity decreased from 4135.70M to 1949.52M, which pruned

Table 7
Compared with SOTA AutoML-based methods on ImageNet.

Model	GPU-Time	Top-1	Parameters	Pruned	FLOPs	Pruned
ResNet-50	-	76.60%	25.50M	-	4135.70M	-
MetaPruning [11]	41.48 hours	72.17%	15.72M	38.43%	2260.00M	45.35%
DMCP [21]	35.03 hours	76.20%	-	-	2200.00M	46.30%
FNP-0.25	**0.44 hours**	**75.10%**	**15.09M**	**40.82%**	**1949.52M**	**52.86%**
ABCPruner-0.8 [10]	16.24 hours	73.86%	11.24M	54.02M	1890.60M	54.29%
FNP-0.38	**0.65 hours**	**74.50%**	**10.82M**	**57.57%**	**1340.37M**	**67.59%**

52.86%, parameters decreased from 25.50M to 15.09M, which pruned 40.82%, while maintaining a precision of 75.10%. With a higher preprocessing ratio γ of 0.38, computational complexity decreased from 4135.70M to 1340.37M, which pruned 67.59%, parameters decreased from 25.50M to 10.82M, which pruned 57.57%, while maintaining an accuracy that is 0.64% higher than ABCPruner [10]. The superior performance of this highlights FNP in both compression effectiveness and inference efficiency.

Compared with the time consumption achieved by other SOTA AutoML-based methods, our FNP approach significantly reduces the time expenditure by an impressive 95.56%, with a GPU time of only 1.84 hours. Compared with the 41.48 hours reported in MetaPruning [11], which requires evaluating multiple pruned networks on validation data using predicted weights, our proxy-based GA completes the search in significantly less time, requiring only 20 iterations. Even when compared with the most time-intensive ABCPruner [10] techniques, our method demonstrates a remarkable 88.08% reduction in duration. Despite this efficiency, the accuracy of our FNP networks remains superior to those employing a similar compression ratio, underscoring the balance between speed and performance that our method achieves.

4.4. Ablation Study

In this section, we discuss the effectiveness of the proposed IM method and SoF method, which is part of the FGA approach, as well as the proposed KNN-based fine-tuning methods. We evaluate them on ResNet-50 using ImageNet, comparing the performance of three variants in a pruning ablation study under the same initial compression ratio. The traditional GA serves as a baseline, following a standard evolutionary optimization process without involving any additional operations. To enhance the effectiveness of pruning, we propose GA with IM, which incorporates a preprocessing step that eliminates redundant or ineffective structures before the search. Building upon this, GA with IM and SoF further refines the searching decision by introducing a variant computation model for scoring, thereby accelerating the selection process.

4.4.1. Effectiveness of IM

We investigated whether incorporating preprocessing could enhance the subsequent GA-NAS process, rather than directly applying GA-NAS to compose the network. We compare the performance between the PrunedNet with and without channel prediction on ImageNet, improving accuracy by 1.04%, and evaluate the accuracy with the preprocessing generated by these two PrunedNets. As shown in Table 8, the PrunedNet without preprocessing prediction achieves lower accuracy. It is obvious that selecting a better pruning ratio of the model in advance has a great influence on searching for the optimal PrunedNet $\mathcal{N}(\mathcal{V}_p)$, and also shows that the effectiveness of the model has a great correlation with the setting of the pruning ratio of each layer. Furthermore, compared with the search process without preprocessing, we have achieved a 51% reduction in time, from 41.48 hours to 15.00 hours.

Table 8
the Effectiveness of Each Component.

Algorithm Components			Results	
FGA		IM		
GA	SoF		Accuracy	Search Time
✓	✗	✗	73.40%	41.48 hours
✓	✗	✓	74.44%	15.00 hours
✓	✓	✓	**74.48%**	**1.84 hours**

Table 9
Kendall's Tau (KT) and Spearman's Rank Correlation (SPR) on NAS-Bench-201.

Method	CIFAR-10		ImageNet	
	KT	SPR	KT	SPR
Grad_norm	0.46	0.63	0.43	0.58
SNIP	0.46	0.63	0.43	0.58
GraSP	0.37	0.54	0.40	0.56
Fisher	0.40	0.55	0.37	0.50
Synflow	0.54	0.73	0.56	0.75
Zen-score	0.29	0.38	0.29	0.40
FLOPs	0.54	0.73	0.49	0.67
#Params	0.57	0.75	0.52	0.69
SoF (Ours)	**0.60**	**0.81**	**0.60**	**0.75**

4.4.2. Effectiveness of the SoF

The results in Table 8 indicate that preprocessing significantly improves search efficiency and search time reduction. Incorporating the SoF further enhances accuracy by 0.04%. Compared with the GA with IM, we have achieved

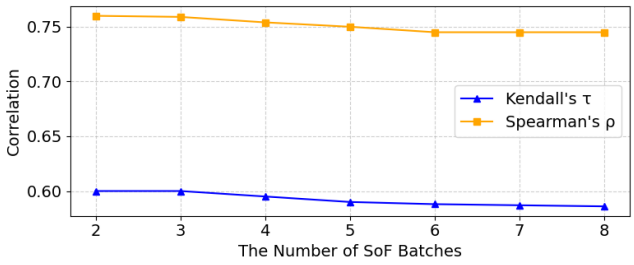

Figure 5: The correlation coefficients of different SoF batch sizes on NAS-Bench-201.

an additional reduction in time. Moreover, compared with the traditional GA, the search time is reduced by 95.56%, suggesting that our method produces better-optimized network structures. The combination of preprocessing and the modified scoring model achieves the best balance between accuracy and efficiency. In summary, Table 8 shows the preprocessing stage acts as an efficient early filtering mechanism, reducing the number of candidate architectures and improving computational efficiency. The scoring mechanism accelerates the genetic selection process, leading to higher-quality architectures in fewer iterations.

To further illustrate the practical role of SoF and its correlation with accuracy, we conducted additional experiments. The results in Table 9 demonstrate that SoF exhibits a strong correlation with accuracy and consistently achieves better proxy performance compared to prior approaches. Notably, SoF is the only proxy that performs more reliably than #Params and is generally the best proxy across different settings. Although the correlations of SoF and #Params appear similar on small-scale datasets, SoF significantly outperforms naive baselines such as #Params on large-scale datasets like ImageNet. In summary, SoF proves to be a highly effective proxy for Zero-Shot NAS.

In Section 3.3.2, we set the default batch number for SoF to 2. To further investigate the impact of the batch size on correlation and the search process, we analyze the correlation under different batch settings on NAS-Bench-201 and validate the pruned ResNet-50 models obtained from the search on ImageNet. As shown in Figure 5, the correlation gradually decreases as the batch size increases. A larger batch introduces stronger noise-smoothing effects during proxy evaluation, which reduces the sensitivity between single evaluation results and true performance, thereby weakening the correlation with final accuracy. At the same time, Table 10 shows that although the final model accuracy fluctuates only slightly, the search time increases significantly with larger batch sizes. Therefore, to strike a balance between efficiency and performance, we finally chose a batch size of 2 for SoF during the search to obtain the optimal PrunedNet $\mathcal{N}(\mathcal{V}_p)$.

Table 10
The effect of diffent batch sizes in the SoF when evaluating ResNet-50 on ImageNet.

Batch size	**2**	**4**	**6**	**8**
Search time	0.65 hours	1.18 hours	2.01 hours	2.20 hours
Accuracy	74.50%	74.41%	74.20%	74.21%

Table 11
The Same PR of ResNet-56 on CIFAR-10.

METHODS	**Top-1**	**Parameters**	**Pruned**
Human-0.7	90.72%	0.27M	68.1%
CLR-RNF-0.7[47]	92.32%	0.26M	69.4%
FNP-0.7	**93.10%**	**0.25M**	**70.6%**
Human-0.56	93.17%	0.49M	42.4%
CLR-RNF-0.56[47]	93.11%	0.38M	55.5%
FNP-0.56	**93.65%**	**0.35M**	**58.9%**

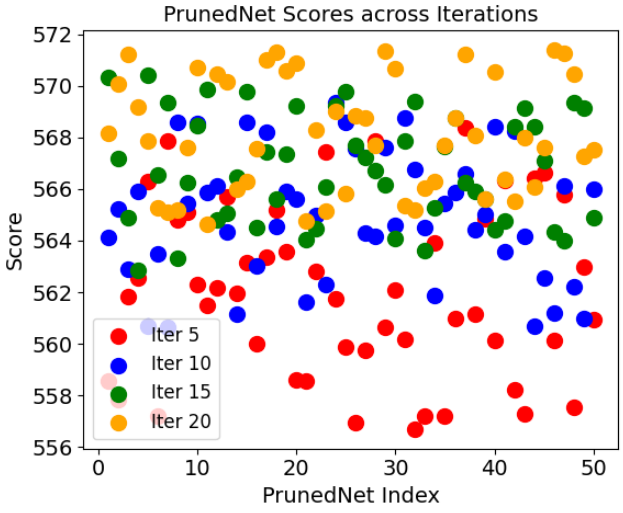

Figure 6: The visualization process of searching.

4.5. Compression Effect of FGA

To investigate the random perturbation mechanism in the FGA process, we consider three different pruning approaches: Human, which corresponds to pure random search; CLRRNF [47], which does not involve a search procedure; and FNP, which explicitly incorporates the random perturbation mechanism of FGA. Table 11 presents a comparison of these three different pruning methods applied to ResNet-56 on CIFAR-10 under the same initial global pruning ratio. Experimental results indicate that our approach reliably achieves a more optimal trade-off between accuracy and compression efficiency. Under a preprocessing ratio γ of 0.7, FNP-0.7 achieves the highest Top-1 accuracy at 93.10%, surpassing both Human-0.7 and CLR-RNF-0.7. Similarly, with a preprocessing ratio γ of 0.56, FNP-0.56 achieves the highest accuracy of 93.65%, while retaining only 0.35M parameters, outperforming other approaches in performance and compactness. These findings highlight the effectiveness

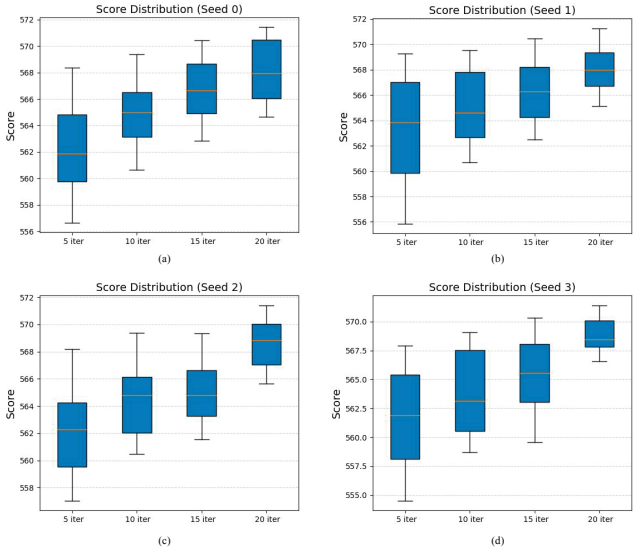

Figure 7: The score distribution of searching.

of FNP in preserving accuracy while improving model compression, largely due to the contribution of the random perturbation mechanism, making it a superior choice for efficient deep learning models.

To further explore the diversity and convergence of the random perturbation mechanism, we first illustrate in Figure 6, the score distributions of 50 candidate models sampled in four representative iterations of the search process: the 5^{th}, 10^{th}, 15^{th}, and 20^{th} iterations. Different colors denote different iterations (red for 5, blue for 10, green for 15, and orange for 20). The x-axis corresponds to the PrunedNets index, while the y-axis represents the evaluation score. As the search progresses, the sampled models cover a wider range of scores, providing richer candidates and increasing the likelihood of identifying higher-performing architectures. Furthermore, Figure 7 presents the score distributions across four random seeds to validate the convergence of the random perturbation mechanism. In all cases, as the number of iterations increases, the variance of the scores gradually decreases, and the median score shifts upward, indicating a clear convergence trend. This shows that the proposed random perturbation mechanism promotes exploration by generating diverse candidates and ensures stable convergence across different runs.

4.6. Effectiveness of KNN

Table 12 reports the effect of different weight selection strategies for fine-tuning on multiple CNNs. During fine-tuning, the methods select weights to be retained according to different criteria, including K-means clustering, ℓ_1 normalization, and random selection. Compared with these conventional fine-tuning strategies, our proposed KNN-based method consistently achieves higher accuracy in all tested architectures. For example, on GoogLeNet and ResNet-56, KNN obtains 95.03% and 93.92%, outperforming the other methods. Unlike conventional approaches that assess the importance of individual filters, our method evaluates the collective significance of filter subsets, prioritizing the most representative filters within the overall network architecture. This design makes the weight selection process more effective, reduces the duration of fine-tuning, and enables the pruned network to quickly recover the accuracy of the original model.

Table 12
The effect of different weight selection strategies for fine-tuning.

Methods	KNN	K-means	L1	Random
VGGNet-16	93.67%	92.50%	93.30%	92.60%
ResNet-56	93.92%	92.80%	93.00%	92.50%
ResNet-110	93.95%	92.50%	93.00%	92.60%
GoogLeNet	95.02%	94.10%	94.40%	94.10%
ResNet-50	75.10%	74.20%	74.30%	73.60%

5. Conclusion and Future Work

In conclusion, we propose FNP, a novel method that enables highly efficient and effective model compression. It substantially reduces model size and computation while maintaining accuracy. Unlike previous approaches that rely on fixed pruning ratios and manual tuning, our method introduces an IM in the preprocessing stage to automatically estimate layer-wise pruning ratios. This significantly accelerates the pruning process. To address the heavy reliance on manual expertise in pruning strategies, we propose FGA, a fine-grained adaptive local search method guided by the SoF proxy metrics to efficiently evaluate and select optimal pruned networks. Extensive experiments and ablation studies show that FNP achieves a superior trade-off between compression, accuracy, and efficiency. Compared with conventional pruning and AutoML-based approaches, it delivers competitive or better results with drastically reduced time costs. In future work, our goal is to accelerate accuracy recovery during fine-tuning and broaden the applicability of FNP to a wider range of neural network architectures. We also plan to further explore the design and effectiveness of the Importance Matrix (IM), as well as extend our framework to more complex tasks such as object detection and semantic segmentation, where pruning plays a crucial role in balancing efficiency and accuracy. In future work, our goal is to accelerate the recovery of accuracy during fine-tuning and expand the applicability of FNP to a wider range of neural network architectures.

References

[1] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, D. Kalenichenko, Quantization and training of neural networks for efficient integer-arithmetic-only inference, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 2704–2713.

[2] G. Fang, X. Ma, M. Song, M. Bi Mi, X. Wang, DepGraph: Towards Any Structural Pruning, in: 2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2023, pp. 16091–16101.

[3] G. Hinton, O. Vinyals, J. Dean, Distilling the knowledge in a neural network, arXiv:1503.02531 (2015).

[4] S. Han, H. Mao, W. J. Dally, Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding, in: International Conference on Learning Representations (ICLR), 2016.

[5] J.-H. Luo, J. Wu, W. Lin, ThiNet: A Filter Level Pruning Method for Deep Neural Network Compression, in: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 5058–5066.

[6] Y. He, X. Zhang, J. Sun, Channel Pruning for Accelerating Very Deep Neural Networks, in: IEEE International Conference on Computer Vision (ICCV), 2017, pp. 1389–1397.

[7] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning Efficient Convolutional Networks through Network Slimming, in: 2017 IEEE International Conference on Computer Vision (ICCV), 2017, pp. 2755–2763.

[8] P. Molchanov, A. Mallya, S. Tyree, I. Frosio, J. Kautz, Importance Estimation for Neural Network Pruning, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 11264–11272.

[9] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, HRank: Filter Pruning Using High-Rank Feature Map, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 1526–1535.

[10] M. Lin, R. Ji, Y. Zhang, B. Zhang, Y. Wu, Y. Tian, Channel Pruning via Automatic Structure Search, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2020, pp. 673 – 679.

[11] Z. Liu, H. Mu, X. Zhang, Z. Guo, X. Yang, T. K. T. Cheng, J. Sun, MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning, in: IEEE International Conference on Computer Vision (ICCV), 2019, pp. 3295–3304.

[12] W. Wang, V. W. Zheng, H. Yu, A Survey of Zero-Shot Learning, IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI) 42 (2020) 2031–2046.

[13] Y. Xian, C. H. Lampert, B. Schiele, Zero-Shot Learning:A Comprehensive Evaluation of the Good, the Bad and the Ugly, in: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2018, pp. 4582–4591.

[14] Y. He, L. Xiao, Structured Pruning for Deep Convolutional Neural Networks: A Survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 46 (2023) 2900–2919.

[15] H. Cheng, M. Zhang, J. Q. Shi, A Survey on Deep Neural Network Pruning: Taxonomy, Comparison, Analysis, and Recommendations, IEEE Transactions on Pattern Analysis and Machine Intelligence 46 (12) (2024) 10558–10578.

[16] H. Li, A. Kadav, I. Durdanovic, H. Samet, H. P. Graf, Pruning Filters for Efficient ConvNets (2017). arXiv:1608.08710.

[17] N. Lee, T. Ajanthan, P. H. S. Torr, SNIP: Single-shot Network Pruning based on Connection Sensitivity, ArXiv abs/1810.02340 (2018).

[18] H. Cheng, M. Zhang, J. Q. Shi, Influence Function Based Second-Order Channel Pruning: Evaluating True Loss Changes for Pruning is Possible Without Retraining, IEEE Transactions on Pattern Analysis and Machine Intelligence 46 (12) (2024) 9023–9037.

[19] J. Frankle, M. Carbin, The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks, arXiv:1803.03635 (2018).

[20] H. Xing, J. Xiang, L. Xiong, Q. Wen, Q. Liu, Y. Wang, A Sparse Sharing Multitask Framework for Building Footprint Extraction From Remote Sensing Imagery Following the Dual Lottery Ticket Hypothesis, IEEE Transactions on Geoscience and Remote Sensing 62 (2024) 1–16.

[21] S. Guo, Y. Wang, Q. Li, J. Yan, DMCP: Differentiable Markov Channel Pruning for Neural Networks, in: 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020, pp. 1536–1544.

[22] W. Xu, Y. Xian, J. Wang, B. Schiele, Z. Akata, Attribute prototype network for zero-shot learning (2021). arXiv:2008.08290.

[23] J. Guo, Learning Robust Visual-semantic Mapping for Zero-shot Learning (2021). arXiv:2104.05668.

[24] G. Ying, X. He, B. Gao, B. Han, X. Chu, EAGAN: Efficient Two-Stage Evolutionary Architecture Search for GANs, in: S. Avidan, G. Brostow, M. Cissé, G. M. Farinella, T. Hassner (Eds.), Computer Vision – ECCV 2022, 2022, pp. 37–53.

[25] Z. Han, Z. Fu, S. Chen, J. Yang, Contrastive embedding for generalized zero-shot learning (2021). arXiv:2103.16173.

[26] J. Lee, B. Ham, AZ-NAS: Assembling Zero-Cost Proxies for Network Architecture Search, in: 2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024, pp. 5893–5903.

[27] F. Hutter, L. Kotthoff, J. Vanschoren, Automated Machine Learning: Methods, Systems, Challenges, Springer, 2019.

[28] X. He, K. Zhao, X. Chu, AutoML: A Survey of the State-of-the-Art, ACM Computing Surveys (CSUR) 54 (8) (2021) 1–36.

[29] B. Zoph, Q. V. Le, Neural Architecture Search with Reinforcement Learning, in: International Conference on Learning Representations (ICLR), 2017.

[30] H. Liu, K. Simonyan, Y. Yang, DARTS: Differentiable Architecture Search, in: International Conference on Learning Representations (ICLR), 2019.

[31] A. Vo, N. H. Luong, Efficient Multi-Objective Neural Architecture Search via Pareto Dominance-based Novelty Search, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2024, p. 11461155.

[32] H. Ye, Y. Duan, M. Tan, Y. Chen, F. Wu, X. Wang, BigNAS: Scaling Up Neural Architecture Search with Big Single-Stage Models, in: European Conference on Computer Vision (ECCV), 2022.

[33] Y. Sun, Y. Xue, M. Zhang, Evolving Deep Convolutional Neural Networks by Genetic Algorithm for Image Classification, in: Genetic and Evolutionary Computation Conference (GECCO), 2019, pp. 256–264.

[34] L. Lin, S. Chen, Y. Yang, Z. Guo, AACP: Model Compression by Accurate and Automatic Channel Pruning, in: 2022 26th International Conference on Pattern Recognition (ICPR), 2022, pp. 2049–2055.

[35] X. Dong, Y. Yang, Network Pruning via Transformable Architecture Search, in: Neural Information Processing Systems, 2019.

[36] Z. Li, G. Yuan, W. Niu, P. Zhao, Y. Li, Y. Cai, X. Shen, Z. Zhan, Z. Kong, Q. Jin, Z. Chen, S. Liu, K. Yang, B. Ren, Y. Wang, X. Lin, NPAS: A Compiler-aware Framework of Unified Network Pruning and Architecture Search for Beyond Real-Time Mobile Acceleration, in: 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021, pp. 14250–14261.

[37] C. Wang, C. Wang, G. Zhang, R. B. Grosse, Picking Winning Tickets Before Training by Preserving Gradient Flow, Vol. abs/2002.07376, 2020.

[38] H. Tanaka, D. Kunin, D. L. K. Yamins, S. Ganguli, Pruning neural networks without any data by iteratively conserving synaptic flow (2020). arXiv:2006.05467.

[39] L. Liu, S. Zhang, Z. Kuang, A. Zhou, J. Xue, X. Wang, Y. Chen, W. Yang, Q. Liao, W. Zhang, Group Fisher Pruning for Practical Network Compression, in: International Conference on Machine Learning, 2021.

[40] G. Li, Y. Yang, K. Bhardwaj, R. Marculescu, ZiCo:Zero-shot NAS via Inverse Coefficient of Variation on Gradients, ArXiv abs/2301.11300 (2023).

[41] J. S. Bergstra, Y. Bengio, Algorithms for Hyper-Parameter Optimization, Journal of Machine Learning Research 13 (2012) 281–305.

[42] D. Whitley, A Genetic Algorithm Tutorial, Statistics and Computing 4 (2) (1994) 65–85.

[43] T. Hospedales, A. Antoniou, P. Micaelli, A. Storkey, Meta-Learning in Neural Networks: A Survey, IEEE Transactions on Pattern Analysis and Machine Intelligence 44 (9) (2022) 5149–5169.

[44] S. J. Pan, Q. Yang, A Survey on Transfer Learning, IEEE Transactions on Knowledge and Data Engineering 22 (10) (2010) 1345–1359.

[45] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, F. Hutter, Efficient and Robust Automated Machine Learning, Advances in Neural Information Processing Systems (2015) 2962–2970.

[46] E. Real, C. Liang, D. R. So, Q. V. Le, Automl-zero: Evolving machine learning algorithms from scratch (2020). arXiv:2003.03384.
[47] M. Lin, L. Cao, Y. Zhang, L. Shao, C.-W. Lin, R. Ji, Pruning Networks With Cross-Layer Ranking k-Reciprocal Nearest Filters, IEEE Transactions on Neural Networks and Learning Systems 34 (11) (2023) 9139–9148.
[48] X. Sun, H. Shi, Towards Better Structured Pruning Saliency by Reorganizing Convolution, 2024 IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) (2024) 2193–2203.
[49] S. Lin, R. Ji, C. Yan, B. Zhang, L. Cao, Q. Ye, F. Huang, D. Doermann, Towards Optimal Structured CNN Pruning via Generative Adversarial Learning, in: 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2019, pp. 2785–2794.
[50] Ambuj, Feature representation fidelity preservation during neural network pruning for enhanced compression efficiency, Neurocomputing 634 (2025) 129901.
[51] Y. Sui, M. Yin, Y. Xie, H. Phan, S. Zonouz, B. Yuan, CHIP: CHannel Independence-based Pruning for Compact Neural Networks, in: Neural Information Processing Systems, 2021.
[52] D. Jiang, Y. Cao, Q. Yang, On the Channel Pruning using Graph Convolution Network for Convolutional Neural Network Acceleration , in: L. D. Raedt (Ed.), Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22, International Joint Conferences on Artificial Intelligence Organization, 2022, pp. 3107–3113, main Track.
[53] W. Huang, X. Wang, Z. Zhao, L. Su, S. Sui, J. Wang, Pruning CNN based on Combinational Filter Deletion, in: IECON 2024 - 50th Annual Conference of the IEEE Industrial Electronics Society, 2024, pp. 1–6.
[54] H. Shang, J.-L. Wu, W. Hong, C. Qian, Neural Network Pruning by Cooperative Coevolution, ArXiv abs/2204.05639 (2022).
[55] Y. Guan, N. Liu, P. Zhao, Z. Che, K. Bian, Y. Wang, J. Tang, DAIS: Automatic Channel Pruning via Differentiable Annealing Indicator Search, IEEE Transactions on Neural Networks and Learning Systems 34 (12) (2023) 9847–9858.
[56] Y. He, P. Liu, L. Zhu, Y. Yang, Filter Pruning by Switching to Neighboring CNNs With Good Attributes, IEEE Transactions on Neural Networks and Learning Systems 34 (10) (2023) 8044–8056.

Table 13
Other hyperparameter settings.

Parameter	Key	Value
the λ of VGG-16	λ_1	0.5
the λ of ResNet-56	λ_2	10
the λ of ResNet-110	λ_3	5
the λ of GoogLeNet	λ_4	20
the λ of ResNet-50	λ_5	0.4

Table 14
Detailed hyperparameter settings for FGA.

Parameter	Key	Value
Iter	n	20
Iteration Gene	s	50
Mutation	$s/2$	25
Crossover	$s/2$	25
Perturb Factor	F	4
Population Size	g	20*50

A. Detailed Hyperparameter Settings for FNP

Before we introduce the detailed settings, we note that different models employ different λ settings on the preprocessing stage, as shown in Table 13. The detailed search settings of FGA are provided in Table 14. For other implementation details and remaining hyperparameters, please refer to our released code.