Applied Soft Computing 139 (2023) 110229

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Convolutional neural network pruning based on multi-objective feature map selection for image classification

Pengcheng Jiang [a], Yu Xue [a,*], Ferrante Neri [b]

[a] School of Software, Nanjing University of Information Science and Technology, Nanjing, 210044, China
[b] NICE Group, Department of Computer Science, University of Surrey, Guildford, GU2 7XH, United Kingdom

ARTICLE INFO

Article history:
Received 23 November 2022
Received in revised form 15 February 2023
Accepted 14 March 2023
Available online 22 March 2023

Keywords:
Network pruning
Feature map selection
Convolutional neural networks
Multi-objective optimisation
Image classification

ABSTRACT

Deep convolutional neural networks (CNNs) are widely used for image classification. Deep CNNs often require a large memory and abundant computation resources, limiting their usability in embedded or mobile devices. To overcome this limitation, several pruning methods have been proposed. However, most of the existing methods focus on pruning parameters and cannot efficiently address the computation costs of deep CNNs. Additionally, these methods ignore the connections between the feature maps of different layers. This paper proposes a multi-objective pruning based on feature map selection (MOP-FMS). Unlike previous studies, we use the number of floating point operations (FLOPs) as a pruning objective in addition to the accuracy of the pruned network. First, we propose an encoding method based on feature map selection with a compact and efficient search space. Second, novel domain-specific crossover and mutation operators with reparation are designed to generate new individuals and make them meet the constraint rules. Then, decoding and pruning methods are proposed to prune networks based on the results of feature map selection. Finally, multi-objective optimisation is used for evaluation and individual selection. Our method has been tested with commonly used network structures. Numerical results demonstrate that the proposed method achieves better results than other state-of-the-art methods in terms of pruning rate without decreasing the accuracy rate to a high degree.

1. Introduction

Deep convolutional neural networks (CNNs) demonstrate great ability in many computer vision tasks, such as image classification [1], object detection [2] and segmentation [3]. To achieve a high level of accuracy, many large CNNs have been designed that cost a huge amount of computation resources, seriously limiting their applicability to many real-world problems [4,5]. Large and computationally expensive CNNs are not suitable for embedded or mobile devices, which have limited storage space and computing resources [6,7]. For example, one of the most commonly used network structures, VGG-16 [8], requires more than 500 Mb of storage for parameters (*i.e.*, weights) and more than 15G floating point operations (FLOPs), leading to severe challenges for small devices. At present, to overcome the hardware limitations, small devices often send their calculations to external devices on the cloud. Accordingly, this approach relies heavily on the internet network availability and negatively affects the usability of image classification algorithms on small devices.

Recently, many researchers have attempted to solve this problem by designing smaller and more efficient CNNs. The set of techniques associated with these studies is known as network compression. Among the network compression techniques, the most commonly used approaches are network pruning [9–11], knowledge distillation [12], parameter quantisation [13] and neural architecture search (NAS) [14,15]. Knowledge distillation [16] is used to design small networks and employs large models to assist the training of these networks to make them competent for certain complex tasks. The focus of this approach is still on the need for experts to design small networks with good generalisation performance.

Parameter quantisation [13] identifies redundancies and reduces the number of similar parameters. Typical approaches to this include parameter clustering and binary networks. Parameter clustering methods identify clusters of similar parameters and replace the parameters' values with their means. This effectively reduces the number of parameters that need to be stored but does not reduce the computation costs. Binary networks use '+1'

* Corresponding author.
E-mail addresses: pcjiang@nuist.edu.cn (P. Jiang), xueyu@nuist.edu.cn (Y. Xue), f.neri@surrey.ac.uk (F. Neri).

https://doi.org/10.1016/j.asoc.2023.110229
1568-4946/© 2023 Elsevier B.V. All rights reserved.

and '−1' to represent all the network parameters, which reduces the pressure of storage and calculation; however, due to the low generalisation ability, they can only be used on small datasets.

NAS [17,18] is an automatic design method for neural networks. Due to its ability to automatically find the optimal model on a variety of datasets, it is also applied to network compression. Multi-objective optimisation algorithms can be used to search the solution that meets the requirements of multiple objectives to simultaneously meet the needs for task accuracy, computation cost and storage capacity. However, most previous studies have been unable to make good use of the existing pre-trained networks. Some studies also depend on having a well-designed search space for searching a small network, which requires significant expert experience [19].

Network pruning, which is the focus of the present study, seeks for and removes unnecessary parameters and sub-structures within existing networks. Pruning methods can be divided into two categories: (1) those that remove unnecessary parameters to reduce the storage capacity [20] and (2) those that modify the inner structure of the CNN to reduce both its size and FLOPs during calculation [21]. The removal of parameters makes the parameters irregular, meaning this approach is called irregular pruning. Here, '0' replaces the positions, which does not affect the structure. Regular pruning refers to removing or modifying structures to make the network smaller. In comparison with other network compression methods, network pruning uses the existing structures and parameters better and can reduce the need for expert intervention.

Alternatively, modern approaches use dynamic neural networks for pruning. At first, a shrinking learning method is used to obtain a large model that can take into account the performance of substructures. Next, dynamic structures are implemented using dynamic depth, dynamic width, dynamic convolutional kernel size, dynamic input size and dynamic routing [22,23]. This model selects the most appropriate substructure to complete the inference based on the input and usage scenarios. In addition, the use of dynamic parameters is a good strategy to complete dynamic inference by adjusting the coefficients of parameters, features or tasks [24,25]. This approach relies on a unique training method but is highly convenient in practice.

Irregular pruning sets the parameters with a small norm to zero, meaning networks with less storage can be obtained [20]. This method does not change the network structure and does not include the performance of an explicit action to reduce the computation on standard equipment. On the other hand, when specific professional hardware is available, it may lead to benefits in terms of computational efficiency [26–28]. For example, some methods rely on specific devices to support the calculation of sparse tensors for sparse convolution calculation [29,30].

As an alternative to irregular pruning, a number of regular pruning methods have been proposed [31]. These methods remove network structures instead of weights. For example, Li et al. [32] have proposed pruning filters with a smaller norm. They set the number of pruned filters for each layer manually, which depends on expert ability. Considering that the pruning problem can be naturally formulated as an optimisation problem, several meta-heuristic methods can be used to find pruning solutions. For example, Zhang et al. [33] have pruned networks using a genetic algorithm with integer encoding, employing the loss function and 0-norm as two objectives. Additionally, Zhou et al. [34,35] have proposed a multi-objective evolutionary approach to perform pruning, using the loss function and 1-norm as the two objectives. However, these approaches all focus too much on the norm of weights and ignore the importance of the pruning targets, namely, the pruning rates of the FLOPs and parameters. Some other more-standard pruning studies focus on the removal of parameters, meaning they cannot achieve a high pruning rate on FLOPs [32,36].

This paper proposes multi-objective pruning based on feature map selection (MOP-FMS), formulating the pruning as a constrained combinatorial bi-objective optimisation problem and proposing a domain-specific evolutionary algorithm to address the pruning problem of CNNs. The main contributions of MOP-FMS are summarised as follows:.

- Considering the relation between the feature maps from different layers, the pruning problem is formulated as a bi-objective optimisation problem with feature map selection, and the accuracy rate and computation cost are simultaneously optimised.
- A novel feature map-based encoding method and a unique decoding method are proposed for pruning common structures or networks with additive aggregation.
- Special initialisation, crossover and mutation operators are designed with the quick reparation method to satisfy the encoding constraints of this specific problem.

The remainder of this article is organised as follows. Section 2 presents related works on network compression. Section 3 outlines the implementation details of the proposed MOP-FMS. Section 4 displays the experimental results reported in this article. Section 5 provides a conclusion of the work.

2. Related work

2.1. Network pruning

Network pruning is the most direct method to compress existing neural networks [20] and is popular due to its simplicity and effectiveness. Frankle et al. [37,38] have experimentally demonstrated that while, in general, large networks are easier to train directly, while small networks are more difficult in this regard, small networks pruned from large networks can be easily trained to high performance. A theoretical study reaching the same conclusion was performed by Malach et al. [39]. As an example of an irregular pruning method, in [20], parameters characterised by an L_1 norm smaller than a prearranged threshold were set to zero. One limitation of this method is that the pruned network often loses accuracy.

Let us here consider a neural network. The calculation of the output from an input vector can be seen as a set of successive products of matrices by vectors, where each matrix represents each pair of neural network layers. The application of irregular pruning impacts the entries of these matrices without impacting their sizes. More specifically, after pruning, these matrices contain multiple zeros, hence making them sparse.

Scattered zeros in a matrix do not always lead to significant benefits in terms of computational cost. Conversely, if the zeros are clustered on entire rows or columns, calculations can be easily accelerated using, for example, the basic linear algebra subroutine (BLAS) library [26]. Regular pruning can be viewed as a technique that poses to zero entire rows and columns in these matrices. This transformation logically corresponds to a reduction in the sizes of the matrices and thus a reduction in the computation cost in FLOPs to calculate the output.

Fig. 1 represents examples of matrices of layer transitions in two scenarios: after irregular pruning and after regular pruning. The pruned parameters are represented in grey, while the remaining ones are indicated by yellow. As a result, the use of sparse tensor calculations and special optimisations is required to accelerate irregular pruning, which requires expert experience and poses new challenges for deployment.

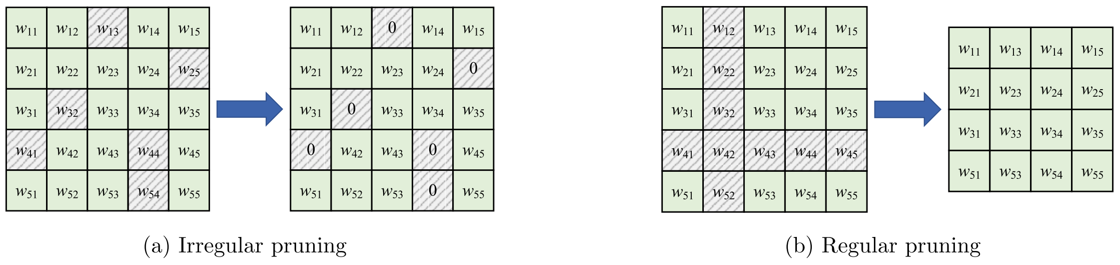

Fig. 1. Parameter matrix after using pruning methods. (a) Irregular pruning: The resulting matrix contains scattered zeros, and its manipulation cannot be accelerated. (b) Regular pruning: The resulting matrix has entire rows and columns removed, making it smaller than the original.

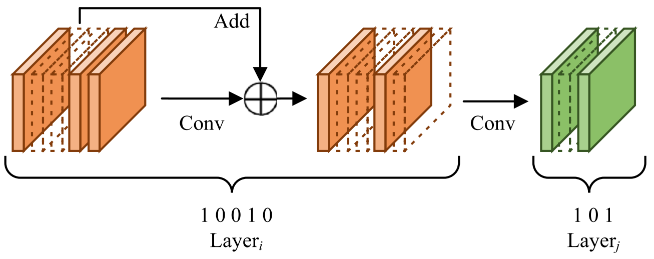

Fig. 2. Feature maps in two consecutive layers: the input feature maps on the left hand side gets processed and generate output feature maps on the right hand side. The output feature maps are divided in two categories: those that have an additive relation with the input ($Output = Conv(Input)+Input$) indicated in orange and those belonging to the common layers for which there is no additive relation ($Output = Conv(Input)$) in green.

Several regular pruning methods have recently been proposed. Hu et al. [40] define the average percentage of zeros of an output to determine whether the corresponding parameters can be removed. This method can minimise the impact on the existing network output but offers low performance in terms of compression rate. Liu et al. [21] suggest that the scaling factors used as affine transformation parameters in batch normalisation can be used to guide the unnecessary channels. However, this method requires the use of a special loss function with a special scale factor to control the scale. These complications in the model pose challenges to the practical application of the method. He et al. [41] use least absolute shrinkage and selection operator regression to choose which channels to remove. Li et al. [32] calculate the L_1 norm of filter weights to determine which filters to be pruned, while Wang et al. have improved this method in GREG-2 [42] using a growing regularisation scheme. He et al. [43] designed the soft filters pruning method, which uses the L_2 norm to determine which filters to be cleared at each epoch.

In recent studies, other forms of pruning have been explored. Some of these include pattern pruning [44,45], layer pruning [46], block pruning [31], filter pruning [9], channel pruning [47] and node pruning [48]. Moreover, some other recent studies have integrated the pruning problem into NAS [49–51]. For example, Dong et al. [52] propose the concept of transformable architecture search (TAS).

These approaches are similar to approaches that search within an existing network using NAS methods [53]. Among the various methods, those based on mathematical programming have displayed excellent performance. In these methods, unwanted parts of the network are gradually removed. The main advantages of these approaches are their theoretical foundations and stability. As examples, Lee et al. Li et al. and Zhang et al. decompose the network pruning process into a convex optimisation problem and solve it using the alternating direction method of multipliers [10,54,55]. Another approach belonging to the category of exact methods is regularisation during back propagation, which aims to make some of the weights unimportant [32,42,56]. Unlike mathematical methods, heuristic algorithms can determine the parts that can be deleted through constant exploration and adjustment of the search algorithm. The principle of using heuristic algorithms for pruning optimisation has low demands and is easy to apply; however, it depends on the reliability of the search algorithm itself.

2.2. Multi-objective evolutionary algorithms

When solving real-world problems, it is often the case that a problem will have multiple objectives that conflict with each other. In this case, satisfying only one objective cannot achieve the desired result. Using a multi-objective evolutionary algorithm is an effective method to solving these practical problems because it uses both an evolutionary algorithm and multi-objective optimisation. At present, multi-objective evolutionary algorithms have been fully researched and developed in many problems, including face recognition [57], image classification [58], feature selection [59,60], and network architecture search [61,62], etc. Additionally, some studies have applied this idea to network pruning and made a number of achievements [33–35].

3. Multi-objective feature map pruning

This section presents the proposed MOP-FMS. The overall framework is introduced in Section 3.5. The key components of MOP-FMS are individually presented and explained in Sections 3.1–3.4.

3.1. Encoding strategy with constraint rules and initialisation

CNNs generally include four kinds of layers: convolution, fully connected, batch normalisation and max (or average) pooling layers. There are multiple convolutional kernels in each convolutional layer. When an input is processed by a convolution layer, each convolutional kernel is applied to the image separately, thus generating certain channels in the output, which are also called feature maps. These feature maps are then used as the input for the following convolution layers to generate additional feature maps. This procedure is repeated for all the convolution layers [63]. Except for the convolution operator, none of the operators change the number of feature maps in each convolution layer.

On the basis of the consideration that feature maps carry important information for image classification tasks, this paper focuses on selecting important feature maps to prune CNNs. As is shown in Fig. 2, all feature maps can be divided into two categories. In the first category, feature maps with additive relations

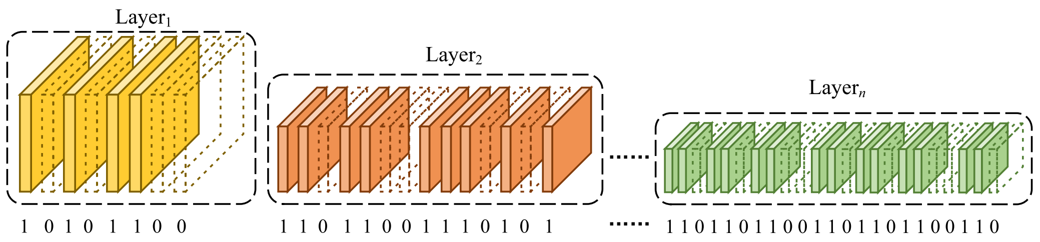

Fig. 3. Example of the encoding strategy for a single individual.

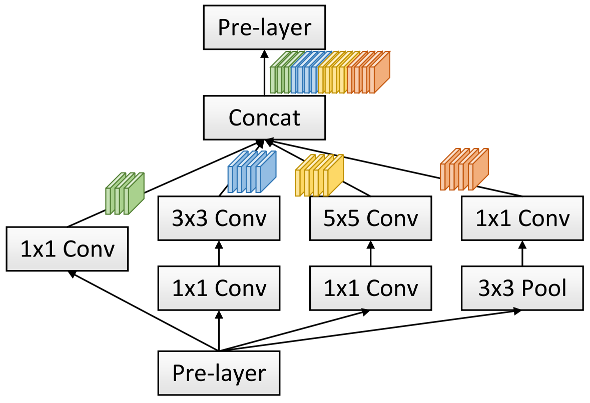

Fig. 4. The feature maps behind concatenation depend on the feature maps from different layers. The same colour indicates the same dependency.

are encoded together and marked in orange. The feature maps from the residual blocks are calculated by adding the input and convolution output, meaning the feature maps before and after this layer are encoded as the same layer. In the other case, feature maps from common layers are directly encoded, as represented in green. All the feature maps across all the convolution layers of the CNN are decoded in one of these ways.

Therefore, in the encoding, we set a binary vector with a length equal to the number of feature maps that need to be encoded. In this binary vector, '0' indicates that the corresponding feature map is removed, while '1' indicates that the corresponding feature map is retained. Fig. 3 provides a graphical representation of an individual. The main limitation of the proposed MOP-FMS is that it is not generally applicable to all CNN architectures. MOP-FMS can be applied only to networks with additive aggregation. For the method to function, the feature maps produced by each layer within the network must depend only on the feature maps produced by the other identical layer. When the feature maps of one layer depend on the feature maps of different layers, this method will not be applicable due to the decoding from genes to the feature map selection of each layer.

Taking the inception block as an example, Fig. 4 represents the feature maps with concatenation. The dependencies among the feature maps are represented as colours. In this case, the final feature maps are stacked by concatenation, meaning they can be divided into four parts. Each part depends on the feature maps of previous layers with same colour. Therefore, obviously, each feature map has a different length, meaning this case cannot be encoded.

In the process of encoding, certain constraints must be satisfied. In the same convolutional layer, encoding all of the feature maps with '0' is not acceptable. If all the feature maps are discarded, the subsequent layers would not be able to receive inputs. Furthermore, we have empirically observed that solutions with too few '1' per layer are likely to display poor performance in terms of accuracy. Conversely, solutions with too many '1' produce very high FLOPs. This means that the number of '1' in each convolution layer must be within a pre-determined range. Therefore, we set several constraints for the convolution layers. Meanwhile, the search space is reduced.

To initialise the population, the following procedure has been applied. The gene of each individual is first generated layer by layer. It is then assumed that the i-th layer has n_i feature maps to be encoded, while the corresponding part of the gene is G_i. One random integer number k_i between the minimum and maximum number of allowed '1' is generated for the i-th layer. Then, in G_i, the number of '1' is k_i, and the number of '0' is $n_i - k_i$. Finally, the gene of the individual is achieved by connecting all G_i where i ranges from 1 to l.

3.2. Crossover, mutation and reparation operators

In this paper, domain-specific crossover and mutation operators are used. In particular, a mask crossover is employed [64]. Two parent individuals, $\mathbf{P}_1$ and $\mathbf{P}_2$, are randomly selected, and a mask vector of length N is also generated. This mask vector contains random numbers between 0 and 1. For each element of the mask vector, if it is greater than a threshold α, then the genes of the corresponding positions in the two parents are swapped to generate two elements of offspring individuals $\mathbf{Q}_1$ and $\mathbf{Q}_2$. The positions with random numbers less than or equal to α do not change values. Fig. 5 gives an example of the crossover.

The mutation operation is then applied to each offspring individual. Another mask vector of random numbers between 0 and 1 is generated. If the element of the mask vector is smaller than a threshold β, then the corresponding gene is flipped from '0' to '1' or vice versa. Fig. 6 gives an example of the mutation operators.

The newly generated individuals then undergo a check to satisfy the constraints, *i.e.*, the feature maps in each layer are scanned. If each layer does not contain enough '1', some genes with '0' are randomly flipped to reach the constraint on the minimum number with '1'. Conversely, if there are too many '1' in a single layer, some of them are randomly selected and flipped to '0' to meet the constraint of having a maximum number of '1'. Fig. 7 shows the reparation operators.

3.3. Decoding and pruning

Each individual is decoded and then the pre-trained network is pruned prior to evaluation. In the decoding process, feature maps of each layer are selected based on each individual. Then, based on the selection of feature maps, the convolutional layers are modified, and some of their kernels and channels are deleted. The

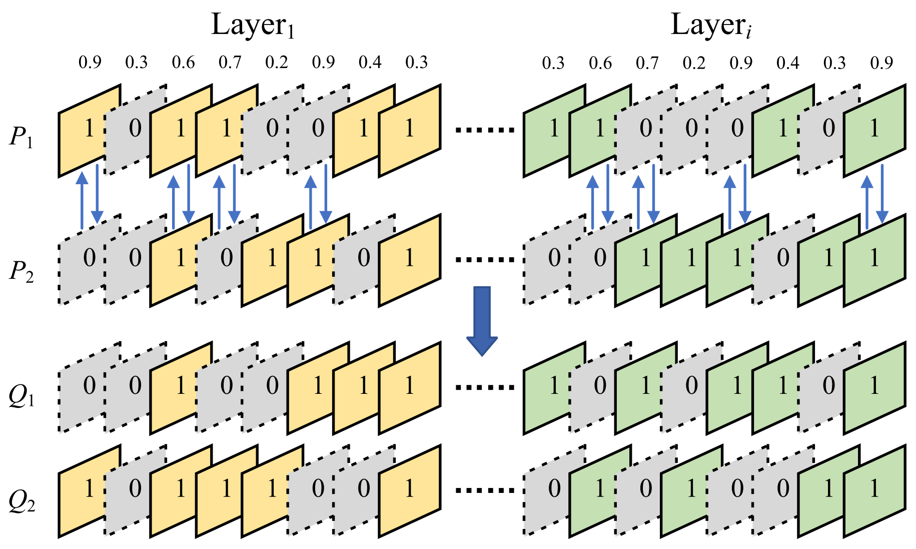

Fig. 5. An example of crossover.

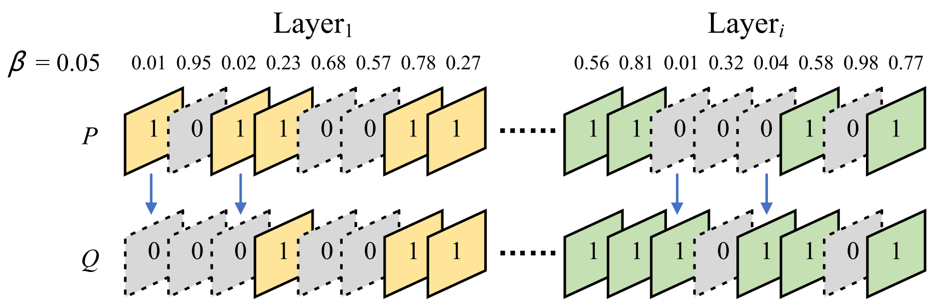

Fig. 6. An example of mutation.

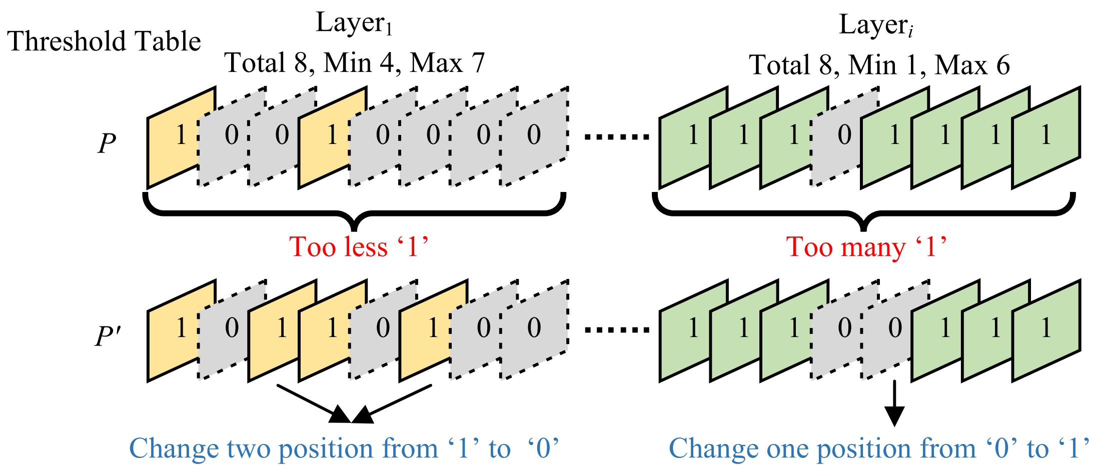

Fig. 7. An example of reparation.

decoding process is the inverse of the encoding process. In this process, the main step is to divide the gene into parts and restore each part to the feature maps of each layer in the network. As is shown in Fig. 8, the individual is divided into a number sections. Then, the feature maps are marked with 'deleted' or 'retained' based on the corresponding gene parts.

To illustrate the pruning progress on the pre-trained networks based on the feature map selection, one pair of pruning operators is shown in Fig. 9 as resulting from the feature map selection of one layer. The symbols F_1, F_2 and F_3 indicate the feature maps. The feature maps F_2 are decoded to produce the selected feature maps F_2'. This affects both previous and subsequent convolutional

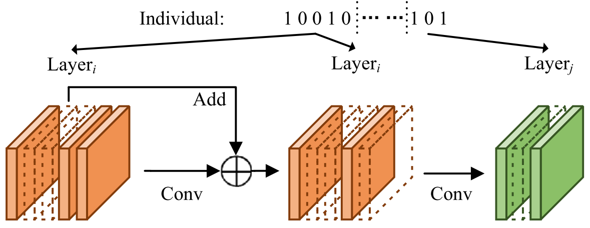

Fig. 8. Decoding process.

layers. Filter pruning is performed on the previous convolutional layer. Based on the removed positions in F_2, which are retained at F_2', the corresponding kernels in the previous convolutional layer are removed, and the other kernels are kept with their weights. Filter pruning is then performed on the next convolutional layer. Each kernel in the next convolutional layer has the same number of channels as F_2. The positions of deleted feature maps in F_2 result in the removal of the corresponding channels in each kernel.

The illustrations in Figs. 8 and 9 provide an intuitive picture of the main benefits of using the proposed encoding based on feature map selection. Direct encoding of the weights to perform the pruning would generate an extremely high-dimensional search space (with up to 10^8 variables), which would make the pruning problem extremely challenging. Although pruning methods directly encoding the positions or channels of filters can be used to avoid this problem, their application may jeopardise the structure of the network and pose an excessive emphasis on weights. The decoding methods adopted by Zhou et al. [31,34] focus on the feature maps but ignore the relations between different layers in residual blocks, meaning the method cannot be adopted on ResNet. The proposed encoding method used in MOP-FMS is highly compact and captures the functional interdependence among the convolution layers, with pruning channels and filters kept together based on the selection of the retrained feature maps, while the relations between the layers are also kept. One idea underpinning the proposed method is that feature maps are of great significance during image classification and can effect the following layers.

3.4. Evaluation and selection

The quality of each newly pruned CNN can be assessed by two objectives. The first objective f_1 is the classification error rate. The second objective f_2 is the computation cost expressed in FLOPs according to the NVIDIA method [65]. For each convolutional layer, the FLOPs are calculated by output size and kernel size as

$$FLOPs = C_o \times W_o \times H_o \times \left(\frac{C_k \times W_k \times H_k}{G_k} + 1 \right) \tag{1}$$

where C_o, W_o and H_o are the number of channels and width and height of the output, respectively. Additionally, C_k, W_k, H_k and G_k are number of channels, width, height and number of groups of the kernel, respectively. For the fully connected layers, the FLOPs are calculated by input size n_{in} and output size n_{out} as Eq. (2).

$$FLOPs = n_{in} \times n_{out} \tag{2}$$

The fitness value f_2 is the sum of the FLOPs at each layer, including the convolutional and fully-connected layers. At the end of each individual evaluation, the fast elitist non-dominated sorting selection method [66] is used to identify those individuals to be retained in the next generation. First, the fast non-dominated sorting is used to divide the population into sets of dominance levels.

Because non-dominant ordering is used, individuals belonging to the same dominance set do not dominate each other. To build the new generation, the best set of dominance (which is not dominated by any other solutions in the population) is taken into consideration. This best set is called the non-dominated set. However, the algorithm needs to select a prearranged number of solutions, which is equal to the population size N_p.

If the non-dominated set does not contain enough solutions to build a population, then other solutions are selected from the set on the next level until the number is satisfied. When the population contains more candidate solutions than the desired population size, the crowded-comparison approach is applied. This approach makes use of the crowding distance, that is, the distance between solutions in the objective space. This criterion selects the solutions that are as sparse as possible within their set of dominance, thus preventing the selection of solutions with similar fitness values [66].

Looking at the method as a whole, it can be observed that MOP-FMS performs regular pruning by using feature maps to

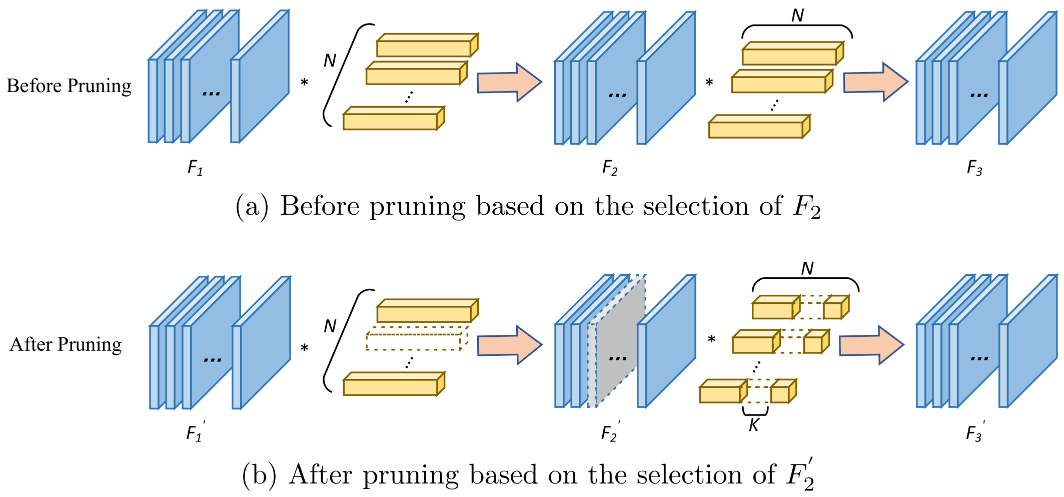
Before Pruning
N
N
*
*
F_1
F_2
F_3
(a) Before pruning based on the selection of F_2
After Pruning
N
N
K
F_1'
F_2'
F_3'
(b) After pruning based on the selection of F_2'

Fig. 9. Illustration of the pruning process on the feature maps in one layer. Parts of F_2 are selected for pruning, as marked by the dashed lines in F_2'. This pruning corresponds to a restructuring of the CNN based on pruning the convolution filters (in yellow) occurring through two mechanisms: channel pruning and filter pruning.

remove some areas of the network to maintain the relationships of the feature maps in each layer.

Algorithm 1 Overall framework of MOP-FMS

```
Input: Number of generations $G$, population size $N_p$, crossover threshold $\alpha$, mutation threshold $\beta$.
Output: The final population $\mathbf{Pop}_G$.
1: $\mathbf{Pop}_0 \leftarrow$ Initialise the population.
2: $\mathbf{FPop} \leftarrow \emptyset$.
3: for each $\mathbf{x}$ in $\mathbf{Pop}_0$ do
4:     $(f_1; f_2) \leftarrow$ Evaluate the fitness of $\mathbf{x}$ (see Section 3.4).
5:     Save $f_1$ and $f_2$ into FPop.
6: end for
7: for $i = 1$ to $G$ do
8:     $\mathbf{QPop}_i \leftarrow$ Crossover and mutation with elements of $\mathbf{Pop}_{i-1}$ as in Section 3.2.
9:     for each $\mathbf{x}$ in $\mathbf{QPop}_i$ do
10:        $(f_1; f_2) \leftarrow$ Evaluate the fitness of $\mathbf{x}$ (see Section 3.4).
11:        Save $f_1$ and $f_2$ into FPop.
12:    end for
13:    $\mathbf{Pop}'_i \leftarrow \mathbf{Pop}_{i-1} \cup \mathbf{QPop}_i$
14:    Apply fast non-dominated sorting, on the basis of $f_1$ and $f_2$ in FPop, to divide $\mathbf{Pop}'_i$ into sets of dominance [66].
15:    Apply the crowded-comparison approach to select from $\mathbf{Pop}'_i$ the $N_p$ solutions composing $\mathbf{Pop}_i$ [66].
16:    Update FPop to match the selected candidate solutions in $\mathbf{Pop}_i$ [66].
17: end for
18: return Final population $\mathbf{Pop}_G$.
```

3.5. Overall framework

The overall framework of the proposed MOP-FMS is outlined in Algorithm 1. As shown, MOP-FMS requires the input of four parameters, total number of generations G, population size N_p, crossover threshold α and mutation threshold β. Firstly, the initial population $\mathbf{Pop}_0$ is composed of binary vectors $\mathbf{x}$ representing the selected feature maps, which are the basis of the following pruning process. For all the candidate individuals $\mathbf{x}$, the fitness values f_1 and f_2 are calculated and stored in **FPop** (lines 1–6).

At each generation, new individuals are generated by the domain-specific crossover and mutation (lines 8) to build an offspring population **QPop**. The fitness values of the newly generated candidate solutions are calculated and recorded into **FPop** (lines 9–12). New individuals are added to the population to form the provisional population $\mathbf{Pop}'_i$. The records of fitness **FPop** and the sets of dominance in $\mathbf{Pop}'_i$ are updated using non-dominated ordering (lines 13–14). The individuals undergoing the following generation are selected (lines 15–16). The population $\mathbf{Pop}_i$ evolves for $i = 1, 2, \ldots, G$ generations to find a non-dominated population $\mathbf{Pop}_G$, which is the final population for feature map selection (lines 7–17).

4. Experiment and results analysis

4.1. Experimental details

In this paper, four popular datasets were used for the experiments: MNIST [67], Fashion MNIST [68], CIFAR-10 [69] and CIFAR-100 [69]. MNIST and Fashion MNIST are two datasets entirely composed of grey images with a size of 28×28. Each of the two datasets contained a training set with 60,000 images and a test set with 10,000 images. The data from both these datasets belonged to 10 categories. In addition, the images of the two datasets were transformed into 32×32 images to match the input size required by ConvNet-3 and LeNet-5. Both CIFAR-10 and CIFAR-100 are datasets composed of colour images with a size of 32×32. Each dataset contained a training set with 50,000 images and a test set with 10,000 images. The images in CIFAR-10 were divided into 10 categories, while those in CIFAR-100 were divided into 100 categories.

To demonstrate the effectiveness of the proposed method, some commonly used CNNs were selected for comparison. These included:

ConvNet-3, LeNet-5 [70], VGG [8] and ResNet [4]. ConvNet-3 was designed for this study by modifying LeNet-5. ConvNet-3 contains only three convolution layers and no fully connected layers. ConvNet-3 and LeNet-5 were used for experiments on MNIST and Fashion MNIST, while VGG and ResNet were used for experiments on the CIFAR datasets.

For each pruning experiment, MOP-FMS was run for $G = 50$ generations, with a population size of $N_p = 20$, a crossover threshold of $\alpha = 0.5$ and a mutation threshold of $\beta = 0.05$. The minimum number of retained or deleted feature maps for each layer was set to $\frac{1}{16}$ of the original number, *i.e.*, we impose that in each binary solution at least $\frac{1}{16}$ and at most $\frac{15}{16}$ of the total bits/genes are '1'. All individuals from the last generation of the population were achieved for final training. After feature map selection, each individual had an accuracy rate of 60%–80%. Then, they were trained for 20 epochs using the output of the original model and fine-tuned with small learning rates.

In all the experiments, one-fifth of the images were randomly selected from the training set to evaluate the accuracy of each sub-network generated by decoding each individual without training. The remaining images from the training set were used to train the pruned networks, which were decoded with final individuals after the population iteration. The test set was used by the trained networks to obtain the final results. Since the first objective was unable to ensure that the best individual achieved higher accuracy than the others after training, all the pruned networks were decoded from individuals in the final population need training.

4.2. Ablation study

To demonstrate the potential performance of the proposed method, we used the multi-objective evolutionary computation technology to prune LeNet-5 and ConvNet-3 on MNIST and Fashion MNIST. Each network was pruned with different pruning rate levels, which are given by the formulas

$$PR_{FLOPs} = \frac{n_{FLOPs}}{N_{FLOPs}} \tag{3}$$

$$PR_{param} = \frac{n_{param}}{N_{param}} \tag{4}$$

where PR is the pruning rate on FLOPs and parameters, n represents the number of saved FLOPs and parameters after pruning and N represents the number of FLOPs and parameters of the original network.

Hence, we can refer to the pruning rate with respect to FLOPs or parameters. It is worthwhile to mention that the pruning rate in terms of FLOPs is the outcome of the optimisation process as it is embedded in f_2 (see Section 3.4). The pruning rate in terms of parameters is indirectly controlled by the constraints when a minimum and maximum number of '1' is chosen.

Table 1 shows the results of LeNet-5 before pruning (baseline) and after the application of MOP-FMS with an increasing pruning rate. For each CNN, the values of accuracy, pruning rate in terms of FLOPs and pruning rate in terms of parameters are shown. The numerical results in Table 1 show that MOP-FMS was able to efficiently prune the original CNN on the two small datasets

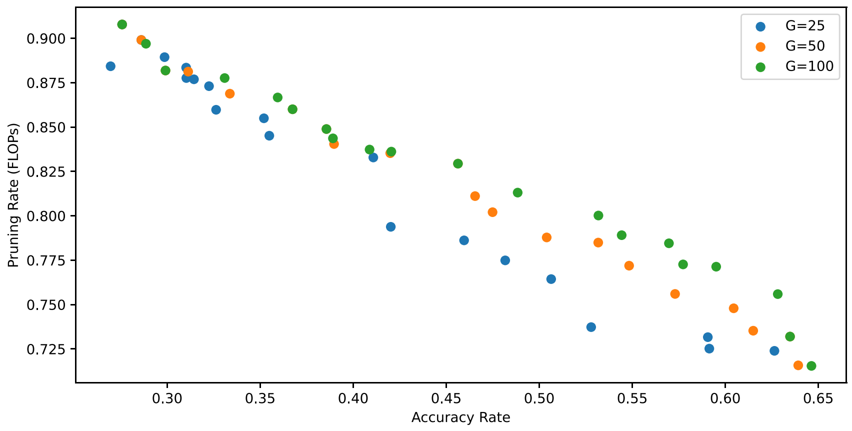

Fig. 10. The effect of the generations (G) on the detected non-dominated set.

Table 1
Experiments with LeNet-5 pruned by MOP-FMS. For MNIST, results with FLOPs pruning rates of 10%, 40% and 70% were evenly selected from the final population. For Fashion MNIST, the results with FLOPs pruning rates of 10%, 30% and 40% were evenly selected from the final population.

Dataset	Method	Accuracy (%)	Pruning rate (%)	
			FLOPs	Parameters
MNIST	LeNet-5 (BL)	99.49%	0%	0%
	LeNet-5 (10%)	**99.50%**	14.28%	0.69%
	LeNet-5 (40%)	99.45%	42.84%	2.07%
	LeNet-5 (70%)	99.28%	**71.41%**	**3.45%**
Fashion MNIST	LeNet-5 (BL)	91.54%	0%	0%
	LeNet-5 (10%)	**91.95%**	14.28%	0.69%
	LeNet-5 (30%)	91.63%	28.56%	1.38%
	LeNet-5 (40%)	91.49%	**42.84%**	**2.07%**

'BL' indicates the baseline network, and it has the same meaning in the following tables.

Table 2
Experiments with ConvNet-3 pruned by MOP-FMS. For MNIST, the results with FLOPs pruning rates of 20%, 40%, 60% and 80% were evenly selected from the final population. For Fashion MNIST, the results with FLOPs pruning rates of 10%, 20%, 40% and 60% are evenly selected from the final population.

Dataset	Method	Accuracy (%)	Pruning rate (%)	
			FLOPs	Parameters
MNIST	ConvNet-3 (BL)	99.04%	0%	0%
	ConvNet-3 (20%)	**99.17%**	16.68%	24.36%
	ConvNet-3 (40%)	98.85%	37.36%	40.74%
	ConvNet-3 (60%)	98.48%	55.33%	55.60%
	ConvNet-3 (80%)	97.70%	**78.74%**	**67.05%**
Fashion MNIST	ConvNet-3 (BL)	89.66%	0%	0%
	ConvNet-3 (10%)	**89.74%**	4.17%	6.09%
	ConvNet-3 (20%)	89.16%	23.40%	17.89%
	ConvNet-3 (40%)	88.27%	41.26%	28.94%
	ConvNet-3 (60%)	87.13%	**62.04%**	**49.13%**

without greatly affecting the classification accuracy. It was also observed that relatively small variations in the network structure may yield a major reduction in the computation cost.

For example, for MNIST, a reduction of only 3.45% of the parameters corresponds to a reduction of 71.41% for the FLOPs. This finding has to be understood in relation to the architecture of LeNet-5. Since the network has only one layer of feature maps that can be selected, the number of parameters to be pruned is limited. On both the MNIST and Fashion MNIST datasets, the accuracy loss caused by pruning was within 0.2%. These results highlight the potential of MOP-FMS for real-time applications and further justifies our choice of linking the FLOPs as one of the objectives.

Table 2 displays the results of ConvNet-3. Since ConvNet-3 has three layers of feature maps that can be pruned without fully connected layers, it can achieve higher pruning rates than those of LeNet-5 without an excessive loss of accuracy. Although the accuracy of the benchmark structure is not very high due to the absence of fully connected layers, MOP-FMS still performed very well considering that there was an accuracy loss of only approximately 0.3%.

Comparing Tables 1 and 2, it was found that MOP-FMS is more suited to networks that have a greater proportion of convolution layers in their structure. In such CNNs, MOP-FMS can achieve a greater pruning rate on the FLOPs and parameters. To display the influence of the set parameters on the performance, we here report an ablation study on the generation number (G), population size (P) and threshold values (α and β). These experiments are performed with VGG-16 on CIFAR-10. First, the number of generations was set to 25, 50 and 100. As shown in Fig. 10, when $G = 50$, the non-dominated set was slightly worse than that corresponding to $G = 100$ but much better than that corresponding to $G = 25$. Considering that $G = 100$ would require runs twice as long in time than that for $G = 50$, we concluded that setting G to 50 was a reasonable compromise. Then, the size of the population P was extended to 60 and 100 to show its influence on performance. In Fig. 11, green and yellow dots do not approximate the Pareto front better than the blue ones. Hence, an increase in population size cannot effectively improve the coverage of the objective space. Finally, to demonstrate the necessity of constraints, the experiments have been repeated with and without constraints. From Fig. 12, it can be seen that, although the distribution of yellow dots (without constraints) in the objective space was not as large as that of blue dots (with constraints) due to the initialisation, the search results of the yellow dots were far worse than those of blue dots.

4.3. Pruning performances on VGG

This subsection compares the performance of MOP-FMS against those of other pruning algorithms for pruning VGG-16 [8] on CIFAR-10 and VGG-19 [8] on CIFAR-100.

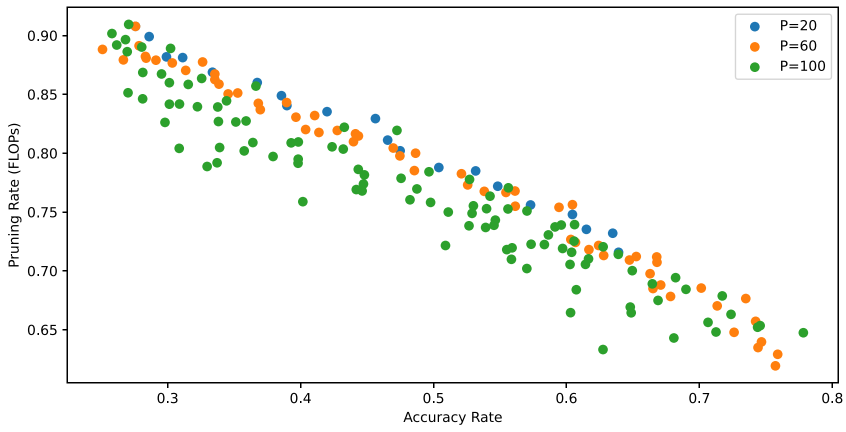

Fig. 11. The effect of the population size on the detected non-dominated set.

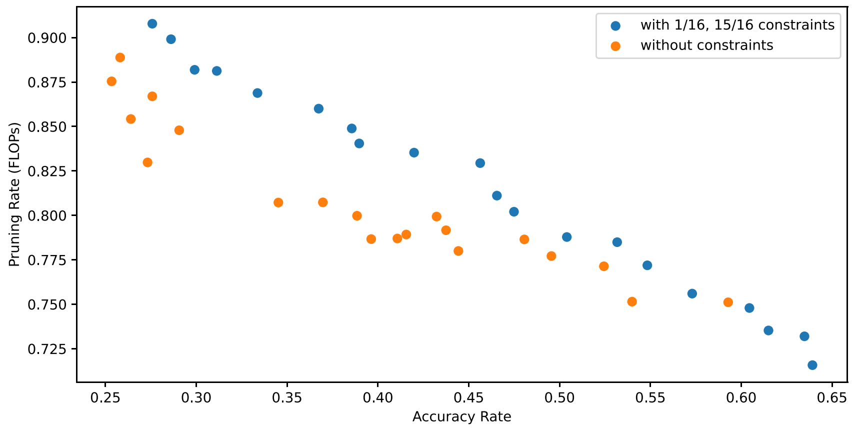

Fig. 12. The effect of the proposed constraints on the detected non-dominated set.

For CIFAR-10, the following pruning algorithms were used for comparison: Variational CNN Pruning [71], Greg-1 [32], GAL-0.1 [36], GAL-0.05 [36], SSS [72], HRank [73], KGEA [35], ThiNet [74] and ABCPruner [75]. These algorithms were pruned on VGG-16 [8] for the classification of CIFAR-10. Table 3 shows the results of the comparison on CIFAR-10.

The experimental results in Table 3 show that pruning rates of over 60% can achieve an accuracy superior to that of all other pruning algorithms with VGG-16 as the baseline. Additionally, compared with other algorithms, MOP-FMS was also obtain to obtain the highest pruning rate, achieving reductions of approximately 90% in terms of both FLOPs and the number of parameters.

To depict the superiority of MOP-FMS compared with its competitors, the results of all the comparison algorithms shown in Table 3 are plotted in Fig. 13 along with the results of the last population of the pruned networks $\mathbf{P}_G$ after fine-turning. The search time for VGG-16 on CIFAR-10 was about 60 min, and the training time for individuals of the last population was within 80 min. It can be seen from the figure that the solutions detected by MOP-FMS dominated all the solutions produced by the 12 competitors except for KGEA [35]. Only one solution in $\mathbf{P}_G$ was dominated by KGEA, but it was very close to its performance. However, in KGEA, Zhou et al. set the size of population to 30, while the number of generations for VGG is 200, which is about six times more computational budget than that used for MOP-FMS.

Table 3
Experiments on CIFAR-10 with VGG-16. The results with a FLOPs pruning rate of 60%, 70%, 80% and 90% were evenly selected from the final population.

Method	Accuracy (%)	Pruning rate (%)	
		FLOPs	Parameters
VGG-16 (BL) [8]	93.83%	0.00%	0.00%
FPC (50%) [9]	94.09%	53.92%	91.24%
FPC (60%) [9]	94.08%	65.63%	95.18%
Variational CNN Pruning [71]	93.18%	39.44%	73.37%
Greg-1 (70%) [32]	93.52%	70.36%	65.33%
Greg-1 (30%) [32]	93.40%	34.34%	63.32%
GAL-0.1 [36]	90.73%	45.21%	81.86%
GAL-0.05 [36]	93.03%	39.60%	77.17%
SSS [72]	93.02%	41.63%	73.30%
HRank (50%) [73]	93.43%	53.59%	82.95%
HRank (60%) [73]	92.34%	65.38%	82.07%
KGEA [35]	92.83%	78.88%	61.17%
ThiNet [74]	93.47%	70.13%	58.86%
ABCPruner [75]	93.35%	66.54%	52.73%
MOP-FMS (60%)	**94.36%**	64.60%	62.18%
MOP-FMS (70%)	93.93%	73.80%	72.25%
MOP-FMS (80%)	91.51%	82.45%	80.50%
MOP-FMS (90%)	87.24%	**90.50%**	**89.08%**

The experimental results of pruning VGG-19 on CIFAR-100 are shown in Table 4. The search time for VGG-19 on CIFAR-100

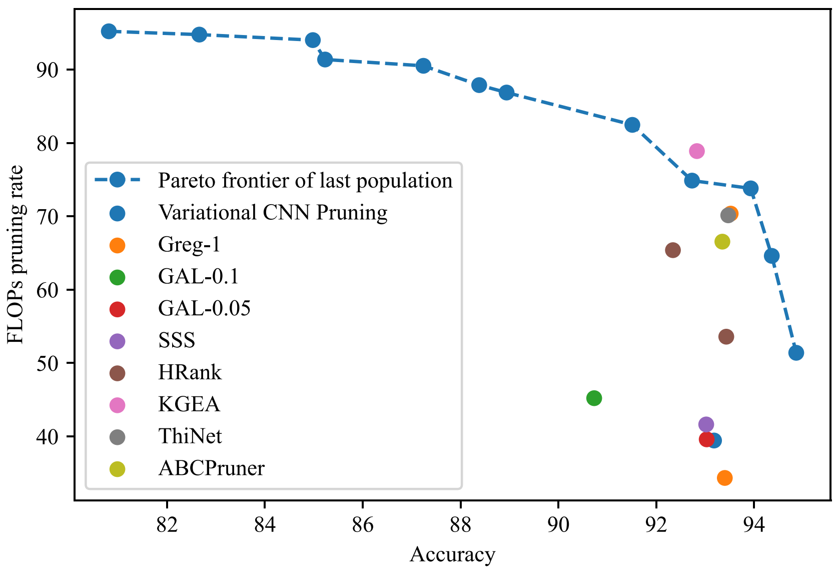

Fig. 13. Pareto frontier of last generation population compared with existing methods.

Table 4
Experiments on CIFAR-100 with VGG-19. The results with a FLOPs pruning rate of 50%, 60% and 70% were evenly selected from the final population.

Method	Accuracy (%)	Pruning rate (%)	
		FLOPs	Parameters
VGG-19 (BL) [8]	73.34%	0%	0%
Kron-OBD (70%) [76] (Extended based on [77])	72.29%	77.24%	37.90%
Kron-OBD (90%) [76] (Extended based on [77])	60.70%	**97.56%**	82.55%
Kron-OBS (70%) [76] (Extended based on [78])	72.12%	74.18%	36.59%
Kron-OBS (90%) [76] (Extended based on [78])	60.66%	97.48%	83.57%
EigenDamage (70%) [76]	72.90%	76.64%	37.40%
EigenDamage (90%) [76]	65.18%	97.31%	**88.63%**
MOP-FMS (50%)	**72.92%**	52.59%	49.18%
MOP-FMS (60%)	70.12%	66.90%	64.86%
MOP-FMS (70%)	65.68%	69.23%	65.76%

was about 80 min, while the training time for the individuals of the last population was within 130 min. In this case, MOP-FMS was compared against Kron-OBS [76], Kron-OBD [76] and EigenDamage [76].

The results show that MOP-FMS can achieve the best accuracy level with pruning rates of around 50%. Although MOP-FMS is generally competitive with other methods, it does not seem to produce high accuracy for higher levels of pruning rates. Considering that the effect of the baseline on CIFAR100 was not very good, it can be guessed that the feature map in VGG19 is more important for the final classification basis, thus making it difficult to improve the accuracy.

To show a qualitative representation of the convergence of the algorithm, Fig. 14 displays one example of the performance trend of both the objectives (Accuracy and FLOPS) over the generations. The oscillations in both trends were due to the fact that the algorithm applied a multi-objective approach and did not attempt to optimise the functions separately.

4.4. Pruning performances on ResNet

To show the pruning ability of MOP-FMS on complex structures, MOP-FMS was used to prune ResNet-56 [4] and ResNet-110 [4]. The comparison results are presented in Tables 5 and 6, with FPC [9], NISP [79], Greg-1 [32], GAL-0.5 [36] and HRank [73] used for comparison.

The proposed MOP-FMS pruned a large number of network structures on each benchmark network without greatly affecting the accuracy and while still greatly reducing the computational cost. During the search time, ResNet-56 originally costed about 25 min, and ResNet-110 costed about 50 min. Afterwards, individuals in the last population were trained, making the entire time for ResNet-56 within 40 min and that for ResNet-56 within 90 min.

Table 5
Experiments on CIFAR-10 with ResNet-56. The results with FLOPs pruning rates of 60%, 70% and 80% were evenly selected from the final population.

Method	Accuracy (%)	Pruning rate (%)	
		FLOPs	Parameters
ResNet-56 (BL) [4]	93.78%	0.00%	0.00%
FPC (30%) [9]	**94.01%**	35.72%	65.88%
FPC (50%) [9]	93.39%	49.74%	**78.82%**
NISP [79]	93.01%	36.13%	42.35%
Greg-1 [32]	93.06%	28.33%	14.12%
GAL-0.6 [36]	92.98%	38.26%	11.76%
HRank (30%) [73]	93.52%	30.05%	16.47%
HRank (50%) [73]	93.17%	50.55%	42.35%
ESNB [31]	93.75%	53.17%	56.47%
MOP-FMS (60%)	93.98%	64.78%	68.92%
MOP-FMS (70%)	93.47%	69.91%	72.28%
MOP-FMS (80%)	93.01%	**76.26%**	76.90%

Table 6
Experiments on CIFAR-10 with ResNet-110. The results with FLOPs pruning rates of 60%, 70% and 80% were evenly selected from the final population.

Method	Accuracy (%)	Pruning rate (%)	
		FLOPs	Parameters
ResNet-110 (BL) [4]	94.43%	0.00%	0.00%
FPC (50%) [9]	**94.54%**	48.20%	67.05%
FPC (60%) [9]	93.83%	64.54%	83.24%
Greg-1 [32]	93.30%	39.28%	32.95%
GAL-0.5 [36]	92.55%	49.00%	45.09%
HRank (50%) [73]	94.23%	41.75%	39.88%
HRank (60%) [73]	93.36%	58.59%	59.54%
MOP-FMS (60%)	94.47%	67.32%	69.38%
MOP-FMS (70%)	94.01%	71.35%	72.10%
MOP-FMS (80%)	93.28%	**83.25%**	**86.14%**

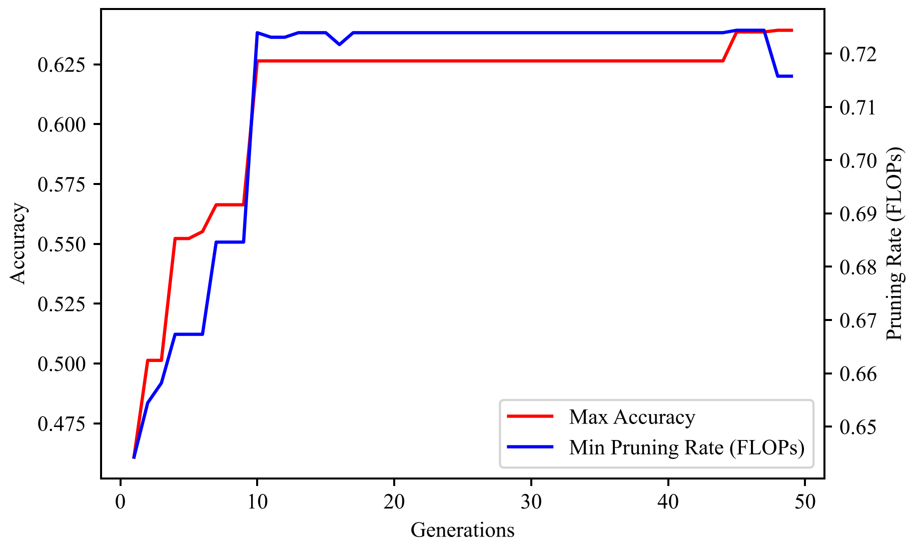

Fig. 14. Convergence trend of the two objectives for VGG-19 on CIFAR-10.

It is worthwhile to comment on the correlation between ResNet structure and MOP-FMS performance. The logic of the proposed MOP-FMS appears to be very well-suited to efficiently pruning the ResNet structure. Due to the fact that the same feature maps within the ResNet structure were used in multiple parts, the MOP-FMS can prune large portions of the network by eliminating only a few feature maps. As shown in Tables 5 and 6, FPC [9] achieved the best accuracy. For MOP-FMS, for a 60% level of the pruned network, the accuracy was similar to that of FPC, but the pruning rate was much higher.

5. Conclusions and future work

This paper proposes a novel pruning algorithm for CNNs with additive aggregation. Unlike other pruning algorithms, the proposed method encodes the pruning task on CNNs as the selection of feature maps. This representation of the solutions enables a highly compact and efficient representation of the network pruning. The pruning task is modelled as a bi-objective optimisation problem where the two objectives are the accuracy of the pruned network and the computation cost, which is expressed in FLOPs. An ad-hoc evolutionary multi-objective optimisation algorithm was also designed to perform the pruning.

We chose four popular CNNs as the baselines and conducted experiments on four datasets. In total, 14 modern pruning algorithms were used for comparison. The results demonstrated that the proposed method was able to achieve excellent performance, especially for the ResNet architecture, because the encoding of the feature map selection is well suited to this structure.

Based on our interpretation, since the deletion of a feature map may result in the pruning of multiple sections of a ResNet, the feature map encoding carries a large amount of important information about the network functioning. The design of the two competing objective functions guarantees that the pruning does not excessively deteriorate the performance and poses an emphasis on the computation cost. The latter is highly important, especially for modern applications that might require real-time image classification despite the fact that the algorithms are embedded in devices with limited hardware. Future work will investigate methods to generalise the applicability of the proposed concept to all CNN architectures.

Future developments of this research will move in two directions. The first is the study and implementation of pruning methods based on feature map representation for large networks such as TinyImageNet and ImageNet. This line of research will also include studies on the scalability of the method. The second is the adaptation of our approach to lightweight network structures, such as MobileNet and ShuffleNet, whose pruning processes may differ from those applicable in the case of standard-size networks. In addition, the relationship between the feature maps of different layers requires further exploration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

This work was partially supported by the National Natural Science Foundation of China (61876089, 61876185, 61902281), the Natural Science Foundation of Jiangsu Province, China (BK20141005) and the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (14KJB520025).

References

[1] Y. Chen, X. Wen, Y. Zhang, W. Shi, CCPrune: Collaborative channel pruning for learning compact convolutional networks, Neurocomputing 451 (2021) 35–45.
[2] Z. Lu, K. Deb, V.N. Boddeti, MUXConv: Information multiplexing in convolutional neural networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 12041–12050.
[3] L. Zhang, X. Hu, Y. Zhou, G. Zhou, S. Duan, Memristive DeepLab: A hardware friendly deep CNN for semantic segmentation, Neurocomputing 451 (2021) 181–191.
[4] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016, pp. 770–778.
[5] C. Szegedy, S. Ioffe, V. Vanhoucke, A.A. Alemi, Inception-v4, inception-ResNet and the impact of residual connections on learning, in: Proceedings of the AAAI Conference on Artificial Intelligence, 2017, pp. 4278–4284.
[6] E. Mininno, F. Neri, F. Cupertino, D. Naso, Compact differential evolution, IEEE Trans. Evol. Comput. 15 (1) (2011) 32–54.
[7] F. Neri, E. Mininno, Memetic compact differential evolution for cartesian robot control, IEEE Comput. Intell. Mag. 5 (2) (2010) 54–65.
[8] K. Simonyan, A. Zisserman, Very deep convolutional networks for large-scale image recognition, in: International Conference on Learning Representations, 2015.
[9] Y. Chen, X. Wen, Y. Zhang, Q. He, FPC: Filter pruning via the contribution of output feature map for deep convolutional neural networks acceleration, Knowl.-Based Syst. 238 (2022) 107876.

[10] G. Lee, K. Lee, DNN compression by ADMM-based joint pruning, Knowl.-Based Syst. 239 (2022) 107988.

[11] Y. Liu, M.K. Ng, Deep neural network compression by Tucker decomposition with nonlinear response, Knowl.-Based Syst. 241 (2022) 108171.

[12] T. Gao, Y. Zhou, S. Duan, X. Hu, Memristive KDG-BNN: Memristive binary neural networks trained via knowledge distillation and generative adversarial networks, Knowl.-Based Syst. 249 (2022) 108962.

[13] J. Yang, X. Shen, J. Xing, X. Tian, H. Li, B. Deng, J. Huang, X.-s. Hua, Quantization Networks, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 7300–7308.

[14] Z. Lu, G. Sreekumar, E. Goodman, W. Banzhaf, K. Deb, V.N. Boddeti, Neural architecture transfer, IEEE Trans. Pattern Anal. Mach. Intell. 43 (9) (2021) 2971–2989.

[15] Y. Xue, Y. Wang, J. Liang, A. Slowik, A self-adaptive mutation neural architecture search algorithm based on blocks, IEEE Comput. Intell. Mag. 16 (3) (2021) 67–78.

[16] P. Chen, S. Liu, H. Zhao, J. Jia, Distilling knowledge via knowledge review, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2021, pp. 5006–5015.

[17] Y. Sun, B. Xue, M. Zhang, G.G. Yen, J. Lv, Automatically designing CNN architectures using the genetic algorithm for image classification, IEEE Trans. Cybern. 50 (9) (2020) 3840–3854.

[18] Y. Sun, B. Xue, M. Zhang, G.G. Yen, Evolving deep convolutional neural networks for image classification, IEEE Trans. Evol. Comput. 24 (2) (2019) 394–407.

[19] Y. Xue, J. Qin, Partial connection based on channel attention for differentiable neural architecture search, IEEE Trans. Ind. Inform. (2022) http://dx.doi.org/10.1109/TII.2022.3184700.

[20] S. Han, J. Pool, J. Tran, W. Dally, Learning both weights and connections for efficient neural network, in: Advances in Neural Information Processing Systems, vol. 28, 2015.

[21] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, C. Zhang, Learning efficient convolutional networks through network slimming, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 2755–2763.

[22] H. Cai, C. Gan, T. Wang, Z. Zhang, S. Han, Once-for-All: Train one network and specialize it for efficient deployment, in: International Conference on Learning Representations, 2019.

[23] Y. Han, G. Huang, S. Song, L. Yang, H. Wang, Y. Wang, Dynamic neural networks: A survey, IEEE Trans. Pattern Anal. Mach. Intell. 44 (11) (2022) 7436–7456.

[24] B. Yang, G. Bender, Q.V. Le, J. Ngiam, CondConv: Conditionally parameterized convolutions for efficient inference, in: Advances in Neural Information Processing Systems, vol. 32, 2019.

[25] N. Ma, X. Zhang, J. Huang, J. Sun, Weightnet: Revisiting the design space of weight networks, in: European Conference on Computer Vision, Cham, 2020, pp. 776–792.

[26] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M.A. Horowitz, W.J. Dally, EIE: Efficient inference engine on compressed deep neural network, in: Proceedings of the 43rd International Symposium on Computer Architecture, 2016, pp. 243–254.

[27] W. Niu, X. Ma, S. Lin, S. Wang, X. Qian, X. Lin, Y. Wang, B. Ren, PatDNN: Achieving real-time DNN execution on mobile devices with pattern-based weight pruning, in: Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020, pp. 907–922.

[28] C. Hong, A. Sukumaran-Rajam, I. Nisa, K. Singh, P. Sadayappan, Adaptive sparse tiling for sparse matrix multiplication, in: Proceedings of the 24th Symposium on Principles and Practice of Parallel Programming, 2019, pp. 300–314.

[29] E. Elsen, M. Dukhan, T. Gale, K. Simonyan, Fast sparse ConvNets, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 14617–14626.

[30] A. Zhou, Y. Ma, J. Zhu, J. Liu, Z. Zhang, K. Yuan, W. Sun, H. Li, Learning N:M fine-grained structured sparse neural networks from scratch, in: International Conference on Learning Representations, 2021.

[31] Y. Zhou, G.G. Yen, Z. Yi, Evolutionary shallowing deep neural networks at block levels, IEEE Trans. Neural Netw. Learn. Syst. (2021) http://dx.doi.org/10.1109/TNNLS.2021.3059529.

[32] H. Li, A. Kadav, I. Durdanovic, H. Samet, H.P. Graf, Pruning filters for efficient ConvNets, in: International Conference on Learning Representations, 2017.

[33] Y. Zhang, Y. Zhen, Z. He, G.G. Yen, Improvement of efficiency in evolutionary pruning, in: Proceedings of the International Joint Conference on Neural Networks, 2021, pp. 1–8.

[34] Y. Zhou, G.G. Yen, Z. Yi, Evolutionary compression of deep neural networks for biomedical image segmentation, IEEE Trans. Neural Netw. Learn. Syst. 31 (8) (2020) 2916–2929.

[35] Y. Zhou, G.G. Yen, Z. Yi, A knee-guided evolutionary algorithm for compressing deep neural networks, IEEE Trans. Cybern. 51 (3) (2021) 1626–1638.

[36] 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: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2785–2794.

[37] J. Frankle, M. Carbin, The lottery ticket hypothesis: Finding sparse, trainable neural networks, in: International Conference on Learning Representations, 2019.

[38] J. Frankle, G. Dziugaite, D.M. Roy, M. Carbin, Linear mode connectivity and the lottery ticket hypothesis, in: International Conference on Machine Learning, 2020, pp. 3259–3269.

[39] E. Malach, G. Yehudai, S. Shalev-Shwartz, O. Shamir, Proving the lottery ticket hypothesis: Pruning is all you need, in: International Conference on Machine Learning, 2020.

[40] H. Hu, R. Peng, Y.-W. Tai, C.-K. Tang, Network trimming: A data-driven neuron pruning approach towards efficient deep architectures, 2016, arXiv preprint arXiv:1607.03250.

[41] Y. He, X. Zhang, J. Sun, Channel pruning for accelerating very deep neural networks, in: Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 1398–1406.

[42] H. Wang, C. Qin, Y. Zhang, Y. Fu, Neural pruning via growing regularization, in: International Conference on Learning Representations, 2021.

[43] Y. He, G. Kang, X. Dong, Y. Fu, Y. Yang, Soft filter pruning for accelerating deep convolutional neural networks, in: Proceedings of the International Joint Conference on Artificial Intelligence, 2018, pp. 2234–2240.

[44] M. Lin, Y. Zhang, Y. Li, B. Chen, F. Chao, M. Wang, S. Li, Y. Tian, R. Ji, 1xN Pattern for Pruning Convolutional Neural Networks, IEEE Trans. Pattern Anal. Mach. Intell. (2022) 1–11.

[45] K. Nakagawa, S. Suzumura, M. Karasuyama, K. Tsuda, I. Takeuchi, Safe Pattern Pruning: An Efficient Approach for Predictive Pattern Mining, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 2016, pp. 1785–1794.

[46] P. Xu, J. Cao, F. Shang, W. Sun, P. Li, Layer Pruning via Fusible Residual Convolutional Block for Deep Neural Networks, 2020, arXiv preprint arXiv:2011.14356.

[47] K. Yamamoto, K. Maeno, PCAS: Pruning channels with attention statistics for deep network compression, in: Proceedings of the British Machine Vision Conference (BMVC), 2019, pp. 106.1–106.13.

[48] T. He, Y. Fan, Y. Qian, T. Tan, K. Yu, Reshaping deep neural network for fast decoding by node-pruning, in: 2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2014, pp. 245–249.

[49] H. He, J. Liu, Z. Pan, J. Cai, J. Zhang, D. Tao, B. Zhuang, Pruning Self-attentions into Convolutional Layers in Single Path, 2021, arXiv preprint arXiv:2111.11802.

[50] Y. Li, P. Zhao, G. Yuan, X. Lin, Y. Wang, X. Chen, Pruning-as-search: Efficient neural architecture search via channel pruning and structural reparameterization, 2022, arXiv preprint arXiv:2206.01198.

[51] H. Mousavi, M. Loni, M. Alibeigi, M. Daneshtalab, PR-DARTS: Pruning-based differentiable architecture search, 2022, arXiv preprint arXiv:2207.06968.

[52] X. Dong, Y. Yang, Network pruning via transformable architecture search, in: Advances in Neural Information Processing Systems vol. 32, 2019, pp. 760–771.

[53] J. Wei, G. Zhu, Z. Fan, J. Liu, Y. Rong, J. Mo, W. Li, X. Chen, Genetic U-Net: Automatically designed deep networks for retinal vessel segmentation using a genetic algorithm, IEEE Trans. Med. Imaging 41 (2) (2022) 292–307.

[54] H. Li, N. Liu, X. Ma, S. Lin, S. Ye, T. Zhang, X. Lin, W. Xu, Y. Wang, ADMM-based weight pruning for real-time deep learning acceleration on mobile devices, in: Proceedings of the 2019 on Great Lakes Symposium on VLSI, New York, NY, USA, 2019, pp. 501–506.

[55] T. Zhang, S. Ye, K. Zhang, J. Tang, W. Wen, M. Fardad, Y. Wang, A systematic DNN weight pruning framework using alternating direction method of multipliers, in: Proceedings of the European Conference on Computer Vision, 2018, pp. 191–207.

[56] M. Cacciola, A. Frangioni, X. Li, A. Lodi, Deep neural networks pruning via the structured perspective regularization, 2022, arXiv preprint arXiv:2206.14056.

[57] Y. Bi, B. Xue, M. Zhang, Multi-objective genetic programming for feature learning in face recognition, Appl. Soft Comput. 103 (2021) 107152.

[58] Y. Bi, B. Xue, P. Mesejo, S. Cagnoni, M. Zhang, A survey on evolutionary computation for computer vision and image analysis: Past, present, and future trends, IEEE Trans. Evol. Comput. (2022) 1.

[59] B. Xue, M. Zhang, W.N. Browne, X. Yao, A survey on evolutionary computation approaches to feature selection, IEEE Trans. Evol. Comput. 20 (4) (2015) 606–626.

[60] P. Wang, B. Xue, J. Liang, M. Zhang, Differential evolution based feature selection: A niching-based multi-objective approach, IEEE Trans. Evol. Comput. (2022) 1–4.
[61] Y. Xue, P. Jiang, F. Neri, J. Liang, A multi-objective evolutionary approach based on Graph-in-Graph for neural architecture search of convolutional neural networks, Int. J. Neural Syst. 31 (09) (2021) 2150035.
[62] Y. Liu, Y. Sun, B. Xue, M. Zhang, G. Yen, A survey on evolutionary neural architecture search, IEEE Trans. Neural Netw. Learn. Syst. (2021) 1–21.
[63] W. Rawat, Z. Wang, Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review, Neural Comput. 29 (9) (2017) 2352–2449.
[64] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Second Edition, in: Natural Computing Series, Springer, 2015.
[65] P. Molchanov, S. Tyree, T. Karras, T. Aila, J. Kautz, Pruning convolutional neural networks for resource efficient inference, in: International Conference on Learning Representations, 2017.
[66] K. Deb, A. Pratap, S. Agarwal, T. Meyarivan, A fast and elitist multiobjective genetic algorithm: NSGA-II, IEEE Trans. Evol. Comput. 6 (2) (2002) 182–197.
[67] MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges, URL http://yann.lecun.com/exdb/mnist/.
[68] H. Xiao, K. Rasul, R. Vollgraf, Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017, arXiv preprint arXiv:1708.07747.
[69] The CIFAR-10 and CIFAR-100 datasets, URL http://www.cs.toronto.edu/~kriz/cifar.html.
[70] Y. LeCun, et al., LeNet-5, convolutional neural networks, URL: http://yann. lecun. com/exdb/lenet, 20 (5) 14.
[71] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, Q. Tian, Variational convolutional neural network pruning, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2019, pp. 2775–2784.
[72] Z. Huang, N. Wang, Data-driven sparse structure selection for deep neural networks, in: Proceedings of the European Conference on Computer Vision, 2018, pp. 317–334.
[73] M. Lin, R. Ji, Y. Wang, Y. Zhang, B. Zhang, Y. Tian, L. Shao, HRank: Filter pruning using high-rank feature map, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 1526–1535.
[74] J.-H. Luo, H. Zhang, H.-Y. Zhou, C.-W. Xie, J. Wu, W. Lin, ThiNet: Pruning CNN filters for a thinner net, IEEE Trans. Pattern Anal. Mach. Intell. 41 (10) (2019) 2525–2538.
[75] 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, 2021.
[76] C. Wang, R. Grosse, S. Fidler, G. Zhang, EigenDamage: Structured pruning in the Kronecker-Factored eigenbasis, in: International Conference on Machine Learning, vol. 97, 2019, pp. 6566–6575.
[77] Y. LeCun, J. Denker, S. Solla, Optimal brain damage, in: Advances in Neural Information Processing Systems, vol. 2, 1989.
[78] B. Hassibi, D. Stork, Second order derivatives for network pruning: Optimal brain surgeon, in: Advances in Neural Information Processing Systems, vol. 5, 1992.
[79] R. Yu, A. Li, C.-F. Chen, J.-H. Lai, V.I. Morariu, X. Han, M. Gao, C.-Y. Lin, L.S. Davis, NISP: Pruning networks using neuron importance score propagation, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 9194–9203.