Effective Model Compression via Stage-wise Pruning

Ming-Yang Zhang    Xin-Yi Yu    Lin-Lin Ou
School of Information Engineering, Zhejiang University of Technology, Hangzhou 310000, China

Abstract: Automated machine learning (AutoML) pruning methods aim at searching for a pruning strategy automatically to reduce the computational complexity of deep convolutional neural networks (deep CNNs). However, some previous work found that the results of many Auto-ML pruning methods cannot even surpass the results of the uniformly pruning method. In this paper, the ineffectiveness of Auto-ML pruning, which is caused by unfull and unfair training of the supernet, is shown. A deep supernet suffers from unfull training because it contains too many candidates. To overcome the unfull training, a stage-wise pruning (SWP) method is proposed, which splits a deep supernet into several stage-wise supernets to reduce the candidate number and utilize in-place distillation to supervise the stage training. Besides, a wide supernet is hit by unfair training since the sampling probability of each channel is unequal. Therefore, the full-net and the tinynet are sampled in each training iteration to ensure that each channel can be overtrained. Remarkably, the proxy performance of the subnets trained with SWP is closer to the actual performance than that of most of the previous AutoML pruning work. Furthermore, experiments show that SWP achieves the state-of-the-art in both CIFAR-10 and ImageNet under the mobile setting.

Keywords: Automated machine learning (AutoML), channel pruning, model compression, distillation, convolutional neural networks (CNN).

Citation: M. Y. Zhang, X. Y. Yu, L. L. Ou. Effective model compression via stage-wise pruning. Machine Intelligence Research, vol.20, no.6, pp.937–951, 2023. http://doi.org/10.1007/s11633-022-1357-9

1 Introduction

Deep convolutional neural networks (deep CNNs)\textsuperscript{[1–4]} achieved outstanding results in many computer vision tasks. However, deep CNNs come with a huge computational cost, which limits application on embedded devices (i.e., mobile phone).

To expand the application scope of deep CNNs, channel pruning methods were proposed. Traditional channel pruning methods always rely on human-design rules\textsuperscript{[5, 6]}. Recently, inspired by the neural architecture search (NAS), some automated machine learning (AutoML)-based pruning work\textsuperscript{[7–9]} has been proposed to automatically prune channels without a human-design mode. Considering a network with ten layers (each layer contains 32 channels), the candidates for each layer and the entire network could be 32 and 32\textsuperscript{10}, respectively. Thus, AutoML-based pruning methods can be seen as fine-grained NAS because there are more candidates than normal NAS\textsuperscript{[10–12]} in each layer.

For the AutoML-based methods mentioned above, some based on reinforcement learning or evolutionary algorithms\textsuperscript{[8, 7, 13]} are quite time-consuming due to iterative retrain for each pruned network. To reduce the computation in pruning, some AutoML-based work\textsuperscript{[9, 10, 11, 15]} share the weights through training a supernet for all candidate pruned networks which are called subnets. A typical weight-sharing pruning approach contains three steps: training a supernet by iteratively sampling and updating different candidates, searching the best subnet based on the evolutionary or greedy algorithm and training the best subnet from the scratch. However, Chu et al.\textsuperscript{[16]} considered that the weight-sharing method causes unfull training in the first step since each candidate (subnet) has only a small sampling probability in training. Moreover, unfull training leads to an inaccurate evaluation in the second step, which means some candidates perform well in weight-sharing while bad in training from scratch. The problem of inaccurate evaluation is particularly obvious in AutoML-based pruning because of more candidates contained in the supernet. Moreover, the width of a supernet also has an impact on the effectiveness of evaluation.

To solve the problem mentioned above, a stage-wise training and searching approach is proposed in this paper. Inspired by \textsuperscript{[17]}, we consider to divide a deep supernet into several stage-wise supernets (i.e., ResNet50\textsuperscript{[18]} consists of 4 stages) to reduce the depth of the supernet. Since each stage of the supernet can be trained and searched independently, the candidate number in one stage is an exponential reduction compared with the whole supernet. With a small search space at each stage, the sampling probability of the candidates is increased, which means that each supernet can be fully trained. To alleviate the unfair training result caused by the width, of the supernet, both the fullnet and the tinynet, which
have the largest and smallest width respectively, are trained in each iteration. Besides, we present a distributed evolutionary algorithm where each stage can be searched independently in terms of an evolutionary algorithm (EA). The constraints (i.e., floating point operations (FLOPs), latency) for each EA are provided by another EA, called the EA manager, where the EA manager searches for the best combination of FLOPs in each stage. Due to the small and independent stage-wise search space, each EA can be sped up in a parallel way.

However, the stage-wise supernet in each stage cannot be trained without internal ground truth. To solve such a problem, an existing pre-trained neural network was used to generate stage-wise feature maps that are viewed as ground truth in each stage. Nevertheless, it is also time-consuming to obtain a pre-trained neural network as the teacher network. Besides, the structural difference between the teacher network and the student network has a strong impact on distillation results. Hence, a stage-wise inplace distillation method is put forward for the fullnet (the largest width subnet) to supervise the learning of subnets. It is worth noting that the fullnet is jointly trained with other subnets. Thus, there does not exist an extra cost for obtaining the fullnet. Furthermore, our method can easily search both single-path and multi-path networks.

Our contribution lies in four folds:

1. We propose a stage-wise training and searching pipeline for both channel pruning and NAS. By splitting a CNN into several stages, the number of stage-wise candidates is exponential reduced in contrast to the net-wise candidates. Hence, each candidate obtains full training, which is the essence of accurate evaluation for searching.

2. To conveniently provide stage-wise ground truth for each stage, a stage-wise inplace distillation method is presented through the joint training of the fullnet and subnets. Thus, the fullnet can easily supervise the learning of the subnets by offering stage-wise feature maps. Compared to DNA, our method avoids the impact of teacher model architecture on distillation results during training.

3. To accelerate the search process, a distributed evolutionary algorithm is suggested. Each stage can be searched by the EA with constraints given by an EA manager in a parallel way. Compared to the previous EA, the distributed EA can speed up approximately $2x$.

4. Experiments show that the proposed method can enhance the ranking effectiveness of searching and achieve the state-of-the-art results in several datasets.

2 Related works

Neural architecture search. The purpose of neural network structure search is to automatically find the optimal network structure with reinforcement learning (RL), evolutionary algorithm, gradient, and parameter sharing methods. RL-based and EA-based methods need to evaluate each sampled network by retraining them on the dataset, which is time-consuming. The gradient-based method can simultaneously train and search the optimal subnet by assigning a learning weight to each candidate operation. However, the gradient-based approach causes unfair training results because some candidates obtain more learning resources than others. Moreover, since gradient-based approaches need more memory for training, they can not be applied to the large-scale dataset. Parameter sharing methods can search on the large-scale dataset by activating only one candidate in each training iteration. Nevertheless, parameter sharing methods cause unfull training results. Unfull or unfair training results will cause an inaccurate search evaluation, which means that the best-searched architecture is not the optimal one after retraining. To solve such a problem, Li et al. proposed a blockwisely search method, which can more fully train each sampled subnets.

Pruning for CNNs. Pruning some redundant weights is a prevalent method to accelerate the inference of CNNs. According to the different granularity of the pruning, it is divided into weight pruning and channel pruning. For weight pruning, individual weights in the channel are removed based on some rules, which causes unstructured sparse filters and can not be accelerated directly on most hardware. Therefore, much recent work has focused on channel pruning. Channel pruning methods can accelerate the inference of CNNs on general-purpose hardware by reducing the number of filters since the remaining filters are structural. Though these methods achieve a remarkable improvement in the practicality of pruning, it still needs human-designed heuristics to guide pruning.

AutoML pruning. Recently, inspired by NAS work, AutoML pruning methods have attracted growing interest in automatically pruning for deep CNNs. Different from NAS, the candidate choices are consecutive in the channel pruning task. Compared to pruning methods based on the human-craft rule, AutoML pruning methods aim to find the best configuration without manual tuning. AMC adopted a deep deterministic policy gradient (DDPG) agent to sample a pruned network. And the performance of the pruned networks is evaluated by training from the scratch, which is time-consuming and cannot be applied to a large-scale dataset. MetaPruning trained a PruningNet that can predict weights for any pruned network, while the amount of the PruningNet is several times of the target network, which leads to unfull train. AutoSlim first trained a slimmable network in which the weights between different widths are shared through the supernet, and then searched for the best subnet in terms of greedy algorithm. However, the width of the convolutional layer in each
subnet must be the same in training. This leads to the problem that the best subnet achieves the highest accuracy with weight-sharing, but at the same time, the best subnet gets poor performance when trained from scratch. To keep the consistency of searching and retraining results, the proposed stage-wise pruning method splits a CNN into several stages and trains them separately under the supervision of the fullnet, which will be explained in Section 3.

Knowledge distillation. Knowledge distillation is used to train the small student model on a transfer feature set with soft labels or intermediate representations provided by the large teacher model. Soft targets lead to the superior performance of knowledge distillation[41]. However, as the network is designed to become deeper and deeper, it is not enough to just transfer knowledge to a student network from a teacher network by soft targets. To solve such a problem, some previous work[42–46] transferred the knowledge for the student network from the internal representation of the teacher network. All existing work assumed that the teacher network has been pre-trained. Nevertheless, it is always time-consuming to train from scratch to obtain a pretrained teacher network that is essential for various knowledge distillation methods. For example, it may cost more than 10 GPU days to train a ResNet on ImageNet. Moreover, Liu et al.[48] found that the architecture of the teacher and the student networks had a huge impact on transferring results. Hence, we proposed a stage-wise inplace distillation method to overcome the gap and reduce the time consumption.

3 Stage-wise pruning

The problem of inaccurate evaluation caused by weight-sharing is introduced in Section 3.1. We find that the depth and width of the supernet have an impact on training adequacy. Thus, the stage-wise inplace-distillation is proposed in Section 3.2 to alleviate the aforementioned drawbacks. To efficiently search the optimal subnet from supernet trained by the stage-wise inplace-distillation, we present a distributed evolutionary algorithm in Section 3.3.

3.1 Challenge of weight-sharing

AutoML pruning methods always need to train a supernet that shares weights for all subnets firstly and almost immediately evaluates the accuracy for each subnet. For many AutoML pruning approaches[7–9], pruning candidates (subnets) directly compared with each other according to the accuracy of the evaluation. The subnets with higher evaluation accuracy are selected and expected to deliver high accuracy after training from the scratch. However, such an intention cannot be necessarily achieved since some subnets which have outstanding performance on shared parameters perform poorly after training from scratch.

To visualize the performance drop of weight-sharing, we train a supernet with different candidate numbers. For a trained supernet, we randomly sample a batch of subnets from the supernet and evaluate them on the validation dataset. Statistical accuracy expectation \( E(a_{\text{super}}) \) is utilized to evaluate whether the supernet is adequately trained. \( E(a_{\text{super}}) \) is written as

\[
E(a_{\text{super}}) = \sum_{i=1}^{k} a_{\text{sub}_i}
\]

where \( h \) denotes the number of the randomly sampled subnet, and \( a_{\text{sub}_i} \) represents the accuracy of the \( i \)-th subnet. As shown in Fig.1(a), with the increase of the candidate number, the top-1 accuracy expectation of supernet dramatically degrades under 100 epochs while it falls slightly under 500 epochs. Moreover, we train three different depth supernets and then calculate their \( E(a_{\text{super}}) \). It is found that the expectation of top-1 accuracy is related to the depth of the CNN, which is shown in Fig.1(b). Subnets are sampled from the super network. Thus, the deeper and wider supernet can represent more subnets. In weight-sharing training method, only one subnet will be sampled and updated in each iteration. Therefore, too many subnets will seriously affect the update efficiency of the supernet. Through experiments on ResNet (shown in Fig.1), we found that deepening or widening the supernet will significantly reduce its performance. FairNAS[46] and DNA[47] found that the supernet with poor performance do not provide accurate performance rankings for subnets. Therefore, how to efficiently train deep and wide supernets is still a challenge in the weight-sharing training method.

Depth of the supernet. CNNs (i.e., ResNet152[8]) are designed deep to enhance the representative ability, which exponentially increases the subnet number in pruning. The subnet number \( N \) that inherits weights from the supernet can be formulated as

\[
\|N\| = g^L
\]

where \( g \) denotes the candidate number for each convolutional layer and \( L \) represents the depth of the CNN. For the channel pruning of a deep CNN, the search space \( N \) is always large (e.g., \( > 30^{56} \)). Hence, many subnets get unfull training results due to weight-sharing, which leads to the ineffectiveness of evaluation.

Width of the layers. Some AutoML pruning works[8, 9, 14, 15, 47] train a single neural network executable at different widths as the supernet. There is not only cross-layer weight-sharing but also within the layer. In one layer, the parameters of different widths (candidates) are shared. For instance, all parameters of 0.25× (width
scaled by 0.25 of the original width) are shared with the half parameters of 0.5×. Each sampling in any training step is independent of the each other. Thus, the sampled probability of the \( i \)-th channel in the \( l \)-th layer can be formulated as

\[
P(c_i^l) = 1 - \frac{i - 1}{m}
\]

where \( m \) denotes the candidate number in the layer. Therefore, the channels with small indexes can be trained more times, which causes unfair training results. Formally, we consider a common supernet that contains \( L \) layers, each with \( m \) channels. In training, a group of ratio sequence \( R \) can be obtained under certain constraints (i.e., FLOPs), where each ratio sequence \( r = [c_1^l, \ldots, c_L^l] \in \mathbb{R} \), \( c_L^l \) denotes the max sampled channel index in the \( L \)-th layer. Because of independent sampling in each layer, the sampled probability of \( r \) can be described as

\[
P(r) = P(c_1^l)P(c_2^l)\cdots P(c_L^l).
\]

According to the inequality of arithmetic and geometric means, we have

\[
P(r) \leq \left( \frac{\sum_{l=1}^L P(c_l^i)}{L} \right)^L.
\]

Equality holds if and only if \( P(c_1^l) = P(c_2^l) = \cdots = P(c_L^l) \). According to (3), \( P(c_1^l) = P(c_2^l) = \cdots = P(c_L^l) \) means that the maximum sampled channel index \( i \) in all layers should be the same. Hence,

\[
P(r) \leq \left( 1 - \frac{cm - 1}{m} \right)^L.
\]

That is to say, the subnet with uniform sampling obtains the most training resource under certain constraints, which means that the pruning strategy of previous work is always prone to getting a uniform sampling model.

### 3.2 Stage-wise inplace distillation for training

As mentioned above, too many candidates in training can lead to ineffective evaluation on searching because of unfull training results. To adequately train the supernet, we divide the supernet \( S \) into \( N \) stages according to the depth. Hence, the search space of supernet \( S \) can be represented as

\[
S = [S_1, \ldots, S_i, S_{i+1}, \ldots, S_N]
\]

where \( S_i \) denotes the stage-wise supernet of the \( i \)-th stage. Then, we can train the supernet by training the stages separately. The learning of the stage \( i \) can be formulated as

\[
W_i^* = \min_{W_i} L_{train}(W_i; S_i; X, Y)
\]

where \( X \) and \( Y \) denote the input data and the ground truth labels, respectively. Subsequently, the candidates’ number for the \( i \)-th stage can be written as

\[
\|S_i\| = g^{L_i}
\]

where \( L_i \) denotes the depth of the \( i \)-th stage and is smaller than \( L \). The search space can be extremely reduced when we train a stage-wise supernet in each stage independently.

However, the internal ground truth of (8) can not be obtained directly from the dataset. One solution is using block-wise feature maps generated by a pre-trained network to supervise the training of subnets. However, it is time-consuming to obtain a pre-trained network through
training from scratch in practice (e.g., ResNet50 > 10 GPU days). Besides, the architecture of teacher and student networks has a huge impact on transferring results\(^\text{[7]}\).

To tackle the above problem, the stage-wise inplace distillation is proposed here. The essential idea behind the inplace distillation\(^\text{[14]}\) is to transfer knowledge inside a single supernet from the fullnet to a subnet in each training iteration. For an individual convolutional layer, the performance of the wider candidate can not be worse than the slim one since the wider one can achieve the performance of the slim one by learning weights from some useless channels to zeros. In stage-wise inplace distillation, we use the stage-wise representation of the fullnet to supervise the training of subnets. The pipeline of the supervision with stage-wise inplace distillation is shown in Fig. 2. The output \(\hat{Y}_{i-1}\) of the \((i-1)\)-th stage from the fullnet is adopted by the input of the \(i\)-th stage of subnets. To supervise the subnets learning from the fullnet, the following MSE loss is considered as the distillation loss in Fig. 2.

\[
L_{\text{train}}(\hat{Y}_{i-1}, Y_i) = \frac{1}{F}(Y_i - \hat{Y}_i)^2 \quad (10)
\]

where \(Y_i\) and \(\hat{Y}_i\) denote the output of the subnets and fullnet in the \(i\)-th stage, respectively. \(F\) is the number of channels in \(Y\).

As mentioned in Secton 3.1, the channels with larger indexes suffer from unfull training. To ensure sufficient training of each channel, an intuitive approach is overtraining. In each iteration, we will sample a fullnet, a subnet, and a tinynet. The fullnet is an original supernet that is no channel pruning. The subnet can be obtained by randomly setting the pruning ratio for each layer. The tinynet sets the smallest pruning ratio for each layer. Given a batch of input images and ground truth labels, we first calculate the task loss (e.g., cross-entropy) and the gradients of fullnet through forward and backward propagation simultaneously. The stage-wise feature maps of fullnet \(\hat{Y} = [\hat{Y}_1, \cdots, \hat{Y}_N]\) are saved. Subsequently, under the supervision of \(\hat{Y}\), the distillation loss in (10) and the gradients of the stage-wise subnet are calculated. Furthermore, in a subnet training process, we train the smallest width (tinynet) to improve the performance of the supernet. Based on this, each channel can be trained at least once in one iteration. The ground truth label has been generated in the fullnet training process. Thus, the training of each stage-wise subnet can be accelerated in a parallel way. The detailed algorithm is described in Algorithm 1. We use the multiprocessing module\(^\text{1}\) to parallelize our algorithm.

**Algorithm 1.** Framework of supervision with stage-wise inplace distillation

**Require:** The fullnet \(S\), the stage-wise supernets \([S_1, \cdots, S_N]\), and the dataset \((X, Y)\);

**Ensure:** The well-trained stage-wise supernets \([S_1, \cdots, S_N]\);

1) for \(t = 1, \cdots, T\) do
  2) Get next mini-batch of data \(x\) and label \(y\) from \((X, Y)\)
  3) Execute fullnet \(y' = S(x)\), and save stage-wise feature maps \(\hat{X} = [x, \cdots, \hat{X}_{N-1}], \hat{Y} = [\hat{Y}_1, \cdots, \hat{Y}_N]\)
  4) Calculate cross entropy loss, \(\text{loss} = CE(y', y)\)
  5) Clear gradients, \(\text{optimizer.zero_grad()}\)
  6) Accumulate gradients, \(\text{loss.backward()}\)
  7) Randomly sample the width for convolutional layers and obtain stage-wise subnets, \(S_t = [S_{t,1}, \cdots, S_{t,N}]\)
  8) Uniformly sample the smallest width for convolutional layers and obtain stage-wise tinynets, \(S_t = [S_{t,1}, \cdots, S_{t,N}]\)
  9) multiprocessing \(s = [S_t, S_{t,1}], x_s = [\hat{X}, \hat{Y}], y_s = [\hat{Y}, \hat{Y}]\) do
   10) Execute subnet, \(y' = s(x_s)\)
   11) Calculate distillation loss, \(\text{loss} = L(y', y_s)\)
   12) Accumulate gradients, \(\text{loss.backward()}\)
  13) end multiprocessing
  14) Update weights, \(\text{optimizer.step()}\)
  15) end for
  16) return trained stage-wise supernets \([S_1, \cdots, S_N]\)

### 3.3 Distributed evolutionary for searching

After the stage-wise supernet is trained, the learning ability of a subnet can be evaluated by its loss at each stage. However, each stage-wise supernet still contains about \(30^{10}\) stage-wise subnets. It is infeasible to evaluate all of them. For previous one-shot pruning methods, random sampling, EA-based, and RL-based methods have been used to sample sub-models from the trained supernet for further evaluation. The most recent work found that EA can search models better than RL but needs to spend more time searching. For the proposed stage-wise inplace distillation, a novel method is suggested to search for the best subnet according to the stage-wise performance under certain constraints.

The EA in Fig. 3 is applied to search for the best stage-wise subnet that has the smallest distillation loss under a given FLOPs constraint. In \([8, 22]\), the genes of each stage-wise subnet were encoded with a vector of channel numbers in each layer. Different from the above work, we aim to imitate the behavior of the teacher in each state. Thus, (10) is used to evaluate the learning ability of each gene. Because the supernet is split into several stages, the search space of an individual EA is shrunk about \(10^{90}\) times. Then the top \(k\) genes with the
lowest loss are selected for generating the new genes with mutation and crossover. The mutation is carried out by changing the proportion of elements in the gene randomly. The crossover means that we randomly recombine the genes into two-parent genes to generate offspring. We can easily enforce the constraint by eliminating the unqualified genes. Through further repeating the top k selection process and the new gene generation process for several iterations, the gene that meets constraints while achieving the lowest loss can be obtained.

The further problem is how to assign the optimal stage-wise constraints for each stage. To automatically find the best assignment plan for stage-wise constraints, a distributed evolutionary algorithm (DEA) is proposed in this section. The workflow of DEA is revealed in Fig. 3. The EA manager is also a kind of evolutionary algorithm that provides the strategy of FLOPs constraint for other EAs. Different from the EA above, the genes in EA manager are encoded with a vector of FLOPs constraint in each stage. The evaluation of each gene is the sum of the distillation losses given by all stage-wise EAs. Subsequently, the top k genes that are kept generate offspring through mutation and crossover. After several repetitions, the optimal stage-wise constraints can be ob-

Fig. 2 Illustration of the stage-wise training. There are three forms of the networks, including the fullnet, the subnet and the tinynet. The fullnet generates and transfers its knowledge to the subnet and the tinynet by minimizing the L2-distance between their stage-wise output feature maps. It is worth noting that these three networks are weight-sharing.

Fig. 3 Illustration of the distributed evolutionary algorithm. Both EA manager and EA are modified from an evolutionary algorithm. Given the FLOPs constraint for the whole network, EA manager is responsible for searching for the best combination of stage-wise FLOPs. The feedback of each FLOPs gene in EA manager is provided by each EA with a search for the smallest distillation loss under the stage-wise FLOPs constraint.
tained from the top 1 genes. The detail is shown in Algorithm 2. Moreover, each stage-wise EA is parallelized to accelerate the searching process. Specifically, we use the teacher network to generate a batch of representation features for each stage. Therefore, each EA can search for the best stage-wise subnet independently. After searching all stages, we can assemble the best model by selecting the best stage-wise subnet from each stage.

**Algorithm 2.** Framework of the distributed evolutionary algorithm

**Require:** The constraint $C$, the fullnet $S$, the stage-wise supernets $[S_1, \ldots, S_N]$ and the dataset $(X, Y)$

**Ensure:** The best subnet: $S_{stop}$
1) Execute the fullnet and save the stage-wise feature maps $Y, y' = S(X), Y = [Y_1, \ldots, Y_N]$
2) Randomly generate a batch of genes $G$ under constraint $C_G, G = [G_1, \ldots, G_N], s.t. ||G_i|| = ||C_i|| = C$
3) for $t = 1, \ldots, T$
4) for $g = G_1, \ldots, G_N$
5) Obtain the stage-wise constraint from $g, g = [C_1, \ldots, C_N]$
6) **multiprocessing** $c = C_1, \ldots, C_N, x_* = X, \ldots, Y_{N-1}, y_* = Y_1, \ldots, Y_N, s = S_1, \ldots, S_N$
7) Search the best stage-wise subnet $s'$ and calculate distillation loss by $EA$ in Algorithm 3, $(s', L_{p_i}) = EA(s, x_*, y_*, c)$
8) **end multiprocessing**
9) Calculate the total loss $L$ for $g, L = L_{s_1} + \cdots + L_{s_N}$
10) **end for**
11) Keep top $k$ genes $G_{topk}$ according to $L$
12) Generate $M$ mutation genes under constraint $C$, $G_{mutation} = [G_{m1}, \ldots, G_{mM}]$
13) Generate $H$ crossover genes under constraint $C$, $G_{crossover} = [G_{c1}, \ldots, G_{cH}]$
14) Generate new population $G, G = G_{mutation} + G_{crossover}$
15) **end for**
16) Select $S_{stop} = [s'_1, \ldots, s'_k]$ with smallest $L$
17) return $S_{stop}$

**Algorithm 3.** Framework of evolutionary algorithm

**Require:** The constraint, $C$, the stage-wise supernet $S$, and the stage-wise feature maps $(X, Y)$

**Ensure:** The best stage-wise subnet: $S_{stop}$ and the stage-wise distillation Loss: $L$
1) Randomly generate a batch of genes $G$ under constraint $C, G = [G_1, \ldots, G_d]$
2) for $t = 1, \ldots, T$
3) for $g = G_1, \ldots, G_d$
4) Construct a stage-wise subnet according to $S$ and $g, S_g$
5) Calculate the distillation loss of $S_g, \ L_g = L(S_g(X), Y), L$ from (10)
6) **end for**
7) Keep top $k$ genes $G_{topk}$ according to $L$
8) Generate $M$ mutation genes under constraint $C$, $G_{mutation} = [G_{m1}, \ldots, G_{mM}]$
9) Generate $H$ crossover genes under constraint $C$, $G_{crossover} = [G_{c1}, \ldots, G_{cH}]$
10) Generate new population $G, G = G_{mutation} + G_{crossover}$
11) **end for**
12) Select $G_{top}$ with smallest $L$
13) Construct the best stage-wise subnet $S_{stop}$ according to $S$ and $G_{top}$
14) return $S_{stop}, L_{G_{top}}$

4 Experiments

In this section, the effectiveness of our proposed stage-wise pruning method is demonstrated. First, we explain the experiment settings on the CIFAR-10 and ImageNet 2012 datasets. Then, we prune ResNet in CIFAR-10 and visualize the consistency of performance between searching and retraining. Moreover, we apply the stage-wise pruning method to ImageNet 2012 and compare the results with those of other state-of-the-art works. Finally, ablation studies are carried out to determine the influence of using inplace distillation.

4.1 Setups

The stage-wise pruning method consists of three steps:

**Stage-wise training.** According to the resolution size of the feature maps, we split ResNet and MobileNet into 4 and 5 stages, respectively. The distillation loss of each stage can be calculated by (10). To match the channel number of the fullnet, the output of each stage is connected with a $1 \times 1$ convolutional layer without BatchNorm and non-linear activation. As MetaPruning, the width of each convolutional layer is subdivided into 31 ratios from 0.1 to 1.0.

On the CIFAR-10 dataset, we randomly sample 200 images for each class from training images as a validation dataset. The remaining images are used to train the supernet. We use momentum stochastic gradient descent (SGD) to optimize the weights, with initial learning rate $\eta = 0.025$, momentum 0.9, and weight decay $3 \times 10^{-4}$. The supernet is trained for 50 epochs with batch size 512, and the learning rate decays $0.1 \times$ per 10 epochs.

On the ImageNet 2012 dataset, we randomly sample 50 images for each class from training images as a validation dataset. The remaining images are used to train the supernet. We use momentum SGD to optimize the weights, with initial learning rate $\eta = 0.1$, momentum 0.9, and weight decay $3 \times 10^{-4}$. The supernet is trained for 100 epochs with batch size 512, and the learning rate decays $0.1 \times$ when the epoch is 30, 60 and 90.

**Stage-wise searching.** After training the stage-wise
supernet as above, the best subnet is searched in each stage. Firstly, we use the fullnet to generate and save the stage-wise feature maps with 2 048 batch size. Subsequently, the hyperparameter of each EA and EA manager is set to 128 population number, 0.1 mutation probability, 10 iterations. We use 4 and 5 multiprocess to speed up the search for ResNet and MobileNet series, respectively. Each process can use 2 GPUs to infer with 2 048 batch size.

Retraining. After searching for the best subnet, we adopt the same training scheme as [8] on ImageNet 2012 for both ResNet and MobileNet series. The same lines as in [12] are followed for the training scheme of ResNet on the CIFAR-10. It is noted that all baseline models are trained under the same scheme mentioned above.

4.2 Pruning ResNet on CIFAR-10 and analysis

To demonstrate the effectiveness of stage-wise pruning, we prune ResNet-56[9] under 50% FLOPs constraint on the small dataset of CIFAR-10. As shown in Table 1, our stage-wise pruning (SWP) method surpasses the baseline model by about 1.4%. Moreover, our method outperforms all other pruning methods in terms of top-1 accuracy.

Table 1 Pruning results of ResNet-56

| Method   | FLOPs (M) | Top-1 accuracy (%) |
|----------|-----------|---------------------|
| ResNet-56| 125.49    | 93.27               |
| FP[48]   | 90.90     | 93.06               |
| RFP[50]  | 90.70     | 93.12               |
| HRank[34] | 88.72     | 93.52               |
| EagleEye[34] | 62.23     | 94.66               |
| **SWP (ours)** | **61.36** | **95.03**           |

To evaluate the consistency of model ranking abilities for our method and other AutoML methods, we visualize the relationship between the proxy performance and actual performance. A PruningNet[8] and a universally slimming network (USNet)[15] are trained as supernets under the same training scheme due to fairly compared with MetaPruning[8] and AutoSlim[9]. The total distillation loss is visualized as the proxy performance of our method. The other two methods take the top-one accuracy of each subnet that inherits weights from the supernet as proxy performance. Each subnet will be trained from scratch in order to obtain its actual performance. As shown in Fig. 4, the proposed method has a strong correlation between the proxy performance and the actual performance, while others barely rank the subnets.

4.3 Pruning MobileNet and ResNet on ImageNet 2012

In addition, our method is extended to lighting models on a large-scale dataset, ImageNet 2012. Table 2 summarizes our results on MobileNet V1[8], MobileNet V2[2] and ResNet-50[3]. It is noted that we experiment with both residual and non-residual networks. We compare our results with uniformly pruned baselines and other recent channel pruning methods. It is shown that our method achieves the best results across different computational budgets. In the case of extreme pruning (i.e., 40M FLOPs), MobileNetV1/MobileNetV2 pruned by SWP outperforms baseline model to a considerable degree (9.7% and 6.4%). Fig.5 compares the curve of top-1 accuracy and FLOPs for the most recent AutoML pruning methods and uniformly pruning methods. Our SWP models can achieve better accuracy with a lower computational complexity than other methods.

4.4 Pruning under latency

More and more attention is paid to directly optimizing the inference time on the target device. Without knowing the implementation details inside the device, SWP learns to prune channels according to the latency estimated from the device. To evaluate the realistic acceleration, we measure the forward time of the search for the best subnet on a 2080Ti GPU under the latency constraint. The results of MobileNet V1 and V2 are shown in Tables 3 and 4. Under the same compression ratio and similar inference time, our method can obtain a better top-one accuracy.

4.5 Visualization of searched models

Channel pruning models are visualized and some insights from the results are discussed. We compare our results with default channels and MetaPruning[8] on Res-
Table 2 Results of ImageNet classification. We show the top-1 accuracy of each method under the same or closed FLOPs. The sign “*” denotes the results implemented by us.

| Network | Method | Acc@1 FLOPs | Parameters | GPU days |
|---------|--------|-------------|------------|----------|
| ResNet-50 | Baseline 0.75x* | 68.4% | 325M | 2.6M – |
| | AMC[7] | 70.5% | 285M | 1.8M – |
| | SN[14] | 69.5% | 325M | 4.3M 10 |
| | MP[9] | 70.4% | 281M | 1.7M 14 |
| | AutoSlim[6] | 69.1% | 325M | 1.8M 14 |
| MobileNet V1 | SWP (ours) | 70.9% | 285M | 1.7M 14 |
| | Baseline 0.25x* | 50.6% | 41M | 0.5M – |
| | MP[90] | 57.2% | 41M | 0.5M 10 |
| | SN[14] | 53.1% | 41M | 3.4M 10 |
| | SWP (ours) | 60.3% | 41M | 0.5M 10 |
| MobileNet V2 | SWP (ours) | 73.4% | 220M | 2.6M 14 |
| | Baseline 0.35x* | 54.3% | 43M | 1.7M – |
| | MP[90] | 58.3% | 43M | 1.9M 10 |
| | SWP (ours) | 60.7% | 43M | 1.9M 10 |
| ResNet-50 | Baseline 1.0x* | 76.6% | 4.1G | 25.5M – |
| | Baseline 0.75x* | 74.8% | 2.3G | 14.7M – |
| | SN[14] | 74.9% | 2.3G | 25.5M 12 |
| | MP[90] | 75.4% | 2.3G | 16.8M 16 |
| | AutoSlim[6] | 75.6 | 2.3G | 15.4M 16 |
| | AOFP-C1[53] | 75.6 | 2.6G | – – |
| | C-SGD-50[50] | 74.5 | 1.7G | – – |
| | SNIP[47] | 74.9 | 2.0G | 13.7M – |
| | DSD[50] | 74.7 | 2.0G | 14.8M – |
| | DCP[9] | 75.2 | 2.0G | 15.3M – |
| ResNet-50 | ThiNet-50[57] | 74.7 | 3.4G | 12.4M – |
| | SWP (ours) | 76.1 | 2.0G | 14.9M 16 |
| ResNet-50 | Baseline 0.5x* | 72.0% | 1.0G | 6.8M – |
| | SN[14] | 72.5% | 1.0G | 25.5M 12 |
| | ThiNet-30[70] | 72.1% | 2.2G | 8.7M – |
| | MP[9] | 73.4% | 1.0G | 6.5M 14 |
| | AutoSlim[6] | 74.0% | 1.0G | 6.4M 14 |
| | SWP (ours) | 75.6% | 1.0G | 6.4M 14 |

Net-50. In Figs. 6(a)–6(c), we show the channel number in the top, middle, and bottom layers of bottleneck blocks on ResNet-50, respectively. Firstly, it is found that our method is prone to prune more channels from top layers compared to MetaPruning. It is noted that although the top layers have a small number of channels, the output feature maps of the top layer can be extracted by the next middle layer where kernel size = 3. Hence, pruning the top layers can reduce computational complexity. Secondly, our method and MetaPruning keep more channels for downsampling layers because the feature map size is shrunk. Moreover, our method prunes fewer channels for bottom layers, since the feature maps between the subnet and the fullnet should be as close as possible.

4.6 Ablation study

The choice of the teacher network. The influence of the distillation strategy in ResNet-50 is analyzed. In our method, the teacher and student networks are jointly trained by stage-wise inplace distillation. In Strategy 1, we use a pre-trained network with the same architecture as fullnet to supervise the training of subnets. In Strategy 2, EfficientNet-B0[58], of which performance surpasses ResNet-50 with lower parameters, is employed as the teacher network. The results are shown in Table 5. It is found that the performance of the model searched with the inplace distillation method is almost the same as the one searched with a pre-trained method. Hence, the fullnet can supervise the training of the subnets while training itself. Therefore, it is unnecessary to spend much of time to obtaining a pre-trained teacher model. Moreover, despite the fact that EfficientNet-B0 has outstanding performance compared to ResNet-50, the models searched with EfficientNet-B0 have worse performance. It may be caused by the large gap between the architecture of the teacher and the student networks.

To further figure out the reason why using ResNet-50 to search can achieve better performance of models than EfficientNet-B0, we visualize the channel number in the bottom layers of each block. As shown in Fig. 7, the model searched with EfficientNet-B0 keeps fewer channels. Moreover, EfficientNet-B0 has much fewer channels than ResNet-50. For example, EfficientNet-B0 has only 40 channels, while ResNet-50 has 64 channels in the bottom layer of the first stage. Thus, the student does not need many channels to imitate the stage-wise feature maps generated by EfficientNet-B0. However, over-pruning the channels from the bottom layers will result in insufficient information transmission between the stages.

The stage number. In previous experiments, the supernet was simply split into 4 (ResNet) or 5 (MobileNet series) stages according to downsampling. The optimal stage number is still unknown. We set the stage numbers as 8, 4, 2, and 1 for ResNet-50. It is seen from Table 6...
that the performance of the model searched with 8 stages is almost the same as the one searched with 4 stages and surpassed the one searched with fewer than 4 stages. Besides, since we need to save the intermediate feature maps of the teacher network to supervise the training of the student network, more memory is required if a larger stage number is set.

**Acceleration of the EA manager.** EA manager is an essential component of the distributed evolutionary algorithm, which decouples computation constraints and subnet performance in the search process. To demonstrate the effectiveness of the EA manager, we conduct two searches on the stage-wise ResNet-50 supernet. Shown in Fig. 8, EA with EA manager converges in only 10 iterations, while EA without EA manager needs nearly 20 iterations.

### 4.7 Conclusions

In this work, we have presented stage-wise channel pruning. A stage-wise training process based on inplace distillation and a distributed evolutionary algorithm is proposed in this paper. It is found that the large search space causes low accuracy of the supernet. Hence, we split a supernet into several stage-wise supernets to de-
grade the complexity of the search space in both training and searching. The experiments show the effectiveness of our proposed method by delivering a higher accuracy than in previous work on both the CIFAR-10 and ImageNet datasets. In addition, the consistency of the proxy and actual subnet performance is greatly improved. Experiments with various distillation strategies prove that inplace distillation can replace pre-trained distillation, thereby reducing the time to train a teacher network from scratch. We further discuss the impact of stage numbers on search results and found that splitting the supernet according to downsampling is the best tradeoff between memory and accuracy. In the feature work, we

---

Table 5 Comparison of stage-wise pruning with different distillation strategies

| Teacher  | Student FLOPs | Acc@1  |
|----------|---------------|--------|
| Ours     | 2.0G          | 76.1%  |
|          | 1.0G          | 75.6%  |
| Strategy 1 | 2.0G          | 76.1%  |
|          | 1.0G          | 75.6%  |
| Strategy 2 | 2.0G          | 75.6%  |
|          | 1.0G          | 73.1%  |
will extend our work to other computer vision tasks, such as object detection, super-resolution, etc.

Acknowledgements

This work was supported by Natural Science Foundation of Zhejiang Province, China (No. LY21F030018) and National Key R&D Program of China (No. 2018YFB1308400).

Declarations of conflict of interest

The authors declared that they have no conflicts of interest to this work.

References

[1] A. G. Howard, M. L. Zhu, B. Chen, D. Kalenichenko, W. J. Wang, T. Weyand, M. Andreetto, H. Adam. MobileNets: Efficient convolutional neural networks for mobile vision applications, [Online], Available: http://arxiv.org/abs/1704.04861, 2017.

[2] M. Sandler, A. Howard, M. L. Zhu, A. Zhmoginov, L. C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Salt Lake City, USA, pp. 4510–4520, 2018. DOI: 10.1109/CVPR.2018.00474.

[3] K. M. He, X. Y. Zhang, S. Q. Ren, J. Sun. Deep residual learning for image recognition. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Las Vegas, USA, pp. 770–778, 2016. DOI: 10.1109/CVPR.2016.90.

[4] G. Huang, Z. Liu, L. Van Der Maaten, K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, Honolulu, USA, pp. 2261–2269, 2017. DOI: 10.1109/CVPR.2017.243.

[5] Y. He, P. Liu, Z. W. Wang, Z. L. Hu, Y. Yang. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Long Beach, USA, pp. 4335–4344, 2019. DOI: 10.1109/CVPR.2019.00447.

[6] M. A. Carreira-Perpinan, Y. Idelbayev. “Learning-Compression” algorithms for neural net pruning. In *Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition*, IEEE, Salt Lake City, USA, pp. 8532–8541, 2018. DOI: 10.1109/CVPR.2018.00890.

[7] Y. H. He, J. Lin, Z. J. Liu, H. R. Wang, L. J. Li, S. Han. AMC: AutoML for model compression and acceleration on mobile devices. In *Proceedings of the 15th European Conference on Computer Vision*, Springer, Munich, Germany, pp. 815–832, 2018. DOI: 10.1007/978-3-030-01234-2_48.

[8] Z. C. Liu, H. Y. Mu, X. Y. Zhang, Z. C. Guo, X. Yang, K. T. Cheng, J. Sun. MetaPruning: Meta learning for automatic neural network channel pruning. In *Proceedings of IEEE/CVF International Conference on Computer Vision*, IEEE, Seoul, Republic of Korea, pp. 3295–3304, 2019. DOI: 10.1109/ICCV.2019.00339.

[9] J. H. Yu, T. Huang. AutoSlim: Towards one-shot architecture search for channel numbers, [Online], Available: http://arxiv.org/abs/1903.11728, 2019.

[10] Z. C. Guo, X. Y. Zhang, H. Y. Mu, W. Heng, Z. C. Liu, Y. C. Wei, J. Sun. Single path one-shot neural architecture search with uniform sampling. In *Proceedings of the 16th European Conference on Computer Vision*, Springer, Glasgow, UK, pp. 544–560, 2020. DOI: 10.1007/978-3-030-58517-4_32.

[11] H. Cai, C. Gan, T. Z. Wang, Z. K. Zhang, S. Han. Once-for-all: Train one network and specialize it for efficient deployment. In *Proceedings of the 8th International Conference on Learning Representations*, Addis Ababa, Ethiopia, 2020.

[12] H. X. Liu, K. Simonyan, Y. M. Yang. DARTS: Differentiable architecture search. In *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, USA, 2019.

[13] Q. G. Huang, K. Zhou, S. Y. You, U. Neumann. Learning to prune filters in convolutional neural networks. In *Proceedings of IEEE Winter Conference on Applications of Computer Vision*, Lake Tahoe, USA, pp. 709–718, 2018. DOI: 10.1109/WACV.2018.00083.

[14] J. H. Yu, L. J. Yang, N. Xu, J. C. Yang, T. S. Huang, Slim
mable neural networks. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, USA, 2019.

[15] J. H. Yu, T. Huang. Universally slimmable networks and improved training techniques. In Proceedings of IEEE/CVF International Conference on Computer Vision, IEEE, Seoul, Republic of Korea, pp. 1803–1811, 2019. DOI: 10.1109/ICCV.2019.00189.

[16] X. X. Chu, B. Zhang, R. J. Xu. FairNAS: Rethinking evaluation fairness of weight sharing neural architecture search. In Proceedings of IEEE/CVF International Conference on Computer Vision, IEEE, Montreal, Canada, pp. 12219–12228, 2021. DOI: 10.1109/ICCV48922.2021.01202.

[17] C. L. Li, J. F. Feng, L. C. Yuan, G. R. Wang, X. D. Liang, L. Lin, X. J. Chang. Block-wisely supervised neural architecture search with knowledge distillation. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, Seattle, USA, pp. 1986–1995, 2020. DOI: 10.1109/CVPR42600.2020.00206.

[18] Y. Liu, X. H. Jia, M. X. Tan, R. Vemulapalli, Y. K. Zhu, B. Green, X. G. Wang. Search to distill: Pearls are everywhere but not the eyes. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, Seattle, USA, pp. 7536–7545, 2020. DOI: 10.1109/CVPR42600.2020.00756.

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

[20] Y. Q. Liu, Y. N. Sun, B. Xue, M. J. Zhang, G. G. Yen, K. C. Tan. A survey on evolutionary neural architecture search. IEEE Transactions on Neural Networks and Learning Systems, to be published. DOI: 10.1109/TNNLS.2021.3100554.

[21] B. Zoph, Q. V. Le. Neural architecture search with reinforcement learning. In Proceedings of the 5th International Conference on Learning Representations, Toulon, France, 2017.

[22] B. Zoph, V. Vasudevan, J. Shlens, Q. V. Le. Learning transferable architectures for scalable image recognition. In Proceedings of IEEE/CVF Conference on Computer Vision and Pattern Recognition, IEEE, Salt Lake City, USA, pp. 8697–8710, 2018. DOI: 10.1109/CVPR.2018.00907.

[23] E. Real, A. Aggarwal, Y. P. Huang, Q. V. Le. Regularized evolution for image classifier architecture search. In Proceedings of the 33rd AAAI Conference on Artificial Intelligence and 31st Innovative Applications of Artificial Intelligence Conference and 9th AAAI Symposium on Educational Advances in Artificial Intelligence, Honolulu, USA, Article number 587, 2019. DOI: 10.1609/aaai.v33i01.330414780.

[24] Y. H. Xu, L. X. Xie, X. P. Zhang, X. Chen, G. J. Qi, Q. Tian, H. K. Xiong. PC-DARTS: Partial channel connections for memory-efficient architecture search. [Online]. Available: http://arxiv.org/abs/1907.05737, 2019.

[25] X. Chen, L. X. Xie, J. Wu, Q. Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In Proceedings of IEEE/CVF International Conference on Computer Vision, IEEE, Seoul, Republic of Korea, pp. 1294–1303, 2019. DOI: 10.1109/ICCV.2019.00138.

[26] C. X. Yan, X. J. Chang, Z. H. Li, W. L. Guan, Z. Y. Ge, L. Zhu, Q. H. Zheng. ZeroNAS: Differentiable generative adversarial networks search for zero-shot learning. IEEE Transactions on Pattern Analysis and Machine Intelligence, to be published. DOI: 10.1109/TPAMI.2021.3127 346.

[27] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, J. Dean. Efficient neural architecture search via parameter sharing. In Proceedings of the 35th International Conference on Machine Learning, Stockholm, Sweden, pp. 4092–4101, 2018.

[28] W. Jia, W. Xia, Y. Zhao, H. Min, Y. X. Chen. 2D and 3D palmprint and palm vein recognition based on neural architecture search. International Journal of Automation and Computing, vol. 18, no. 3, pp. 377–409, 2021. DOI: 10.1007/s11633-021-1292-1.

[29] P. Z. Ren, Y. Xiao, X. J. Chang, P. Y. Huang, Z. H. Li, X. J. Chen, X. Wang. A comprehensive survey of neural architecture search: Challenges and solutions. ACM Computing Surveys, vol. 54, no. 4, Article number 76, 2022. DOI: 10.1145/3477582.

[30] M. Zhang, H. Q. Li, S. R. Pan, X. J. Chang, C. Zhou, Z. Y. Ge, S. Su. One-shot neural architecture search: Maximising diversity to overcome catastrophic forgetting. IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 43, no. 9, pp. 2921–2935, 2021. DOI: 10.1109/ TPAMI.2020.3035351.

[31] S. Han, J. Pool, J. Tran, W. J. Dally. Learning both weights and connections for efficient neural networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems, Montreal, Canada, pp. 1135–1143, 2015.

[32] C. Gamanayake, L. Jayasinghe, B. K. K. Ng, C. Yuen. Cluster pruning: An efficient filter pruning method for edge AI vision applications. IEEE Journal of Selected Topics in Signal Processing, vol. 14, no. 4, pp. 802–816, 2020. DOI: 10.1109/JSTSP.2020.2971418.

[33] G. L. Li, X. Ma, X. Y. Wang, L. Liu, J. L. Xue, X. B. Feng. Fusion-catalyzed pruning for optimizing deep learning on intelligent edge devices. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 11, pp. 3614–3626, 2020. DOI: 10.1109/TCAD.2020.3013050.

[34] G. L. Li, X. Ma, X. Y. Wang, H. S. Yue, J. S. Li, L. Liu, X. B. Feng, J. L. Xue. Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning. Journal of Systems Architecture, vol. 124, Article number 102431, 2022. DOI: 10.1016/j.sysarc.2022.102431.

[35] Z. Liu, J. G. Li, Z. Q. Shen, G. Huang, S. M. Yan, C. S. Zhang. Learning efficient convolutional networks through network slimming. In Proceedings of IEEE International
of norm-less-informative $Y$. $2763$, European

tional $Z$. $3066410$. $H$. $7138$, 10.1109/CVPR.2017.754.

ceive optimization, $P$. $7130$ DOI: $pp$. $2017$. $7138$

ualization: $Z$. $Proceedings$ $A$. $J$. $Yim$, $81$


twise $Y$. $IEEE$. $Ding$, $J$. $Howard$, $Q$. $Le$. $MnasNet$: Platform-aware neural

architecture search for mobile. In Proceedings of $IEEE/CVF$ Conference on Computer Vision and Pattern Recognition, IEEE, Long Beach, USA, pp. 4938–4948, 2019. DOI: 10.1109/CVPR.2019.00508.

M. X. Tan, B. Chen, R. M. Pang, V. Vasudevan, M. Sandler, A. Howard, Q. V. Le. MnasNet: Platform-aware neural architecture search for mobile. In Proceedings of $IEEE/CVF$ Conference on Computer Vision and Pattern Recognition, IEEE, Long Beach, USA, pp. 2815–2823, 2019. DOI: 10.1109/CVPR.2019.00293.

X. T. Gao, Y. R. Zhao, L. Dudziak, R. D. Mullins, C. Z. Xu. Dynamic channel pruning: Feature boosting and suppression. In Proceedings of the 7th International Conference on Learning Representations, New Orleans, USA, 2019.

J. H. Luo, J. X. Wu, W. Y. Lin. ThiNet: A filter level pruning method for deep neural network compression. In Proceedings of $IEEE$ International Conference on Computer Machine Intelligence Research 20(6), December 2023
Vision, Venice, Italy, pp. 5068–5076, 2017. DOI: 10.1109/ICCV.2017.541.

[58] M. X. Tan, Q. V. Le. EfficientNet: Rethinking model scaling for convolutional neural networks. In Proceedings of the 36th International Conference on Machine Learning, PMLR, Long Beach, USA, pp. 6105–6114, 2019.

Ming-Yang Zhang received the B.Sc. degree in automation from Zhejiang University City College, China in 2017. He is currently a Ph.D. degree candidate in control theory and engineering at Department of Information and Engineering, Zhejiang University of Technology, China. His research interests include model compression, neural architecture search and machine learning.

E-mail: 1111903012@zjut.edu.cn (Corresponding author)
ORCID iD: 0000-0001-7862-0566

Xin-Yi Yu received the B.Sc. degree from Harbin University of Science and Technology (HUST), China in 2002, the M.Sc. degree from HUST, China in 2005 and the Ph.D. degree from Harbin Institute of Technology, China in 2009. His research interest is robotics and automation, especially the development and industrialization of industrial robots.

E-mail: yuxy@zjut.edu.cn

Lin-Lin Ou received the Ph.D. degree in control theory and engineering from Shanghai Jiao Tong University, China in 2006. She is currently a professor with Department of Automation, Zhejiang University of Technology, China.

Her research interests include theoretical aspects of time-delayed control systems, applications to industrial process control, robot control and cooperative control.

E-mail: linlinou@zjut.edu.cn