Compression of deep neural networks: bridging the gap between conventional-based pruning and evolutionary approach

Yidan Zhang¹ · Guangjin Wang² · Taibo Yang² · Tianfeng Pang² · Zhenan He¹ · Jiancheng Lv¹

Received: 18 August 2021 / Accepted: 2 March 2022 / Published online: 9 June 2022
© The Author(s) 2022

Abstract
Recently, many studies have been carried out on model compression to handle the high computational cost and high memory footprint brought by the implementation of deep neural networks. In this paper, model compression of convolutional neural networks is constructed as a multiobjective optimization problem with two conflicting objectives, reducing the model size and improving the performance. A novel structured pruning method called Conventional-based and Evolutionary Approaches Guided Multiobjective Pruning (CEA-MOP) is proposed to address this problem, where the power of conventional pruning methods is effectively exploited for the evolutionary process. A delicate balance in pruning rate and model accuracy has been automated achieved by a multiobjective optimization evolutionary model. First, an ensemble framework integrates pruning metrics to establish a codebook for further evolutionary operations. Then, an efficient coding method is developed to shorten the length of chromosome, thus ensuring its superior scalability. Finally, sensitivity analysis is automatically carried out to determine the upper bound of pruning rate for each layer. Notably, on CIFAR-10, CEA-MOP reduces more than 50% FLOPs on ResNet-110 and improves the relative accuracy. Moreover, on ImageNet, CEA-MOP reduces more than 50% FLOPs on ResNet-101 with negligible top-1 accuracy drop.

Keywords Deep neural networks · Evolutionary algorithm · Filter pruning · Multiobjective optimization

1 Introduction
Deep neural networks (DNNs) have achieved remarkable progress on image processing [10], speech recognition [7], machine translation [53], and other complex tasks, owing to their superior learning ability brought from the inherent deep structure. However, it also comes with issues such as over-parameterization, where the massive computational cost and high memory footprint have severely restricted the implementation of DNNs on real-world problems. Naturally, model compression [3] comes into consideration. It allows the deep models to run efficiently on resource-constrained devices by significantly reducing the computational load and the storage of the model while maintaining and even improving the performance [12, 18, 29].

As illustrated in Fig. 1, conventional pruning algorithms, e.g., pruning based on the average percentage of zeros [16] in the output and Taylor expansion [33] approximating the impact of each channel on loss, consist of three steps, training, pruning, and fine-tuning. The iterative process continues until the best tradeoff between parameters reduction and performance optimization is achieved. However, the process of iterative pruning and fine-tuning is time-consuming and energy-wasting. Furthermore, these methods rely on domain experts to manually specify pruning criteria and sparsity ratio, which cannot guarantee stable optimal pruning performance. As a result, researchers turn to find effective ways for automatic configuration of these parameters. For example, Neural Architecture Search (NAS) has attracted the attention of the deep learning research community given its performance on automatic architecture design [41, 56]. As one type of NAS, automatically structured pruning determines
the reduced structure of DNNs, e.g., the number of filters, channels, and neurons in each layer [30]. Due to their heuristic characteristics and population-based framework, Evolutionary Algorithms (EAs) [2] have established themselves as effective means in solving automatically structured pruning problems. In previous works, EAs greatly improve the performance of deep neural networks through an iterative search of model structure [42, 49, 54]. In the literature, these works mainly focus on improving network performance and EAs are formulated to address single-objective optimization.

The basic idea for structured pruning is removing structured weights, including 2D kernels, filters, channels or layers, which are not sensitive to performance [26, 39]. Nevertheless, excessive parameter reduction will unavoidably lead to model performance degradation [62, 63]. Thus, reducing the scale of parameters and improving the performance of the network are considered two conflicting objectives. In view of this, model pruning can be regarded as a multiobjective optimization problem (MOP) [40]. MOP refers to an optimization problem involving two or more objectives, all of which cannot be optimal simultaneously. Particularly, in MOPs, multiobjective optimization evolutionary algorithms (MOEAs) [5, 59] provide powerful search ability for exploring both converged and diversified Pareto front (PF), where improvement in one objective leads to a degradation in at least one other objective. PF is the collection of objectives of a set of Pareto solutions that are non-dominated with respect to each other [34].

Using MOEAs on structured pruning problems has made some progress lately, which can efficiently search for the Pareto optimal solutions with a delicate balance between model size and performance [51, 55, 62, 63]. In the early study, filter pruning is formally established as an MOP [62]. Then, a set of novel genetic operators are proposed to ease the burden of manually layer-wise tuning [63]. Recently, sparse learning is combined with genetic algorithms to achieve network channel pruning [51]. In [55], it focuses on improving the genetic algorithm to achieve better solutions in the search space. However, the commonly used binary coding approach inadvertently encodes the filters, resulting in the length of the chromosome equal to the number of filters. For instance, on the award-winning VGG16 with 4,224 filters (convolutional layers) and 9,192 neurons (fully connected layers), the chromosome length is 13,416 bits (4,224+9,192). Thus, tens of thousands of filters embedded in CNNs result in the unexpected long chromosome, the exponentially increased search space brings the difficulty in converging to the Pareto optimal front during the evolutionary search. Furthermore, the evolutionary process always starts with a randomly initialized population, and the quality of the initial population directly affects the search efficiency of EAs. However, there remains a lack of an effective way to ensure the quality of the initial population.

In general, both conventional- and EA-based pruning methods have their own advantages. The former can avoid the difficulty from the expanding search space and achieve a faster result, while the latter has shown promising performance on automatically structured pruning due to their capabilities on gradient-free optimization and massive parallelization. However, they also suffer from some deficiencies. For example, conventional pruning techniques rely on domain experts to manually specify pruning criteria and sparsity ratio, which inadvertently cannot guarantee stable optimal pruning performance. Furthermore, they adopt iterative pruning and fine-tuning process to achieve the balance between parameters reduction and performance optimization, which is time-consuming and energy-wasting. On the other hand, EA-based pruning methods lack effective means to ensure the quality of the initial population and have difficulty in finding the global optimal solution in a large search space.

To our best knowledge, in the literature, there is no research work considering exploiting the power from both types of methods on the pruning problems. Motivated by this, we devote this paper to bridging the gap between conventional-based pruning and evolutionary approach. By fully exploring the power of conventional pruning methods for evolutionary algorithms, we propose a novel structured pruning method, named CEA-MOP, for multiobjective CNN compression problems with two conflict objectives, reducing the scale of parameters and improving the performance of the network. To achieve this goal, we first propose an ensemble framework in which conventional pruning methods can collaboratively establish a model pool with various fine-pruned models, thus forming a codebook for further evolutionary operations. Afterward, the model pool is encoded as a quality initial population for MOEAs, which helps to improve their search efficiency. Then, by performing our proposed encoding and decoding method,
the search efficiency is further improved and sensitivity analysis on each layer of the model is automatically carried out. Thus, the gap between conventional pruning methods and evolutionary algorithms is bridged. In summary, there are three main contributions made in this paper.

- We develop an ensemble framework that can integrate any model pruning metrics to establish a codebook for further evolutionary operations. The framework overcomes the disadvantage of a single metric that focuses on problem-specific characteristics and only works on special models.
- We design an efficient encoding method to shorten the length of the chromosome, in which one chromosome represents a model structure with each gene representing a set of filters, thus achieving dimensionality reduction and pushing the population toward the Pareto optimal front as much as possible.
- Our proposed MOEA automatically carries out sensitivity analysis on each layer of the model and determines the upper bound of pruning rate for each layer according to its sensitivity degree, which constrains the search direction during the evolutionary process and guides the search towards the target region, thus improving the algorithm’s search efficiency and reducing its computational load.

The remaining sections complete the presentation of this paper. Section 2 provides a literature review about existing researches on model compression and multiobjective optimization. Our proposed structured pruning method and its details are explained in Sect. 3. In Sect. 4, we elaborate on the experimental results given selected benchmark functions. Finally, the conclusion is drawn in Sect. 5 along with pertinent observations.

2 Related work

In this section, two types of model compression approaches are described in detail, including conventional convolutional neural network (CNN) compression and evolutionary algorithms assisted compression.

2.1 Conventional convolutional neural network compression

Remarkable progress has been made on model compression of CNNs [3], which simultaneously reduces the number of network parameters and computation overhead while maintaining and even improving network performance. Through model compression, the deep CNN model can run efficiently on resource-constrained devices.

In the literature, there are five categories of model compression approaches. The first is matrix decomposition [20, 52] where the weight matrix of the model is treated as a full rank matrix and several low rank matrices are used to approximate the original matrix so as to achieve acceleration and compression. However, matrix decomposition involves expensive calculation and complex implementation, and the current method analyzes the importance of parameters layer by layer, which makes it impossible to perform global parameter compression. The second type is quantization [19, 44], which compresses parameter storage by reducing the number of bits required to represent each weight. Although weight sharing and Huffman coding can be applied on the quantized weights for further compression and acceleration [44], parameter quantization still results in appreciable loss of precision. The third category is knowledge distilling (KD) [14, 57]. Based on transfer learning, KD trains an alternative simple network (called Student model) using the output of the pre-trained complex model (called Teacher model) as the supervision signal. KD-based approaches can only work on classification tasks with softmax loss function, which limits its usability. The fourth group is compact network design [15, 61]. Rather than compressing existing large networks, it directly designs alternative smaller and more compact networks in the initial stage of model construction. The model based on this type of method has fewer parameters and requires less computation, which is widely used in embedded platforms. However, this method can only be applied to the convolutional layer. The last group is network pruning [9, 25, 26, 39], which aims at removing redundant parameters such as weights, kernels, filters, and layers to reduce storage resources and computational complexity. Pruning-based methods have attracted much attention because they are robust to various settings, and can achieve good compression performance and rate on both scratch and pre-trained models.

One major branch of network pruning methods is unstructured pruning originated from [9, 22], which prunes redundant weights based on the Hessian value of a loss function. By iterative pruning and retraining, recent work proposed in [8] removes all connections whose weights are lower than a given threshold. In [23], the unsupervised pruning can work well on models for image classification tasks. However, unstructured pruning will lead to non-structured sparsity in the network. Unless supported by dedicated hardware and libraries, compression and acceleration cannot be achieved in unstructured pruning.

On the other hand, structured pruning retains the original convolution structure, and its performance can be further improved using additional techniques, e.g., low-rank approximation and quantization. The operation of convolutional filter pruning to eliminate redundancy of
parameters in CNN architecture works as follows. Given a CNN with \( L \) convolutional layers, let \( N_i \) denotes the number of filters in the \( i \)th convolutional layer, \( i \in \{1, \ldots, L\} \), both the number of output feature maps generated by convolution operation in the \( i \)th layer and the channels of filters in the \((i+1)\)th layer are equal to \( N_i \). We represent the filters of each convolutional layer as \( F = \{F_1, F_2, \ldots, F_L\} \), where \( F_i = \{F^1_i, F^2_i, \ldots, F^{N_i}_i\} \) and \( F^j_i \) represents the \( j \)th filters at layer \( i \). Assuming that \( K \) filters are removed in the \( i \)th layer, then the number of channels for the feature maps and the corresponding channels of the filters in the next layer is reduced to \( N_i - K \) accordingly. The pruning process is shown in Fig. 2.

It is formulated as a combinatorial optimization problem to determine which filters to be pruned without performance degradation. We can express it as follows:

\[
\min_F L(D; F) + \lambda ||F||_0, \quad (1)
\]

where \( L(D; F) \) represents the loss function of the CNN on dataset \( D \) and it could be Euclidean or softmax losses \([24]\) selected independently of pruning. The operator \( ||.||_0 \) in Eq. 1 is the \( l_0 \)-norm, which makes model pruning an NP-hard problem. As the network becomes deeper (most CNNs have more than 1000 filters), it becomes infeasible to examine all possible combinations by conventional greedy methods.

For example, the early popular magnitude-based weight pruning method prunes filters based on their corresponding absolute weights \([25]\) and the average percentage of zeros \([16]\) in the output. However, this approach relies heavily on predefined pruning rates and criteria, which lacks a guarantee on the compression rate and performance of the pruned model. The reconstruction error-based pruning \([11]\) determines which channels to be removed by minimizing feature reconstruction error of the next layer, which requires complicated manual hyper-parameter tuning to achieve a balance between model performance and model size. In \([33]\), Taylor expansion is proposed to approximate the impact of each channel on loss and prunes accordingly to ensure performance. These methods discussed above can be roughly summarized as greedy and rule-based pruning, also known as “saliency-based” pruning \([25]\). They greatly reduce the search space through the pre-defined compression policy, thus improving the search speed. However, they can only find local optimal solutions and heavily rely on the relative importance of parameters, which does not always exist in real-world applications.

Another kind of pruning algorithm focuses on the mutual information (MI) between parameters to find the global optimal solution. For example, a collaborative channel pruning (CCP) algorithm utilizes the inter-channel relationship to determine reserved channels \([36]\). Furthermore, in \([31]\), the pruning problem is approximated as a differentiable objective and solved by gradient-based methods. In \([48]\), a method based on subspace clustering has been proposed to compress filters. Most above approaches follow the process depicted in Fig. 1, which achieves desired compression performance through relatively time-consuming multi-stage optimization. However, as argued in \([30]\), a pruned model trained using randomly initialized weights performs better than the same model after fine-tuning. A recent study in \([50]\) also shows that more diverse candidate structures can be directly pruned from randomly initialized weights.

### 2.2 Evolutionary algorithms assisted compression

EA \([2]\) is a type of metaheuristic method inspired by biological evolution, which is widely utilized on NP-hard problems through genetic operations, such as mutation, crossover, and selection. Applying EAs to model compression problems can automatically eliminate redundant parameters avoiding excessive hyper-parameters tuning and can restore structures that have been pruned. The success largely rests on two crucial components, a genetic representation of the solution domain and a fitness function to evaluate each individual. In \([49]\), it represents each
compressed network as a binary individual and introduces a fitness function controlling the tradeoff between compression rate and performance degradation. In [45], it focuses on an efficient coding method to evolve appropriate network structures and parameters for tasks with a large number of parameters. The results show that the single-objective optimization using EAs can compress the neural network effectively, but face difficulty in ensuring accuracy simultaneously.

Different from the above single-objective algorithm, MOEAs can optimize multiple conflicting objectives in a single run. In recent years, some attempts have been made on the multiobjective optimization model for network pruning, in which the conflicting objectives are compress rate and network performance. The filter pruning is formally defined as an MOP and a knee-guided evolutionary algorithm (KGEA) [62] is proposed to effectively search for the solution with a quality tradeoff between model size and performance. In [63], on the basis of optimizing network loss and the number of parameters simultaneously during evolution, a novel genetic operator is designed to enable automated optimal pruning ratio discovery. Recently, [51] proposes a channel pruning method, in which the sparse learning is first utilized to generate candidate subnetworks, and then the genetic algorithm is adopted to select the optimal subnetwork from the candidates. In this study [55], several improvements toward MOEAs are proposed including combined Gaussian initialization, progressive shrinking mutation, and fine-grained crossover. The disadvantage is that the binary coding method directly encodes the filters, which results in a large search space and makes it difficult to find a solution with a delicate tradeoff between both objectives within an acceptable time.

In this paper, we devote bridging the gap between conventional-based pruning and evolutionary approach, through which, CEA-MOP, a novel pruning method for compressing deep CNNs is proposed. CEA-MOP inherits the advantages and overcomes the disadvantages of both methods. Compared with conventional pruning, the proposed method not only avoids iterative pruning and fine-tuning but also requires no sensitivity analysis on each layer to determine the pruning rate. In contrast to evolutionary pruning, the quality of the initial population is guaranteed and the search efficiency is greatly improved.

### 3 Method

In this section, the details of the proposed CEA-MOP for CNN compressing are presented. First, the filter pruning problem is modeled (3.1). Besides, the overall process of CEA-MOP is introduced (3.2). Then, the ensemble framework integrating four different model pruning metrics is given (3.3). After that, the encoding and decoding approaches are discussed (3.4). Furthermore, the process of obtaining the optimal solution for compressing CNNs via MOEA is explained (3.5).

#### 3.1 Problem modeling

Given a CNN $C$ with $L$ convolutional layers, $F = \{F_1, F_2, F_3, \ldots, F_L\}$ denotes the filter set, where $F_i = \{F_{i1}, F_{i2}, \ldots, F_{iN_i}\}$ includes all filters on the $i$th layer and $N_i$ represents the number of filters on the $i$th layer. We model the proposed structured pruning approach as the following bi-objective optimization model [38, 58, 60]:

$$
\begin{align*}
& f_1 : \min_{M} \mathcal{L}(D; \mathcal{M} \circ F) \\
& f_2 : \min_{M} \sum_{i=1}^{L} ||\mathcal{M}_i||_1,
\end{align*}
$$

(2)

where $f_1$ and $f_2$ represent model performance and the number of filters, respectively. $D$ denotes the dataset used to evaluate the performance of $C$. $\mathcal{M}$ is a mask that determines whether a particular filter is included or pruned during feed-forward propagation. The size of $\mathcal{M}$ is the same as that of $F$, which is fixed in a well-trained CNN. The notation “$\circ$” is element-wise multiplication which represents the multiplication of the corresponding elements of each matrix. The operator $||.||_1$ refers to the $l_1$-norm and $||\mathcal{M}_i||_1$ calculates accumulated absolute values of $\mathcal{M}_i$ (the mask in the $i$th layer). $F_i^j$ represents the $j$th filters at layer $i$, we have:

$$
\mathcal{M}_i^j = \begin{cases} 
0, & \text{removing the filter } F_i^j \\
1, & \text{otherwise}
\end{cases}
$$

(3)

where $\mathcal{M}_i^j$ represents the mask for the $j$th filter on the $i$th convolutional layer and $\mathcal{M} \circ F$ refers to removing part of the filters from $C$ to form a pruned model.

Since excessive filter elimination will unavoidably lead to model performance degradation, we treat model pruning as an MOP in Eq. 2. In MOP, It is not possible to have a single solution that simultaneously achieves optimal on all objectives. In theory, the target for an MOP with conflicting objectives is to investigate a set of Pareto solutions, each of which is non-dominated by any other solution. A feasible solution is to explore the entire search space by MOEAs to automatically search for the Pareto optimal solutions in one single run. Finally, an optimal solution with a satisfactory tradeoff between both objectives is selected by a multicriteria decision making rule.
3.2 The overall process of CEA-MOP

By combing conventional and evolutionary pruning, CEA-MOP preserves the most important filters in each layer to achieve the above pruning objectives. The main process of CEA-MOP consists of two steps. The first step is codebook generation, which starts with a well-trained CNN \( C \). Then, an ensemble framework that integrates multiple pruning criteria is utilized to repeatedly identify the most important filters of \( C \) to form some quality masks (\( M \) in Eq. 2). Thus, a model pool with some pruned structures (\( M \circ F \) in Eq. 2) is build. Afterward, the mask of each pruned structure is encoded into a codebook \( B \) for further evolutionary operations. The second step is evolutionary pruning. First, the multiobjective optimization evolutionary process accepts the codebook as the initial population. Based on the codebook, a novel encoding scheme is designed, in which each chromosome encodes the mask of a model structure and each gene encodes a segment of a mask from the codebook. By crossover and mutation, multiple mask fragments from the codebook are recombined to form new pruned structures. Besides, the objective values for each pruned structure including the number of filters (\( f_2 \) in Eq. 2) and the classification error rate (\( f_1 \) in Eq. 2) are evaluated on the validation set without model fine-tuning. Through environmental selection, individuals with high non-domination levels [5] enters the next generation. When the evolution operation terminates, select an optimal pruned structure from a set of Pareto optimal structures.

3.3 Ensemble framework

Generally, no single pruning metric alone can faithfully eliminate the redundancy of CNNs without performance degradation [37]. Every metric focuses on some problem-specific characteristics while neglecting other information, thus can only work well on specified models. By integrating the following four pruning criteria, our proposed ensemble framework can maximize the advantages of each criterion and overcome the disadvantages of the others to form a more reliable codebook. Moreover, the ensemble framework can effectively exploit the power of conventional methods and overcome the shortcomings of evolutionary pruning, thus providing good preparation for the next stage of the evolutionary process. Please note, any newly emerging pruning metrics can also be included. In this paper, the four metrics integrated are:

(a) Absolute Weighted Sum (AWS) [25]: calculates the redundancy value of a filter by its absolute weighted sum.

(b) Average Percentage of Zeros (APoZ) [16]: calculates the redundancy value of a filter by the average percentage of zeros on its outputs.

(c) Taylor Expansion Loss (TEL) [33]: calculates the redundancy value of a filter by approximating the change in the cost function based on Taylor expansion induced by pruning it.

(d) Filter Pruning via Geometric Median (FPGM) [13]: calculates the redundancy value of a filter by the distance between it and the Geometric Median [6] of filters in the same layer.

In general, AWS and APoZ are considered “saliency-based” pruning. Since they identify the importance of filters based on pre-defined rules, while FPGM is mutual information-based pruning. Meanwhile, TEL measures the importance of filters on loss while considering the mutual information between parameter and loss can be grouped into both “saliency-based” and “mutual information based” pruning. The detailed analysis of the four pruning metrics integrated in this paper is as follows. AWS and APoZ are known as the “smaller-norm-less-important” criterion, two pre-requisites are required to utilize them as discussed in [13]. First, the deviation of filter norms should be significant. Second, the norms of those filters that can be removed should be arbitrarily small, that is, close to zero. In other words, it is expected that the contributions of filters with smaller norms to the network are absolutely small. An ideal norm distribution is expected to meet those two requirements. Unfortunately, based on some analysis and experimental observations, this is not always true. The pruning criterion of FPGM is to prune the most replaceable filters containing redundant information, which can still achieve good performance when the norm-based criterion fails. However, this approach relies heavily on predefined pruning rates for each layer. The last pruning method TEL is based on the influence of filter pruning on loss to deal with the situation where pruning criteria developed without regard to model performance are ineffective. In fact, it is suboptimal to approximate the change of the loss function based on Taylor expansion.
Although these four metrics have achieved satisfactory results on some CNN compression tasks, each of them cannot maintain good performance on other applications. For example, if a kernel is overfitted, it may lead to the failure of AWS. In addition, when there are many filters in the same convolutional layer with very small redundancy, the criterion of FPGM cannot be effective. In summary, a single pruning criterion is more likely to fail when the situation is complex and diverse, while pruning networks with multiple metrics are more likely to achieve better results than a single metric could. Therefore, we propose an ensemble framework integrating multiple pruning criteria to ensure that under any scenario there are at least one metric works well. Thus, through an ensemble framework, the shortcoming of each approach is offset and their power is exploited to the maximum degree. This framework can be applied directly to a variety of CNNs with little modification.

Algorithm 1 shows the main ensemble framework. Specifically, we train a CNN $C$ with a total of $T (T = \sum_{i=1}^{L} N_i)$ filters on dataset $D$. Given a pruning rate which is shared by all layers in $C$, calculate the number of filters $K_i$ that should be removed from the $i$th layer. In the $i$th layer, the redundancy value of the $g$th filter is calculated according to $V_j$. Select the first $K_i$ filters from $F_i$ sorted by $E_g$. Then, $M_i = \{M_1^i, M_2^i, ..., M_{N_i}^i\}$ is the mask for the $i$th layer, where $M_j^i \in \{0, 1\}$ determines the $j$th filter on the $i$th layer is included or pruned. One-Dimensional-Expansion($\mathcal{M}$) expands $M_i$ into a one-dimensional binary vector $m = [s_1, s_2, ..., s_T]$ and added to the codebook $B$. Where $s_g \in \{0, 1\}$ denotes whether the $g$th layer is included or pruned.
corresponding gth filter is masked. When R different settings of pruning rates and four pruning criteria are applied, we have a total of 4R binary vectors, each of which is a mask for pruning the filters of the original model. It should be emphasized that when integrating each mask, we simply count the filters that should be removed from the original model according to a predetermined pruning criterion and pruning rate to form a mask. More importantly, the process of forming a mask is done at once, rather through iterative pruning and fine-tuning.

For example, the original model \( C \) has two convolutional layers, each of which has five filters, then we get \( T = 10 \) and \( \mathcal{M} = \{\{11111\},\{11111\}\} \) for the original model. Given only one pruning rate, four pruning criteria AWS, APoZ, TEL, and FPGM, we can generate four binary vectors, each of which consists of 10 bits. The ensemble process is shown in Fig. 3, where the input includes the mask \( \mathcal{M} \) of a well-trained CNN, a pruning rate, and four pruning criteria, while the output is a codebook \( B = [B_1, B_2, \ldots, B_4] \) with \( B_j = [s_1, s_2, \ldots, s_{10}] \).

3.4 Encoding and decoding

Let \( T \) represent the total number of filters of the original model. Then, there are \( 2^T \) different combinations corresponding to \( 2^T \) solutions for compressing the original model. With the increasing value of \( T \), the number of solutions increases exponentially, which results in a large search space and makes it difficult to achieve a well-compressed model with limited time and energy.
1) Encoding: To solve the above problem, we design an efficient encoding method for dimensionality reduction. Based on the codebook $B$ obtained by the ensemble framework and the features of EAs [2], we set the length of chromosome to 50, in which one chromosome represents a solution with each gene representing a set of filters. Let $I$ represent the length of the chromosome.

In general, the specific encoding scheme is structured into two steps. The first step is to divide the binary vectors in the codebook according to a given chromosome length. The second step is to assign values to individuals according to the codebook.

**ALGORITHM 3: Decoding**

1. **Input:**
   - $I$: the length of chromosome
   - $C$: a well-trained CNN
   - $L$: the number of convolutional layers of $C$
   - $B = [B_1, B_2, ..., B_{4R}]$: codebook
   - $X = [x_1, x_2, ..., x_I]$; individual code
   - $F = \{F_1, F_2, ..., F_L\}$: the filter set of $C$
   - $N_i$: the number of filters for the $i$th layer

2. **Output:**
   - $C_y$: a pruned model

3. **for** $j = 1, 2, I$ **do**
   4. $y_j \leftarrow B[x_j][j]$; $y_j$ is a one-dimensional vector
5. **end for**
6. $Y' \leftarrow \text{One-Dimensional-Expansion}(Y)$; 
   
   $Y = [y_1, y_2, ..., y_I]$ and $Y' = [s_1, s_2, ..., s_T]$ with $T = \sum_{i=1}^{L} N_i$
7. low $\leftarrow 1$
8. high $\leftarrow N_I$
9. $M_1 \leftarrow Y'[\text{low} : \text{high}]$;
10. **for** $i = 2, 3, L$ **do**
11. low $\leftarrow \text{high} + 1$
12. high $\leftarrow \text{high} + N_i$
13. $M_i \leftarrow Y'[\text{low} : \text{high}]$; $M_i$ is the mask for the $i$th layer, which determines if a particular filter is included or pruned
14. **end for**
15. $C_y \leftarrow M \circ F$; $C_y$ is a pruned model obtained by removing part of the filters from $C$
16. Initializes $C_y$ with the weights inherited from model $C$;
17. Return the pruned model $C_y$;

First, each binary vector $B_j = [s_1, s_2, ..., s_T]$ in $B = [B_1, B_2, ..., B_{4R}]$ is divided into $I$ segments, given the chromosome length of $I$. The detail of the dividing process is shown in Algorithm 2. The entire process of dividing the codebook $B$ is performed only once. Let $\text{len} = \frac{T}{I}$ denote the total number of filters $T$ divided by the number of segments $I$ (round down), the first $I - 1$ segments of $B_j$ contain $\text{len}$ elements, while the last segment contains $(T - (I - 1) \times \text{len})$ elements. Using the codebook output from Fig. 3, where $R = 1$ and $T = 10$, the length of chromosome equals to three. As shown in Fig. 4, we split each binary vector $B_j = [s_1, s_2, ..., s_{10}]$ into three segments to obtain a resized codebook.

Then, we employ an integer encoding scheme. The range of each gene depends on the length of the codebook $4R$. Let $X = [x_1, x_2, ..., x_I]$ represent the code of an individual, $x_k$ ranges from 1 to $4R$ and points to the $k$-th seg-
2) Decoding: To evaluate the performance of the evolved solutions, each chromosome (the genotype of an individual) is decoded into a pruned model (the corresponding phenotype). The decoding process is divided into two steps. First, for the code $X = \{x_1, x_k, \ldots, x_I\}$ of individual $P_Y$, we decode every $x_k$ to get a binary vector by looking up the codebook $B$. When this process is completed, a total of $I$ binary vectors can be obtained to form a mask. We then decode the mask by removing part of the filters from the original model to get a pruned model. In this way, we use one gene to encode the information of multiple filters in order to achieve dimensionality reduction.

The details of the decoding process are shown in Algorithm 3. The individual code $X = [x_1, x_2, \ldots, x_I]$ is decoded against the codebook $B$ to get a sequence of vectors $Y = [y_1, y_j, \ldots, y_I]$ as follows: $y_j = B[x_j][j]$. Then, calculate the mask $M = \{M_1, M_2, \ldots, M_L\}$ corresponding to $Y$, where $M_i$ is the mask for the $i$th layer. Then, remove the corresponding filters in the filter set $F = \{F_1, F_2, \ldots, F_L\}$ with the bit value 0 in the mask $M$ and obtain a pruned model $C_Y$.

Finally, we inherit parameters from the well-trained original model to initialize $C_Y$.

Assuming the original model $C$ has two convolutional layers, each of which has five filters. Continue with the codebook $B = [B_1, B_2, B_3, B_4]$ output from Fig. 4, where the length of the codebook equals four and the length of chromosome equals three. As shown in Fig. 5, a random individual code $X = [4, 1, 3]$ is decoded by looking up the codebook $B$ to get a mask (the left part of Fig. 5). Then, under the guidance of this mask, we remove some filters from each layer of the original model $C$ with the bit value 0 in the mask to obtain a pruned structure (the right part of Fig. 5).

The encoding length $I$ is smaller than the number of filters (usually 64) of all layers. Therefore, the length of the binary vector decoded by $x_k$ is longer than the total number of filters in most layers. For most layers, the pruning information of the same layer comes from the same gene code $x_k$. In this way, we can maintain each layer’s structural integrity to be compressed, increase the scalability of the scheme, improve the search efficiency, and perform automatic sensitivity analysis.
3.5 Obtaining the optimal pruned model via MOEA

The overall process is shown in Algorithm 4. We start with initializing a population $P_1$. Each individual is a solution compressing the original model. Each generation of the evolutionary process consists of the following operations. The parent $P_t$ produces the offspring $Q_t$ through genetic operations. The offspring $Q_t$ is merged into the parent $P_t$ to form $O_t$, and the objective value for each individual is evaluated. Then, individuals are sorted into different ranks ($F_1, F_2, \ldots$) by non-dominated sorting [5]. After that,
individuals with higher ranks are subsequently selected to enter $P_{t+1}$ if the mating pool is not fully occupied. Once $|P_{t+1}| + |F_t| > 4R$, which means the number of individuals in $F_t$ exceeds the number of remaining mating slots, a rank mechanism based on crowding distance is utilized to decide which solutions will be selected from $F_t$. In this way, Pareto optimal solutions from the population at each generation are preserved into the next generation with higher priority, which guides the population towards the optimal Pareto front. The evolutionary process repeats the above operations until the termination condition is satisfied. After that, we select only one optimal solution with a delicate balance between two pruning objectives from the last generation by Minimum Manhattan Distance (MMD) [4]. Finally, an optimal pruned model is obtained by the process of decoding, weight inheritance, and fine-tuning.

1) Population Initialization: Let $P_1 = \{p_1, p_2, \ldots, p_{4R}\}$ represent the initial population. We initialize the code of individual $p_k$ with $X = [x_1, x_2, \ldots, x_I]$, where $x_j = k$ and $1 \leq k \leq 4R$. Individual $p_k$ encodes the $k$-th mask in the codebook $B$, and the initial population $P_1$ encodes the entire codebook, which is generated by the ensemble framework proposed by 3.3. In this way, evolutionary pruning begins with a quality population, in which each individual encodes a fine-pruned structure.

2) Crossover and Mutation: CEA-MOP adopts one-point crossover (line 6, Algorithm 4), which means that only one crossing point is randomly set in the individual coding, and then part of the chromosomes of two paired individuals are exchanged at this point. Specifically, we successively select a pair of individuals from the current population as parents (each individual will be selected once). A random number $r_p \in [0, 1]$ is generated. If $r_p < p_c$, two children are generated by exchanging part of the chromosomes of two paired parents, and then the parents are replaced by the children. Otherwise, the chromosomes of the parents keep the same.

During mutation (line 7, Algorithm 4), a random number $r_m \in [0, 1]$ is generated for each gene of the individual in the offspring. If $r_m < p_m$, change the value of the gene to any random integer between $[1, 4R]$, otherwise, the value will not change.

Starting with the same codebook $B = [B_1, B_2, B_3, B_4]$ illustrated in Fig. 4, where the original model has two convolutional layers, with each having five filters. The process of crossover and mutation is shown in Fig. 6. The code of one individual is decoded into a mask and a pruned structure in turn. In Fig. 6, we illustrate the genetic operators on codes, masks, and pruned structures in turn. First, we detail the example of crossover shown in the upper part of Fig. 6. Given a random code $[4, 1, 3]$ of $p_1$, a mask $[[001], [110], [1100]]$ is first decoded against the codebook $B$, which is further decoded into a pruned structure, where the pruning rate (the ratio of the white circle) of the first layer is 40%, and that of the second layer is 60%, respectively. In the crossover, the third gene of $p_1$ and $p_2$ is exchanged. Accordingly, the third segment of the mask of $p_1$ and $p_2$ is exchanged. The offspring generated from $p_1$ is $q_1$, where the code is $[4, 1, 1]$, and the decoded mask is $[[001], [110], [1101]]$. For the pruned structure decoded by $q_1$, the pruning rate of the first layer remains 40%, while that of the second layer has decreased from 60% to 40%. The process of mutation is shown at the bottom of Fig. 6. For the newly generated offspring $q_1$, the value of its third gene is changed from one to two. Similarly, the third segment of the mask and the filter information of the pruned structure is changed.

For the individual $X = [x_1, x_2, \ldots, x_I] (1 \leq x_j \leq 4R)$, it integrates multiple pruning rates and pruning criteria. $x_j$ is constantly changing as $X$ crosses with other individuals or mutates itself which means that the model structures with various pruning rates and pruning criteria in the design domain are freely combined to form new pruned models.

As discussed in [25], some layers are quite sensitive to filter pruning, which means that when some filters from the sensitive layers are pruned away, it may not be possible to recover the original accuracy. One way is to experimentally analyze the sensitivity of each layer, then manually determine the number of filters to prune for each layer based on their sensitivity, which is resource-intensive and suboptimal. Therefore, a compression method that can automatically discover optimal pruning ratio is desired.

In the direct binary coding scheme, since genetic operators constantly modify the binary bit of each filter in each layer with the same predefined probabilities, the chances of each layer being skipped are very small. In our encoding scheme, each gene represents a series of filters. In the process of crossover and mutation, genetic operators constantly modify the integer of each gene, which means that a qualified mask with a certain pruning rate is decoded from the codebook to mask the corresponding series of convolutional filters. In this way, a series of filters have a chance to be skipped or pruned with a small percentage of pruning rate. Thus, sensitivity analysis on each layer of the model is automatically carried out and the upper bound of pruning rate for each layer according to its sensitivity degree is determined.

3) Constraint Handling: In the evolutionary process, if we do not consider accuracy, technically speaking, any pruning rate can be achieved. That is, the second objective (the number of filters) is considered relatively easier to be optimized than the first objective (the error rate of the model on validation set). In order to prevent the phenomenon that too few filters are reserved thus producing a large number of models without satisfying performance, a
constraint is applied on non-dominated sorting. By adding a maximum constraint $C_{\text{cons}}$ to the first objective, the performance of the resulting solution can be guaranteed. The solutions with the first objective value lower than $C_{\text{cons}}$ are feasible solutions, otherwise they are infeasible. When two solutions $p$ and $q$ are compared, $p$ is considered dominating $q$ under the following conditions.

- $p$ is feasible and $q$ is not;
- $p$ and $q$ are both feasible and the objective value of $p$ is lower than that of $q$;
- $p$ and $q$ are both infeasible and the first objective value of $p$ is lower than that of $q$.

4) Computational Complexity: In the evolutionary process, the fitness evaluation requires no re-training before evaluating the accuracy of each pruned model, which is computationally efficient. Thus, the main computational complexity of the evolutionary pruning comes from the validation of model accuracy, which is shown as follows:

$$G \times |P| \times T(\text{validation}), \quad (4)$$

where $G$ is the maximum number of iterations, $|P|$ is the population size, $T(\text{validation})$ is the time required to test the accuracy of a model on the validation set.

### 4 Experiments

The experiments are designed for answering a series of critical questions: Question 1 (Q1): Whether the proposed CEA-MOP is effective in compressing widely used models? Question 2 (Q2): How does CEA-MOP perform compared with other popular pruning models? Question 3 (Q3): How efficient is CEA-MOP compared with other popular pruning models? Question 4 (Q4): How does the number of integrated pruning metrics affect the performance of CEA-MOP? Question 5 (Q5): Whether CEA-MOP can be extended to a highly densely connected DenseNet network? In our pruning experiment, CIFAR10 [21], CIFAR100 [47], and ImageNet [43] datasets are considered widely used benchmarks, while ResNet [10] is treated as the common network architectures and DenseNet [17] is treated as highly densely connected network.

In 4.1, we introduce the features of all datasets adopted in the experiment. The baseline algorithms are described in detail in 4.2 and three criteria are formulated to evaluate the performance of pruning methods in 4.3. The implementation details of the experiments are introduced in 4.4. Then, we compare CEA-MOP with conventional and evolutionary pruning approaches for the residual network ResNet (20, 32, 56, and 110-layer) on the CIFAR10 dataset, which is reported in 4.5. Next, we apply our method to the task of compressing ResNet (18, 34, 50, and 101-layer) on the ImageNet dataset, which is described in 4.6. Besides, we conduct the experiment on DenseNet-40 on the CIFAR100 dataset in 4.7. After that, we systematically analyze the experimental results in 4.8. Finally, we analyze some parameters during the experiment in 4.9.

### 4.1 Dataset

We have well-exploited three popular benchmark datasets, CIFAR10, CIFAR100, and ImageNet. In the widely used standard dataset CIFAR10, there are 10 categories of RGB color images, i.e., planes, cars, birds, etc. The dataset contains a total of 60,000 pictures and is divided into
50,000 training pictures and 10,000 test pictures. On the other hand, CIFAR100 has 100 classes. Each class contains 600 images, 500 of which are used as training images while the remaining 100 as test images. Alternatively, CIFAR100 can be divided into 20 categories, each containing five subclasses. ImageNet is a large-scale dataset containing 1.28 million training images and 50,000 validation images of 1000 classes.

4.2 Baseline algorithms

We compare our method with several conventional and evolutionary pruning methods. The conventional method includes: (1) AWS [25] prunes filters through an absolute weighted sum of filters. (2) ThiNet [32] establishes the objective of filter pruning as an optimization problem, and prunes the filter based on the outputs of its next layer. (3) Artificial Bee Colony pruning algorithm (ABCPruner) [28] proposes a new filter pruning method based on artificial bee colony algorithm to automatically find optimal pruned structure. (4) Sparse Structure Selection (SSS) [18] utilizes group sparsity with $l_1$-regularization on the scaling parameters to generate a sparse network by training with class labels. (5) Generative Adversarial Learning (GAL) [29] proposes a label-free generative adversarial learning to prune the network with a sparse soft mask. (6) Channel Pruning (CP) [11] determines which channels to be removed by minimizing the feature reconstruction error of the next layer. (7) Soft Filter Pruning (SFP) [12] dynamically prunes the filters in a soft manner with less dependence on the pre-trained model. The evolutionary pruning methods considered are as follows. (1) KGEA [62] establishes filter pruning as an MOP. (2) Towards Evolutionary Compressing (TEC) [49] represents each compressed network as a binary individual and designs a fitness function with a tradeoff parameter.

4.3 Metrics

We adopt three metrics to evaluate the performance of the proposed CEA-MOP for compressing CNNs: the classification accuracy of the model on the given dataset, the floating point operations (labeled FLOPs in the table of experimental results) reduction between the pruned model and the original model, the parameters reduction between the pruned model and the original model, obviously the larger, the better.

FLOPs refer to the number of computations required for model inference which directly corresponds to the time complexity. Parameters refer to the number of parameters in the model, which directly determines the size of the model and also affects the amount of memory occupied when the model is inferred, corresponding to the space complexity. The number of parameters and floating point operations (FLOPs) is computed as follows:

**Convolutional layer:**

$$\begin{cases} 
\text{parameter} = C_{\text{out}} \times C_{\text{in}} \times K_w \times K_h \\
\text{FLOPs} = C_{\text{out}} \times C_{\text{in}} \times H \times W \times (2 \times K_w \times K_h - 1) 
\end{cases}$$

(5)

**Fully connected layer:**

$$\begin{cases} 
\text{parameter} = C_{\text{out}} \times C_{\text{in}} + C_{\text{out}} \\
\text{FLOPs} = C_{\text{out}} \times C_{\text{in}} \times 2 
\end{cases}$$

(6)

where $C_{\text{in}}$ and $C_{\text{out}}$ represent the number of input and output channels, $K_w$ and $K_h$ represent the size of conventional filters, $H$ and $W$ represent the height and width of the feature maps.

4.4 Experiment settings

The experiment of CEA-MOP consists of four parts: pre-training, integration, pruning, and fine-tuning. First, we train ResNet (18, 20, 32, 34, 50, 56, 101, and 110-layer), and DenseNet (40) as the original models from scratch. Given 50 different settings of pruning rates, we generate a codebook with 200 pruned models by integrating four conventional pruning criteria [13, 16, 25, 33]. The evolutionary pruning process accepts the 200 pruned models as the initial population with each individual representing a pruned model. Through crossover and mutation, individuals with higher classification accuracy and fewer filters are selected into the next generation. Once the maximum iteration number is met, terminate the evolution, output only one optimal pruned model with a delicate balance between model size and classification accuracy via the minimum Manhattan distance (MMD) [4] approach. To improve the performance of the pruned model, we inherit parameters from the well-trained original model to initialize it, then fine-tuning it on the training set for some epochs. We run all experiments on Tesla V100 16GB Graphics Processing Unit (GPU)

**Training Settings.** The CNNs are all implemented in Pytorch [35]. For ResNets and DenseNets, we use the Stochastic Gradient Descent algorithm (SGD) with a mini-batch size of 64. The learning rate starts from 0.1 and is divided by 10 when the error plateaus. We use a weight decay of 0.0001 and a momentum of 0.9. Based on dataset sizes and computing resources, the training epochs for ResNets on CIFAR10, DenseNets on CIFAR100, and ResNets on ImageNet are set to 300, 160, 100, respectively.

---

1. [https://github.com/pytorch](https://github.com/pytorch)
Integration Settings. To ensure some sensitive layers are pruned with minimal pruning rates in filters or have a chance to skip being clipped, the upper and lower limits of pruning rate for integrating the codebook are set to 0% and 49%, respectively. For pruning criteria AWS [25] and FP-GM [13] that simply involve operations on the weights of filter, we compute the redundancy value for each filter on the CPU. For pruning criteria APoZ [16] and TEL [33] that involve operations on the output for each filter, we use the GPU to compute the redundancy value.

Evolutionary Settings. The population size is 200 and the length of individual coding is 50. The probabilities of crossover and mutation are set to 1 and 0.01, respectively. For ResNets on CIFAR10, the maximum iteration number is 500 and the accuracy constraint is 60%. For ResNets on ImageNet, the maximum iteration number is 250, due to the high time cost of evaluating each newly generated individual on the validation sets of ImageNet. The accuracy constraint is set to 0%, due to the underfitting of models on this datasets. After the algorithm terminates, one individual is selected from the last generation. Due to the random feature of evolution, the evolutionary process is repeated five times, and the one with the median accuracy is selected for comparison.

Fine-tuning Settings. On CIFAR10, we fine-tune the pruned ResNet networks for 120 epochs with a learning rate of 0.0001. On ImageNet, we fine-tune the pruned ResNet networks for 100 epochs with a learning rate of 0.0001. On CIFAR100, we fine-tune the pruned DenseNet networks for 60 epochs with a learning rate of 0.0001. Other parameter settings are the same as those for training strategy.

4.5 Experiments on CIFAR10

We firstly perform our method on the CIFAR10 dataset for ResNets with different depths, including 20, 32, 56, and 110, which contain one input layer, one output layer, and a number of hidden layers. Each hidden layer is composed of residual blocks and each block contains two convolutional layers with a kernel size of 3×3.

Results on CIFAR10. To answer Q1, we show the results of compressing ResNet with different depths on CIFAR10 in Table 1. The “FLOPs ↓” is the FLOPs reduction between the pruned model and the original model, the larger the better. The “Parameters ↓” is the parameters reduction between the pruned model and the original model, the larger the better. The top1-accuracy of pruned ResNet increases from depth 20 to depth 110. Detailely, ResNet-20 achieves 40.0% parameters reduction, 42.2% FLOPs reduction with an accuracy drop of 1.26%. Moreover, the top1-accuracy of the pruned ResNet-110 has increased by 0.10% compared to the baseline ResNet-110 with 58.2% parameters reduction and 59.9% FLOPs reduction. Generally speaking, ResNet-110 is highly generalized and overly parameterized on CIFAR10. The experimental results verify the effectiveness of CEA-MOP in compressing the residual-designed networks.

Comparison with other methods. To answer Q2, we use ResNet-56 as our baseline model following previous works [27, 29, 50]. In Table 2, we compare CEA-MOP with conventional pruning methods AWS [25], ThiNet [32], SSS [18], ABCPruner [28], SFP [12], GAL [29], and CP [11], and evolutionary pruning methods KGEA [62] and TEC [49]. The results show that CEA-MOP obtains higher accuracy compared with conventional pruning and evolutionary pruning methods. For example, AWS achieves 27.6% FLOPs reduction with 0.63% accuracy drop and TEC achieves 50.2% FLOPs reduction with 2.11% accuracy drop. On the other hand, CEA-MOP achieves 51.1% FLOPs reduction with merely 0.05% accuracy drop. Hence, CEA-MOP is more advantageous in finding the optimal pruned structure compared with popular pruning methods.

Efficiency. To answer Q3, we use the overall training epochs (omitting the epochs to train the original model) to measure the computational complexity following previous work [29]. The epoch represents a complete training of the model using all the training samples of the dataset. For the sake of comparison, the max number of epochs required for fine-tuning ResNets on CIFAR10 is set to 120. As shown in Table 2, CEA-MOP which belongs to a three-stage pruning method requires smaller training epochs of 128 compared with others. Notably, the ensemble framework requires eight epochs and the evolutionary process requires no additional training epoch. 120 epochs are required to fine-tune the output network of evolutionary pruning. Saliency-based pruning methods (ThiNet and CP) are extremely time-consuming since they require iterative pruning and optimizing (220 epochs) to obtain a pruned structure and additional 120 epochs to fine-tune the pruned network. At the expense of more accuracy drop, conventional pruning methods AWS, SSS, and SFP require no additional epochs for searching the pruned structure. A total of 220 epochs is required by GAL, where 100 epochs are required to retain the model, and additional 120 epochs for fine-tuning the pruned structure, which is more than that required by CEA-MOP, and the accuracy drop is higher than that of CEA-MOP. Besides, CEA-MOP also shows a lower accuracy drop against the evolutionary-based pruning methods.
KGEA and TEC, which belong to a two-stage method where no additional epoch is first adopted to evolve the optimal pruned structure and 120 epochs are required for fine-tuning the pruned network. For pruning methods achieving non-dominated performance as CEA-MOP, they require more epochs, e.g., ABCPruner, although it achieves 54.1% FLOPs reduction, it costs 12 epochs to retain the model, and additional 120 epochs to fine-tune the pruned structure. Meanwhile, the performances of pruning methods requiring fewer epochs are clearly dominated by CEA-MOP. Therefore, CEA-MOP is considered more efficient compared with conventional and evolutionary pruning methods.

### 4.6 Experiments on ImageNet

We further perform our method on ResNets for the large-scale ImageNet. The ResNets are with different depths, including 18, 34, 50, and 101, which contain one input layer, one output layer, some hidden layers. Each hidden layer is composed of residual blocks. For ResNet-18 and ResNet-34, each block contains two convolutional layers with a kernel size of $3 \times 3$. For ResNet-50 and ResNet-101, each block contains three convolutional layers with a kernel size of $3 \times 3$ or $1 \times 1$.

#### Results on ImageNet

We also answer Q1 via the experiment on ImageNet which is a widely used large-scale dataset. Besides Top1-accuracy, we follow the previous works [13, 29] to report the Top5-accuracy on the ImageNet dataset. By comparing Tables 1 and 3, we can see that the accuracy drops on ImageNet are more than those on CIFAR-10. To explain, ResNet itself is a compact model, there might exist fewer redundant parameters. On the other hand, ImageNet is a large-scale dataset containing 1000 categories, which is much complex than the small-scale CIFAR-10 with only 10 categories. Moreover, CEA-MOP can still reduce some FLOPs with insignificant top-1 accuracy drop. For example, for pruning a pre-trained

| Model              | Top1-acc (%) | Parameters ↓ (%) | FLOPs ↓ (%) |
|--------------------|--------------|------------------|-------------|
| ResNet-20 baseline | 92.26        | 0.0              | 0.0         |
| CEA-MOP            | 91.00        | **40.0**         | **42.2**    |
| ResNet-32 baseline | 92.63        | 0.0              | 0.0         |
| CEA-MOP            | 92.01        | **38.4**         | **41.3**    |
| ResNet-56 baseline | 93.69        | 0.0              | 0.0         |
| CEA-MOP            | 93.64        | **49.4**         | **51.2**    |
| ResNet-110 baseline| 93.78        | 0.0              | 0.0         |
| CEA-MOP            | 93.88        | **58.2**         | **59.9**    |

The “FLOPs ↓ (%)” is the FLOPs reduction from the original model to the pruned model, the larger the better. The “Parameters ↓ (%)” is the parameters reduction from the original model to the pruned model, the larger the better. The best results are highlighted in bold.

| Model              | FLOPs ↓ (%) | Top1-acc (%) | Epochs |
|--------------------|-------------|--------------|--------|
| ResNet-56 baseline | 0.0         | 93.69        | –      |
| CEA-MOP            | 51.1        | 93.64        | 128 (8 + 120) |
| AWS                | 27.6        | 93.06        | 120    |
| ThiNet-60          | 50.2        | 92.88        | 340 (220 + 120) |
| SSS                | 50.6        | 93.16        | 120    |
| ABCPruner          | 54.1        | 93.23        | 132 (12 + 120) |
| SFP                | 52.6        | 92.26        | 120    |
| GAL-0.6            | 37.6        | 93.38        | 220 (100 + 120) |
| GAL-0.8            | **60.2**    | 91.58        | 220 (100 + 120) |
| CP                 | 50.6        | 91.80        | 340 (220 + 120) |
| KGEA               | 50.4        | 93.00        | **120** |
| TEC                | 50.2        | 91.58        | 120    |

The epoch represents a complete training of the model using all the training samples of the dataset. For baseline which is not been pruned, no additional training epochs are required, so the notation ‘-‘ means they are not shown in tables.
ResNet-101, CEA-MOP reduces more than 50% FLOPs of the model with 0.19% top-5 accuracy loss and only negligible (0.22%) top-1 accuracy drop, which demonstrates that CEA-MOP can remove redundant filters from ResNet to achieve a compressed network given a large dataset.

**Comparison with other methods.** To further answer Q2, we use ResNet-50 as our baseline model which is widely used in previous works [13, 28, 29]. The effectiveness of CEA-MOP for pruning ResNet-50 on ImageNet is compared with conventional pruning methods, AWS, ThiNet, SSS, ABCPruner, SFP, GAL, and CP, and evolutionary pruning methods, TEC and KGEA. As shown in Table 4, CEA-MOP model achieves 50% FLOPs reduction with 1.32% accuracy drop on ResNet-50. However, the accuracy of SSS and CP models decreases by 2.09% and 3.97%, respectively. CEA-MOP maximizes the advantages of individual conventional and evolutionary pruning methods through an ensemble framework, which is the main reason for its superior performance.

**Efficiency.** We also show the overall training epochs for all methods in Table 4 to answer Q3. For the sake of comparison, the max number of epochs required for fine-tuning ResNet-101, CEA-MOP reduces more than 50% FLOPs of the model with 0.19% top-5 accuracy loss and only negligible (0.22%) top-1 accuracy drop, which demonstrates that CEA-MOP can remove redundant filters from ResNet to achieve a compressed network given a large dataset.

### 4.7 Experiments on CIFAR100

**DenseNet-40 on CIFAR100.** To answer Q5, we conduct an experiment of compressing DenseNet on CIFAR100. For DenseNet, we use a layer 40 DenseNet with a growth rate of 12 (DenseNet-40), which is widely used in previous work [30, 64]. First, we train DenseNet-40 on the training set of CIFAR100 for 160 epochs as the original model. In ResNet, each layer connects to several layers (typically two or three layers). However, in DenseNet, all layers are connected to each other. Specifically, each layer accepts all the outputs from preceding layers as its additional input. If we directly remove some filters of each layer on DenseNet, the input of the subsequent layers will be influenced, thus

### Table 3 Results of pruned ResNet from CEA-MOP on ImageNet

| Model              | Top1-acc (%) | Top5-acc (%) | Parameters ↓ (%) | FLOPs ↓ (%) |
|--------------------|--------------|--------------|------------------|-------------|
| ResNet-18 baseline | 69.75        | 89.06        | 0.0              | 0.0         |
| CEA-MOP            | 68.01        | 88.03        | 18.7             | 32.5        |
| ResNet-34 baseline | 74.01        | 91.09        | 0.0              | 0.0         |
| CEA-MOP            | 72.90        | 90.80        | 33.1             | 41.3        |
| ResNet-50 baseline | 76.27        | 92.99        | 0.0              | 0.0         |
| CEA-MOP            | 74.95        | 92.46        | 31.7             | 55.2        |
| ResNet-101 baseline| 77.60        | 93.44        | 0.0              | 0.0         |
| CEA-MOP            | 77.38        | 93.25        | 44.5             | 53.4        |

### Table 4 Comparison of pruned ResNet on ImageNet

| Model              | FLOPs ↓ (%) | Top1-acc (%) | Top5-acc (%) | Epochs          |
|--------------------|-------------|--------------|--------------|-----------------|
| ResNet-50 baseline | 0.0         | 76.27        | 92.99        | –               |
| CEA-MOP            | 55.1        | 74.95        | 92.46        | 108 (8 + 100)   |
| AWS                | 53.7        | 71.04        | 90.05        | 100             |
| ThiNet-30          | 73.2        | 68.42        | 88.30        | 296 (196 + 100) |
| ThiNet-50          | 58.3        | 71.01        | 90.02        | 296 (196 + 100) |
| SSS-26             | 43.2        | 71.82        | 90.79        | 100             |
| SSS-32             | 31.3        | 74.18        | 91.91        | 100             |
| ABCPruner          | 53.9        | 73.86        | 91.69        | 112 (12 + 100)  |
| SFP                | 41.7        | 62.14        | 84.60        | 100             |
| GAL-0.5            | 43.2        | 71.95        | 90.04        | 190 (90 + 100)  |
| GAL-0.5-joint      | 55.2        | 71.80        | 90.82        | 190 (90 + 100)  |
| CP                 | 33.5        | 72.30        | 90.80        | 296 (196 + 100) |
| KGEA               | 39.3        | 72.01        | 91.10        | 100             |
| TEC                | 34.9        | 70.73        | 89.43        | 100             |
resulting in significant performance degradation. Thus, filter pruning methods directly imposing pruning criteria on filters to identify their redundancy are effective for VGGNet and ResNet. Unfortunately, they do not work for DenseNet. As a result, different from the previous experiment, we impose pruning criteria, AWS, APoZ, TEL, and FPGM, on the batch normalization (BN) layers, which have no influence on the residual input of each layer.

Since all baseline algorithms are designed without considering the DenseNet structure, we use the four pruning metrics integrated as the baseline algorithms. We migrate the four integrated methods (AWS, APoZ, TEL, and FPGM) to DenseNet. The experimental results are shown in Table 5. For APoZ and TEL based on iterative pruning and optimizing, they require an additional 144 epochs to search for the pruned structure. Although FPGM and AWS require only 60 epochs to fine-tune the pruned structure, the accuracy drops are significant. One reason for poor performance is that none of the four pruning standards is specifically designed for the DenseNet structure. Only TEL directly depending on the loss of the model to remove parameters achieves comparable Top1-accuracy with the original model. Therefore, the ensemble framework can overcome the shortcoming of each approach and effectively exploit their power to the maximum degree. CEA-MOP integrating these four metrics achieves 22.66% FLOPs reduction with only 0.21% accuracy drop, which verifies the extensibility and flexibility of the proposed ensemble framework.

4.8 Results discussion

As shown by the results in Table 2 and Table 4, the proposed pruning algorithm CEA-MOP can reduce the number of FLOPs in CNN significantly while preserving good test accuracy with moderate computation. Compared with evolutionary pruning KGEA and TEC, CEA-MOP can achieve significantly higher test accuracy and FLOPs reduction with slightly more computational cost. Compared with conventional pruning AWS and SSS that take less training epoch, there is a significant improvement in accuracy and FLOPs reduction for CEA-MOP. Moreover, CEA-MOP achieves better classification performance with less computational cost than the non-dominated pruning methods GAL, ABCPruner, and CP. The results suggest that CEA-MOP can produce a better tradeoff between model size and performance than conventional and evolutionary pruning methods.

4.9 Analysis

In this section, we experiment on the CIFAR10 dataset with the widely used ResNet-56 model that has a total of 2042 filters as the baseline model to analyze the parameters in the ensemble framework and evolutionary process.

Analysis of the number of pruning criteria. We conduct the following experiment integrating different pruning criteria to answer Q4. In the Model column, CEA-MOP refers to the design that APoZ, TEL, FPGM, and AWS are utilized to initialize the search space. In (1), (2), (3), and (4), we remove FPGM, APoZ, TEL, and AWS, respectively. The experiment results are shown in Table 6. On the one hand, the performance of the model declined the most after the removal of FPGM. In this study, FPGM may contribute most to our ensemble framework, as it is the only one of the four pruning criteria that does not rely on fine-tuning. On the other hand, the accuracy of the pruned ResNet on CIFAR10 is comparable with the original model regardless of the integration of three or four pruning criteria. Therefore, the number of integrated pruning criteria can be adjusted according to the actual requirements, and we only provide an example as a case study.

Analysis of solutions from different generations. Figure 7 shows the Pareto front (PF) of solutions from different generations during evolution. Each solution represents a pruned model. At the 10th generation, the pruning rate of solutions (blue circle) in the population is low, and there is a significant drop in accuracy when more filters are pruned. After 250 generations of evolution, a set of solutions (yellow squares) dominate those obtained at the 10th generation are generated, which suggests that the search direction of the population has been guided towards the optimal Pareto Front, where the solution can achieve a good compromise between the number of filters and model

| Method       | FLOPs ↓ (%) | Top1-acc (%) | Epochs       |
|--------------|-------------|--------------|--------------|
| DenseNet-40 baseline | 0.0         | 74.61        | –            |
| CEA-MOP      | 22.7        | 74.40        | 68 (8 + 60)  |
| APoZ         | 65.26       | 204 (144 + 60)|            |
| FPGM         | 64.92       | 204 (144 + 60)|            |
| TEL          | 74.20       | 204 (144 + 60)|            |
| AWS          | 66.26       | 60           |              |
performance. When the evolutionary process stops (the maximum iteration number is met), the last generation (the 500th generation) of the population converges and outputs a batch of optimal solutions for compressing ResNet-56 on the CIFAR10 dataset.

Analysis of the optimal solutions from different generations. For the 480th, 490th, and 500th generations, we select one optimal solution by MMD approach for each generation and illustrate them in Fig. 8. Each solution represents a pruned model. We can see that the optimal solutions from different generations, i.e., the blue square, the yellow pentagon, and the red pentagram basically overlap with each other. Thus, we choose only one optimal solution from the last generation (the 500th generation).

Analysis of the output populations from different evolutionary runs. In the experiment, we repeat the evolutionary process five times and take the pruned model with the median accuracy as the preferred one. As it can be seen from Fig. 9, the PF from different evolutionary runs is quite close to each other. Meanwhile, not only are solutions from the same population non-dominated to each other, but most solutions from different runs are non-dominated. Besides, populations from different runs converge to the same objective space. The results suggest that the output populations from different runs have all converged towards the optimal Pareto Front.

Analysis of the optimal pruned models from different evolutionary runs. We repeat the evolutionary process five times and show the optimal pruned model result from each run. To improve the performance of each pruned model, we fine-tune them on the training set for 120 epochs. We use one point in Fig. 10 to represent a model, where x-axis represents the pruning rate in filters, and y-axis denotes the classification accuracy of the model. On the other hand, we evaluate the performance of the pruned models in terms of parameters and FLOPs.

Table 6 Influence of the number of pruning criteria for ResNet-56 ON CIFAR10

| Model              | Top1-acc (%) | Parameters ↓ (%) | FLOPs ↓ (%) |
|--------------------|--------------|------------------|-------------|
| ResNet-56 Baseline | **93.69**    | 0.0              | 0.0         |
| CEA-MOP            | 93.64        | 49.4             | **51.2**    |
| (1)-FPGM           | 93.01        | 35.7             | 40.6        |
| (2)-APoZ           | 93.51        | 45.2             | 47.3        |
| (3)-TEL            | 93.47        | 43.8             | 41.9        |
| (4)-AWS            | 93.58        | 45.8             | 43.1        |

Fig. 7 The PF of solutions from different generations. Blue circles, yellow squares, and red triangles represent solutions at the 10th, the 250th, and the 500th generations, respectively

Fig. 8 The optimal solutions from different generations. Blue square, yellow pentagon, and red pentagram represent the optimal solutions at the 480th, the 490th, and the 500th generations, respectively

Fig. 9 The PF of solutions from different evolutionary runs. We take the solutions from the last generation for each run.
the whole, the accuracy of the model decreases with the increase of pruning rate. Due to different model structures, there are some differences in accuracy among models with similar pruning rates. Compared to the original ResNet-56 that reaches an accuracy of 93.69%, there is a slight decline in accuracy for the pruned models. The pruning rate of the pruned models is between 43% and 44%, and the accuracy of the pruned models is between 93.40% and 93.70%, which means that the differences between these models from different evolutionary runs are relatively subtle. To comparison, we select the model with the median accuracy as the preferred one.

Analysis of the running time of CEA-MOP model. There are four stages of CEA-MOP, where the training for the original model requires 300 epochs, the ensemble framework requires eight epochs, and the fine-tuning stage requires 120 epochs. The evolutionary process requires no additional training epochs but requires running 500 generations, in which most of the time overhead comes from evaluating the accuracy of each pruned model. Here, we take one GPU with 16 GB memory as an example. In practice, the training, evaluation, and fine-tuning of the model can be carried out in parallel on multiple GPUs. The training batch size, validate batch size, and test batch size is set to 64, 500, and 256, respectively. The running times of one epoch and one generation are 0.01 hours and 0.0125 hours, respectively. Therefore, the actual time cost of running once CEA-MOP model is about 10.5 hours (0.01 × 300 + 0.01 × 8 + 0.0125 × 500 + 0.01 × 120).

5 Conclusion

In this paper, model compression of convolutional neural networks is constructed as a multiobjective optimization problem with two conflicting objectives, reducing the scale of parameters and improving the performance of the network. To accomplish this, we propose a novel structured pruning method based on EAs, named CEA-MOP. First, an ensemble framework is developed to generate a qualified initial population for the next evolutionary process. Then, we propose an efficient coding method, which shorten the length of chromosome no matter how deep the model is, thus helping EAs run efficiently in order to push the population toward the Pareto optimal front as much as possible. After that, the evolutionary process automatically searches for the delicate tradeoff between the scale of parameters and model performance and outputs a set of Pareto optimal solutions for compressing the original model. Finally, sensitivity analysis is automatically carried out on each layer of the model and determines the upper bound of pruning rate for each layer, which guides the search towards the target region, thus improving the algorithm’s search efficiency and reducing its computational load. Extensive experiments on CIFAR10 and ImageNet have demonstrated the efficacy of CEA-MOP over conventional and evolutionary-based pruning methods. In future research, we will exploit the interpretability [1] of the model based on the pruned models [46, 47].

Acknowledgements This paper is supported by the joint research program between the Nuclear Power Institute of China and Sichuan University. This work is supported by the National Natural Science Foundation of China under Grant 62076172, the Key Research and Development Project of Sichuan under Grant 2019YFG0494 and 2021YFG0027, the National Key Research and Development Project of China under grant 2017YFB0202403, the State Key Program of National Science Foundation of China under Grant 61836006, and the National Natural Science Fund for Distinguished Young Scholar under Grant 61625204.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article’s Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article’s Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.
References

1. Bau D, Zhou B, Khosla A, Oliva A, Torralba A (2017) Network dissection: quantifying interpretability of deep visual representations. In: CVPR

2. Black T (1998) Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming. Genetic Algorithms/T. Bäck

3. Cheng Y, Wang D, Zhou P, Zhang T (2017) A survey of model compression and acceleration for deep neural networks. arXiv: 171009282

4. Chiu W, Yen GG, Juan T (2016) Minimum Manhattan distance approach to multiple criteria decision making in multiobjective optimization problems. IEEE Trans Evol Comput 20(6):972–985

5. Deb K, Pratap A, Agarwal S, Meyarivan T (2002) A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Trans Evol Comput 6(2):182–197

6. Fletcher PT, Venkatasubramanian S, Joshi SC (2008) Robust statistics on Riemannian manifolds via the geometric median. In: CVPR

7. Graves A, Mohamed A, Hinton GE (2013) Speech recognition with deep recurrent neural networks. In: ICASSP

8. Han S, Pool J, Tran J, Dally WJ (2015) Learning both weights and connections for efficient neural network. In: NIPS

9. Hassibi B, Stork DG (1992) Second order derivatives for network pruning: optimal brain surgeon. In: NIPS

10. He K, Zhang X, Ren S, Sun J (2016) Deep residual learning for image recognition. In: CVPR

11. He Y, Zhang X, Sun J (2017) Channel pruning for accelerating deep convolutional neural networks. In: CVPR

12. He Y, Kang G, Dong X, Fu Y, Yang Y (2018) Soft filter pruning for deep convolutional neural networks. In: IJCAI

13. He Y, Liu P, Wang Z, Hu Z, Yang Y (2019) Filter pruning via automatic structure search. In: IJCAI

14. He Y, Zhang X, Sun J (2017) Channel pruning for accelerating deep convolutional neural networks. In: CVPR

15. Howard AG, Zhu M, Chen B, Kalenichenko D, Wang W, Weyand T, Andreetto M, Adam H (2017) Mobilenets: efficient convolutional neural networks for mobile vision applications. arXiv:170404861

16. Hu H, Peng R, Tai Y, Tang C (2016) Network trimming: a data-driven neuron pruning approach towards efficient deep architectures. arXiv preprint, 160703250

17. Huang G, Liu Z, van der Maaten L, Weinberger KQ (2017) Densely connected convolutional networks. In: CVPR

18. Huang Z, Wang N (2018) Data-driven sparse structure selection for deep neural networks. In: ECCV

19. Jacob B, Kligys S, Chen B, Zhu M, Tang M, Howard AG, Adam H, Kalenichenko D (2018) Quantization and training of neural networks for efficient integer-arithmetic-only inference. In: CVPR

20. Jia K, Tao D, Gao S, Xu X (2017) Improving training of deep neural networks via singular value bounding. In: CVPR

21. Krizhevsky A, Hinton G et al (2009) Learning multiple layers of features from tiny images. Citeseer

22. LeCun Y, Denker JS, Solla SA (1989) Optimal brain damage. In: NIPS

23. Lee N, Ajanthan T, Gould S, Torr PHS (2020) A signal propagation perspective for pruning neural networks at initialization. In: ICLR

24. Li C, Yuan X, Lin C, Guo M, Wu W, Yan J, Ouyang W (2019) AM-LFS: autolm for loss function search. In: ICCV

25. Li H, Kadav A, Durdanovic I, Samet H, Graf HP (2017) Pruning filters for efficient convnets. In: ICLR

26. Li Y, Lin S, Liu J, Ye Q, Wang M, Chao F, Yang F, Ma J, Tian Q, Ji R (2021) Towards compact cnns via collaborative compression. In: CVPR

27. Lin M, Ji R, Wang Y, Zhang Y, Zhang B, Tian Y, Shao L (2020) Hrank: Filter pruning using high-rank feature map. In: CVPR

28. Lin M, Ji R, Zhang Y, Zhang B, Wu Y, Tian Y (2020) Channel pruning via automatic structure search. In: IJCAI

29. Lin S, Ji R, Yan C, Zhang B, Cao L, Ye Q, Huang F, Doermann DS (2019) Towards optimal structured CNN pruning via generative adversarial learning. In: CVPR

30. Liu Z, Sun M, Zhou T, Huang G, Darrell T (2019) Rethinking the value of network pruning. In: ICLR

31. Louizos C, Welling M, Kingma DP (2018) Learning sparse neural networks through l_0 regularization. In: ICLR

32. Luo JH, Wu J, Lin W (2017) Thinet: A filter level pruning method for deep neural network compression. In: ICCV

33. Molchano P, Tyree S, Karras T, Aila T, Kautz J (2017) Pruning convolutional neural networks for resource efficient inference. In: ICLR

34. Mukhopadhyay A, Maulik U, Bandopadhyay S, Coello CAC (2019) A survey of multiobjective evolutionary algorithms for data mining: part I. IEEE Trans Evol Comput 18(1):4–19

35. Paszke A, Gross S, Chintala S, Chanan G, Yang E, DeVito Z, Lin Z, Desmaison A, Antiga L, Lerer A (2017) Automatic differentiation in pytorch. In: NIPS-W

36. Peng H, Wu J, Chen S, Huang J (2019) Collaborative channel pruning for deep networks. In: ICML

37. Persand K, Anderson A, Gregg D (2020) Composition of saliency metrics for channel pruning with a myopic oracle. arXiv: 200403376

38. Precup R, Teban T, Albu A, Zamfirache L, Petreu EM (2020) Evolving fuzzy models for prosthetic hand myoelectric-based control. IEEE Trans Instrum Meas 69(7):4625–4636

39. Qian X, Klabjan D (2021) A probabilistic approach to neural network pruning. In: ICML

40. Rangaiah GP (2008) Multi-objective optimization. IEEE Microw Mag 12(6):120–133

41. Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A (2017) Large-scale evolution of image classifiers. In: ICML

42. Real E, Moore S, Selle A, Saxena S, Suematsu YL, Tan J, Le QV, Kurakin A (2017) Large-scale evolution of image classifiers. In: ICML

43. Russakovsky O, Deng J, Su H, Krause J, Satheesh S, Ma S, Huang Z, Karpathy A, Khosla A, Bernstein MS, Berg AC, Li F (2015) Imagenet large scale visual recognition challenge. Int J Comput Vis 115(3):211–252

44. Song H, Mao H, Dally WJ (2016) Deep compression: compressing deep neural networks with pruning, trained quantization and huffman coding. In: ICLR

45. Sun Y, Yen GG, Yi Z (2019) Evolving unsupervised deep neural networks for learning meaningful representations. IEEE Trans Neural Netw Learn Syst 31(4):1242–1254

46. Sun Y, Xue B, Zhang M, Yen GG (2020) Completely automated CNN architecture design based on blocks. IEEE Trans Neural Netw Learn Syst 31(4):1242–1254

47. Sun Y, Xue B, Zhang M, Yen GG (2020) Evolving deep convolutional neural networks for image classification. IEEE Trans Evol Comput 24(2):394–407

48. Wang D, Zhou L, Zhang X, Bai X, Zhou J (2018) Exploring linear relationship in feature map subspace for convnets compression. arXiv:180305729

49. Wang Y, Xu C, Qiu J, Xu C, Tao D (2018) Towards evolutionary compression. In: KDD

50. Wang Y, Zhang X, Xie L, Zhou J, Su H, Zhang B, Hu X (2020) Pruning from scratch. In: AAAI
51. Wang Z, Li F, Shi G, Xie X, Wang F (2020) Network pruning using sparse learning and genetic algorithm. Neurocomputing 404:247–256
52. Wen W, Xu C, Wu C, Wang Y, Chen Y, Li H (2017) Coordinating filters for faster deep neural networks. In: ICCV
53. Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, Klingner J, Shah A, Johnson M, Liu X, Łukasz Kaiser, Gouws S, Kato Y, Kudo T, Kazawa H, Stevens K, Kurian G, Patil N, Wang W, Young C, Smith J, Riesa J, Rudnick A, Vinyals O, Corrado G, Hughes M, Dean J (2016) Google’s neural machine translation system: bridging the gap between human and machine translation. arXiv:160908144
54. Xie L, Yuille A (2017) Genetic cnn. In: ICCV
55. Xu K, Zhang D, An J, Liu L, Liu L, Wang D (2021) Genexp: multi-objective pruning for deep neural network based on genetic algorithm. Neurocomputing 451:81–94
56. Yang Z, Wang Y, Chen X, Shi B, Xu C, Xu C, Tian Q, Xu C (2019) CARS: continuous evolution for efficient neural architecture search. arXiv:190904977
57. Zagoruyko S, Komodakis N (2017) Paying more attention to attention: improving the performance of convolutional neural networks via attention transfer. In: ICLR
58. Zall R, Kangavari MR (2019) On the construction of multi-relational classifier based on canonical correlation analysis. Int J Artif Intell 17(2):23–43
59. Zhang Q, Hui L (2008) Moea/d: a multiobjective evolutionary algorithm based on decomposition. IEEE Trans Evol Comput 11(6):712–731
60. Zhang X, Zhao Z, Zhang H, Wang S, Li Z (2018) Unsupervised geographically discriminative feature learning for landmark tagging. Knowl Based Syst 149:143–154
61. Zhang X, Zhou X, Lin M, Sun J (2018) Shufflenet: An extremely efficient convolutional neural network for mobile devices. In: CVPR
62. Zhou Y, Yen GG, Yi Z (2019) A knee-guided evolutionary algorithm for compressing deep neural networks. IEEE Trans Cybern 99:1–13
63. Zhou Y, Yi Z, Yen GG (2019) Evolutionary compression of deep neural networks for biomedical image segmentation. IEEE Trans Neural Netw Learn Syst 99:1–14
64. Zhuang L, Li J, Shen Z, Gao H, Zhang C (2017) Learning efficient convolutional networks through network slimming. In: ICCV

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.