Efficient Search for the Number of Channels for Convolutional Neural Networks

Hui Zhu[1,2], Zhulin An[1,*], Chuanguang Yang[1], Xiaolong Hu[1], Kaiqiang Xu[1], Yongjun Xu[1]
[1]*Institute of Computing Technology, Chinese Academy of Sciences, China*
[2]*University of Chinese Academy of Sciences, China*
{zhuhui, anzhulin, yangchuanguang, huxiaolong18g, xukaiqiang, xyj}@ict.ac.cn

***Abstract*—Latest algorithms for automatic neural architecture search perform remarkably but few of them can effectively design the number of channels for convolutional neural networks and consume less computational efforts. In this paper, we propose a method for efficient automatic search which is special to the widths of networks instead of the connections within neural architectures. Our method, functionally incremental search based on function-preserving, will explore the number of channels for almost any convolutional neural network rapidly while controlling the number of parameters and even the amount of computations (FLOPs). On CIFAR-10 and CIFAR-100 classification, our method using minimal computational resources (0.41∼1.29 GPU-days) can discover more effective rules of the widths of networks to improve the accuracy (a∼1.08 on CIFAR-10 and b∼2.33 on CIFAR-100) with fewer number of parameters.**

***Index Terms*—efficient search, function-preserving, the number of channels, functionally incremental search**

I. Introduction

In recent years, deep learning has achieved great success in the field of computer vision. As the crucial factor affecting the performance of convolutional neural networks, many excellent neural architectures have been designed, such as VGG [1], ResNet [2], PyramidNet [3], SENet [4], HCGNet [5] and so on. Although these human-designed architectures constantly refresh the classification accuracy on specific datasets, they rely heavily on expert experience and it is extremely difficult to manually design suitable neural architectures when faced with brand new image tasks. With more and more attention paid to the research of automatic neural architecture search, many prominent neural architectures have been discovered by various kinds of search algorithms [6]–[9]. Some of them have gradually surpassed the human-designed neural architectures in performance.

The choice of the number of channels has become an important research point since the initial human-designed convolutional neural networks were designed. Many classical convolutional neural networks, such as AlexNet [10], VGG and ResNet usually sharply increase the feature map dimension (the number of channels) at downsampling locations in order to roughly increase the diversity of high-level attributes. Several subsequent researches have improved the number of channels for these thin and deep networks and achieve better results. For example, WRNs [11] increase the widths of residual networks and PyramidNet [3] design the additive and multiplicative pyramid shapes to gradually increase the number of channels. With the continuous development in the filed of neural architecture search, many algorithms [12], [13] tend to seriously consider the number of channels for convolutional neural architectures. Although these methods have achieved outstanding results, they are all computationally expensive and the search efficiency is not satisfactory.

In this paper, we propose a novel method based on function-preserving for efficiently searching the widths of convolutional neural networks. Function-preserving transformations derive from Net2Net [14], which target at transferring the information in the well-trained teacher networks into the student networks rapidly. This method has been well used in network morphism [15] and several neural architecture search algorithms [16], [17]. Differently, we pay more attention on the number of channels instead of the connections within neural architectures. We propose the functionally incremental search which based on function-preserving to efficiently explore the number of channels of almost any network. We conducted confirmatory experiments on several classical convolutional neural networks and improved the performances of the original networks while controlling the number of parameters of the models. Further, we can discover search results with different advantages, such as fewer FLOPs but better accuracy, by modifying the fitness evaluation function. Based on the comparisons with other methods and a series of additional supplementary experiments, we further discuss the rules of the number of channels for convolutional neural networks and analyse the efficiency and the effectiveness of our method.

The results of our experiments show that our search method achieves better classification accuracy (accuracy can be improved by about 0.5% on CIFAR-10 and a∼2.33% on CIFAR-100) with fewer number of parameters by only optimizing the number of channels for classical convolutional neural networks. By taking into account FLOPs, our search method can discover more effective widths of networks (accuracy can be improved by about b∼1.08% on CIFAR-10) with fewer number of parameters and similar amount of computations. The efficient search process consumes approximately 0.4∼1.3 GPU-days[1] computational resources which mainly depends on the complexity of networks.

Our contributions are summarized as follows:

*Corresponding author of this work

[1]All of our experiments were performed using a NVIDIA Titan Xp GPU.

978-1-7281-6926-2/20/$31.00 ©2020 IEEE

- We propose an efficient method, functionally incremental search, which can explore the number of channels for almost any convolutional neural network rapidly while controlling the number of parameters.
- We can discover search results with different advantages, such as fewer FLOPs but better accuracy, by modifying the fitness evaluation function.
- We conducted experiments on CIFAR-10 and CIFAR-100, and further analyse the effectiveness and transferability of the results discovered by our search method.

Part of the code implementation is available at https://github.com/Search-Width/Search-the-Number-of-Channels.

II. Related Work

In this section, we review the number of channels designed manually, function-preserving transformations and neural architecture search which are most related to this work.

a) The Number of Channels Designed Manually: For convolutional neural networks, appropriate depth and width choices may achieve higher accuracy and smaller model size. Even a slight change in the number of channels for the neural architecture will have a huge impact on the performance. Based on ResNet [2] blocks, Zagoruyko and Komodakis [11] propose the wide residual networks (WRNs) which decrease depth and increase width of residual networks and show that these are far superior over their commonly used thin and very deep counterparts. In addition, for many classical convolutional neural networks such as VGG [1] and ResNet, the number of channels is sharply increased at downsampling locations. It is considered to increase the diversity of high-level attributes and then roughly ensure effective performance of networks. Instead of this method, PyramidNet [3] gradually increases the feature map dimension at all units to involve as many locations as possible and the design has proven to be an effective means of improving generalization ability. From the existing researches as above, we can notice that convolutional neural networks can be improved by manually designing the number of channels but it relies heavily on human experience.

b) Function-Preserving Transformations: Transferring a well-trained neural network to a new one with its network function completely preserved mainly contains two methods: Net2Net [14] and network morphism [15]. They can transfer the information stored in the teacher neural network into the student neural network rapidly based on function-preserving. Although they target at the same problem, there are several major differences between them. For example, Net2Net's operations only work for idempotent activation functions while network morphism may handle arbitrary non-linear activation functions.Due to the new network immediately performs as well as the original network rather than spending time passing through a period of low performance, we can quickly change the neural architecture while reducing redundant training. Thus, the search algorithm may explore the number of channels for convolutional neural networks with less search time and computational effort.

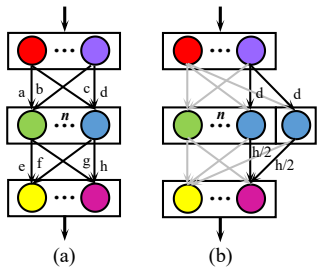

Fig. 1. Widening operation based on functional-preserving. (a) represents the original architecture and (b) represents the architecture after widening. The rectangles and circles represent the convolutional layers and feature maps or filters, respectively. The same color means identical.

c) Neural Architecture Search: The latest algorithms for neural architecture search mainly include evolutionary algorithm (EA) [6], [18], reinforcement learning (RL) [7], [19], [20], Bayesian optimization (BO) [21] and gradient-based methods [9], [22]. The algorithms of automatic neural architecture search are increasingly concerned with their own efficiency (such as search time) besides focusing on the effect of neural architectures discovered. Several methods [16], [17] based on function-preserving perform prominently with less computational effort. However, existing methods basically paid less attention on the number of channels. Fang et al. [12] propose a differentiable method called DenseNAS which can search for the width and the spatial resolution of each block simultaneously. In general, efficient search for the number of channels for convolutional neural networks is an intractable and poorly researched problem. In this work, we propose a novel method based on evolutionary algorithm and function-preserving transformations for efficient search specifically for the number of channels for convolutional neural networks.

III. Proposed Methods

In this section, we propose our methods for efficiently search the number of channels for convolutional neural networks. In the first subsection, we review the fundamental transformation to widen a convolutional layer based on function-preserving. Next, we illustrate in detail the core of our methods, i.e., functionally incremental search. Then, we present the effect and usage of the fitness evaluation function. Finally, we demonstrate the efficient search procedure based on evolutionary algorithm.

A. Widening a Convolutional Layer

Widening a convolutional layer based on function-preserving is the most fundamental transformation which occurs each time the number of channels for every layer is increased after calculating the increment. Due to the new network performs as well as the original one, this operation may implement a smooth and efficient architecture transformation without restarting training.

In transfer learning, the output of the pre-trained teacher network is used as supervisory information to train the student

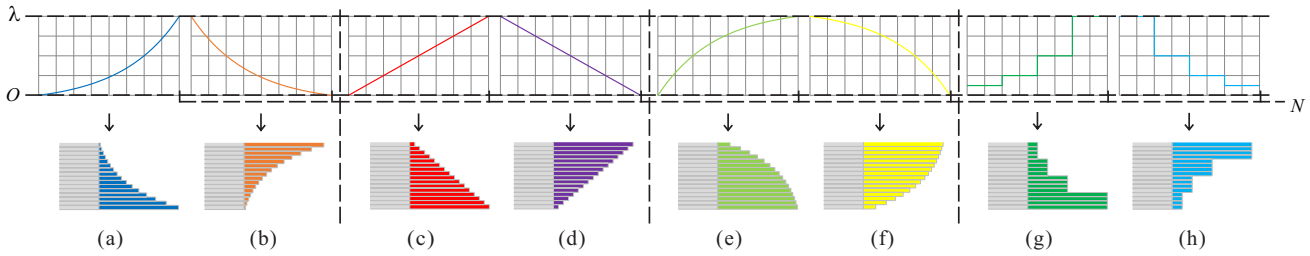

Fig. 2. Visualization of the Functionally incremental search. The top half of (a) $\sim$ (h) are the 8 kinds of functions to incrementally explore the number of channels and the bottom half are the schematic diagrams of the corresponding models after width changes. The range of each function is $0 \sim N$ on the horizontal axis and $0 \sim \lambda$ on the vertical axis.

network. Assume that α is the input to the network, function-preserving transformation is to choose a new set of parameters $\beta^{'}$ for a student network $G(\alpha;\beta^{'})$ which transforms from the teacher network $F(\alpha;\beta)$ such that:

$$\forall\alpha : F(\alpha;\beta) = G(\alpha;\beta^{'}). \tag{1}$$

Assume that the kernel size is $k_1 \times k_2$ and c is the number of input channels of layer i. The number of output feature maps of layer i and layer $i+1$ are f and $f^{'}$, respectively. We denote the old parameters matrix by $W^{(i)} \in \mathbb{R}^{k_1\times k_2\times c\times f}$ and $W^{(i+1)} \in \mathbb{R}^{k_1\times k_2\times f\times f^{'}}$. g represents a random mapping function and U represents the new parameters matrix. The i-th convolutional layer is widened by replicating the parameters along the last axis at random and the parameters in $W^{(i+1)}$ need to be divided along the third axis corresponding to the counts of the same filters in the i-th layer. Specifically, A noise δ is randomly added to every new parameter in layer i+1 to break symmetry. The operation for widening a layer based on function-preserving which is visualized in Fig.1 can be expressed as follows:

$$g(j) = \begin{cases} j & j \le f \\ random\ sample\ from\{1,2,\cdots,f\} & j > f \end{cases}, \tag{2}$$

$$U^{(i)}_{k_1,k_2,c,j} = W^{(i)}_{k_1,k_2,c,g_{(j)}}, \tag{3}$$

$$U^{(i+1)}_{k_1,k_2,j,f'} = \frac{W^{(i+1)}_{k_1,k_2,g(j),f'}}{|\{x|g(x)=g(j)\}|}\cdot(1+\delta),\ \delta\in[0,0.05]. \tag{4}$$

B. Functionally Incremental Search

Functionally incremental search is to widen a network according to the corresponding increments calculated by the specific function on the basis of the current number of channels for every layer of the network. Compared with the mutation for a single layer, the change of architecture for this method is more significant. It is beneficial to get more accurate fitness evaluation for the newly generated network. The procedure of functionally incremental search can refer to the function CHANGE WIDTHS in following Algorithm 1.

1) Designed Functions: We designed more functions with smooth curves than piecewise ones, mainly because the pyramid architecture proved to be more efficient. The functions for incremental search and the schematic diagrams of the corresponding models after width changes are visualized in Fig.2. N represents the number of convolutional layers in the network and the range for the serial number of a single layer which is represented as the independent variable x is $(0, N]$. λ represents the rate of increase, in other words, the number of channels will increase to $1+\lambda$ times at most.

a) Functions with Smooth Curves: The motivation for the functions with smooth curves is mainly derived from additive and multiplicative pyramid-shaped architectures. The function with increasing slop is shown in Fig.2(a) and it can be expressed as:

$$f_a(x) = \lambda\cdot\frac{(\lambda+1)^x-1}{(\lambda+1)^N-1}. \tag{5}$$

The function with constant slop is shown in Fig.2(c) and it can be expressed as:

$$f_c(x) = \frac{\lambda}{N}x. \tag{6}$$

The function with decreasing slop is shown in Fig.2(e) and it can be expressed as:

$$f_e(x) = \lambda\cdot\frac{(\lambda+1)^N-(\lambda+1)^{N-x}}{(\lambda+1)^N-1}. \tag{7}$$

The function shown in Fig.2(b), (d) and (f) are symmetric with the functions in eq.(5), eq.(6) and eq.(7) about $x = N/2$, respectively. They can be expressed as:

$$f_b(x) = \lambda\cdot\frac{(\lambda+1)^{N-x}-1}{(\lambda+1)^N-1}, \tag{8}$$

$$f_d(x) = \lambda - \frac{\lambda}{N}x, \tag{9}$$

$$f_f(x) = \lambda\cdot\frac{(\lambda+1)^N-(\lambda+1)^x}{(\lambda+1)^N-1}. \tag{10}$$

In particular, after the incremental changes according to every two symmetric functions, the number of channels for the convolutional layers in the network may be opposite.

b) Functions with Piecewise Curves: The motivation for the functions with piecewise curves is mainly derived from most human-designed networks. Assume that K_i represents the number of convolution layers before the i-th downsampling operation. Set $n = max\{i\} + 1$, the function with the Step Shape is shown in Fig.2(g) and it can be expressed as:

$$f_g(x) = \begin{cases} \frac{1}{2^{n-1}} \cdot \lambda & 0 < x \leq K_1 \\ \frac{1}{2^{n-2}} \cdot \lambda & K_1 < x \leq K_2 \\ \cdots & \cdots \\ \lambda & K_{n-1} < x \leq N \end{cases}, \tag{11}$$

The function shown in Fig.2(h) is the opposite of the function in eq.(11) and it can be expressed as:

$$f_h(x) = \begin{cases} \lambda & 0 < x \leq K_1 \\ \cdots & \cdots \\ \frac{1}{2^{n-2}} \cdot \lambda & K_{n-2} < x \leq K_{n-1} \\ \frac{1}{2^{n-1}} \cdot \lambda & K_{n-1} < x \leq N \end{cases}. \tag{12}$$

Notice that these two functions are not necessarily symmetric about $x = N/2$ as the number of layers may be different for every two intervals divided by the downsampling operation.

2) Calculate the Increments of the Widths: The methods used to calculate increments mainly include: Calculating by the initial fixed number of channels and calculating by the number of channels that may change during the search process. For the former, Each calculation is based on the initial network in which the number of channels will be unchanged. In this case, several functions (e.g. $f(x) = \lambda$) become useless because their effects can be generated by the addition of other functions. For the latter, each calculation is based on the new network in which the number of channels is changed per round. Assume that Θ is the initial set of the layers in the network and the width of the i-th layer is represented as θ_i. $\Theta^{'}$ is the set after m changes and ξ represents a randomly selected function. It can be expressed as:

$$\Theta = \{\theta_i\}, \quad \Theta^{'} = \left\{ \theta_i \cdot \prod_{p=1}^{m} \left(1 + f_\xi(i)\right) \right\}. \tag{13}$$

In this case, we may add the function $f(x) = \lambda/2$ to ensure that the network with the same number of channels in all convolutional layers can be produced. For the sake of search efficiency and intermediate results, we prefer the latter one in our experiments.

C. Fitness Evaluation Function

The fitness evaluation function is especially important to evaluate and select the individuals. We may finally discover search results with different advantages by modifying fitness evaluation functions. For example, if we only focus on the accuracy of the final model, we can simply define the fitness evaluation function as the accuracy on the validation set. If necessary, we can add FLOPs into the fitness evaluation function to ensure the accuracy and the amount of computations simultaneously. For the part involved FLOPs of this work, we use the following fitness evaluation function:

$$\gamma = \frac{FLOPs_{baseline} - FLOPs_{initial}}{Acc_{baseline} - Acc_{initial}}. \tag{14}$$

$$Fitness = k\gamma \cdot (Acc - Acc_{initial}) - FLOPs. \tag{15}$$

Here, $baseline$ represents the value of compared model and $initial$ represents the value of our initial model. No subscript represents the value of the currently evaluated model. γ is the approximate weight proportion of Acc and $FLOPs$ calculated. k is used to further adjust the proportion of them.

Algorithm 1 Evolutionary Search Algorithm

Input: Initial Model: $\mathcal{M}_{initial}$, Population Size: $\mathcal{P}$, Function Set: $\mathcal{F} = \{f_1, \cdots, f_n\}$, Number of Layers: $\mathcal{N}$, Channel Number List: $\Theta = [\theta_1, \cdots, \theta_{\mathcal{N}}]$, Fitness Evaluation: $\mathcal{F}itness$, Number of Parameters: $\mathcal{P}arams$.
Output: Best Model: $\mathcal{M}_{best}$.

while $\mathcal{P}arams_{best} < \mathcal{P}arams_{threshold}$ **do**
  if $\mathcal{P} < \mathcal{P}_1$ **then**
    $\mathcal{M}_{new} \leftarrow$ CHANGE CHANNELS($\mathcal{M}_{initial}$, $\mathcal{F}$);
  else if $\mathcal{P}_1 \leq \mathcal{P} < \mathcal{P}_2$ **then**
    Select an individual for mutate: $\mathcal{M}_{selected}$;
    $\mathcal{M}_{new} \leftarrow$ CHANGE CHANNELS($\mathcal{M}_{selected}$, $\mathcal{F}$);
  else
    Select an individual for mutate: $\mathcal{M}_{selected}$;
    $\mathcal{M}_{new} \leftarrow$ CHANGE CHANNELS($\mathcal{M}_{selected}$, $\mathcal{F}$);
    Discard $\mathcal{F}itness_{worst}$ individual: $\mathcal{M}_{worst}$;
  end if
  Put $\mathcal{M}_{new}$ into the population;
  Update $\mathcal{F}itness_{best}$, $\mathcal{M}_{best}$, $\mathcal{P}arams_{best}$, in turn;
end while
return $\mathcal{M}_{best}$

function CHANGE WIDTHS($\mathcal{M}_{selected}$, $\mathcal{F}$)
  A randomly selected function from $\mathcal{F}$: f_ξ;
  for $i = 1 : \mathcal{N}$ **do**
    Calculate the increment according to f_ξ: $f_\xi(i)$;
    Update θ_i in $\mathcal{M}_{selected}$: $\theta_i \leftarrow \theta_i(1 + f_\xi(i))$;
  end for
  return $\mathcal{M}_{new}$
end function

D. Search Based on Evolutionary Algorithm

Our search method is based on the evolutionary algorithm. The initial network is randomly mutated several times to form the initial population of size P_1. Then we use tournament selection [23] to select an individual for mutation: a fraction k of individuals is selected from the population randomly and the individual with the highest fitness is final selected from this set. After several rounds, the population size will reach P_2. From this point on, the individual with the lowest fitness in the population will be discard while a new individual is generated through mutation. Thus, the population size remains unchanged (P_2) until the end of the evolution. Detailed search algorithm is described in Algorithm 1.

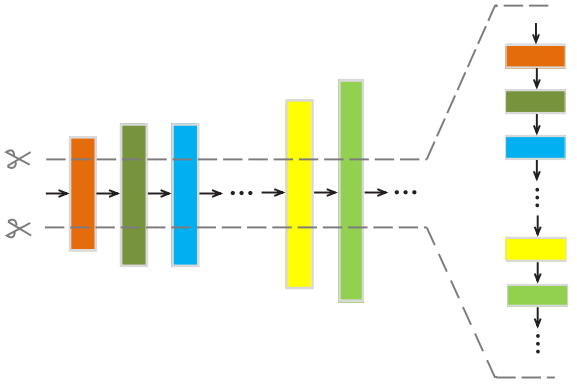

Fig. 3. Select a small number of channels from the classical convolutional neural network to form the initial model for search. Different colored rectangles represent different convolutional layers.

IV. Experiments

In this section, we show the initial model and the dataset first. Then, we introduce the implementation of experiments and report the performances of functionally incremental search. The experiments are mainly implemented in accordance with the methods mentioned in Sect.III and we make a detailed introduction here. Finally, we compare the networks discovered with baselines and other methods to prove the effectiveness and efficiency of our method.

A. Initial Model

The method for constructing the initial model is described in Fig.3. The widths of the original network are usually varied as shown on the left and we choose the same number of channels for every convolutional layer as shown on the right in the schematic diagram. The main purpose of this is to create a large and free enough search space and then we can use our method to discover the widths of networks that may be more suitable for the given dataset. In other words, We will select a small number of channels (usually half of the minimum) from the classical convolutional neural network for search. For example, there are 4 kinds of widths in ResNet-18: 64, 128, 256 and 512. We will construct a small network with the same depth but the number of channels are all initialized to 32. The weights are initialized as He normal distribution [2] and the L2 regularization of 0.0001 is applied to the weights.

B. Dataset

For CIFAR-10 and CIFAR-100, We randomly sample 10,000 images by stratified sampling from the original training set to form a validation set for evaluating the fitness of the individuals while using the remaining 40,000 images for training the individuals during the evolution. We normalize the images using channel means and standard deviations for preprocessing and apply a standard data augmentation scheme (zero-padded with 4 pixels on each side to obtain a 40×40 pixel image, then a 32×32 crop is randomly extracted and the image is randomly flipped horizontally).

C. Search for the Number of Channels

We search for the number of channels with the evolutionary algorithm mentioned above. In our experiments, a mutation represents a functionally incremental change. Each calculation is based on the new network in which the number of channels is changed per round. So we add the function $f(x) = \lambda/2$ besides the other 8 functions and the probability of choosing any mutation is the same from the beginning to the end. Here, we describe the general process of search and some implementation details of different networks respectively.

1) The General Process of Search: In the process of search, the rate of increase λ is fixed to 0.2 or 2. The initial population consists of 12 individuals, each formed by a single mutation from the initial model. The size of k in tournament selection is fixed to 3 and we start to discard individuals when the population size grows to 20. All the networks are trained with a batch size of 128 using SGDR [24] with Nesterov's momentum set to 0.9, initial learning rate $l_{max} = 0.05$, $T_0 = 1$ and $T_{mult} = 2$. The initial models will be trained for different epochs (usually 31 epochs with SGDR) because it takes more epochs to train a large-scale network to convergence. The individuals generated by mutation are trained for 15 epochs and then we evaluate the fitness on the validation dataset. The process is repeated until the number of parameters of the individual with the highest fitness is similar to the comparison network. One search process of ResNet-18 on CIFAR-10 is visualized in Fig.4. The horizontal axis represents the search time. The vertical axis on the left represents the percentage of individuals in each range of fitness as shown in illustration. The vertical axis on the right represents the highest fitness of the population. The bar chart represents the percentage and the line chart represents the change of the highest fitness. Although the process of evolution will slow down gradually, we can notice that the fitness of the population increases steadily and rapidly via our method. Finally, the individual with the highest fitness will be selected and fine-tuned for more training and be used to compare with the original network. Specifically, we search ResNet-18 for several times while other classical convolutional neural networks just once.

2) Some Implementation Details: Here, on the basis of the general process, we additionally illustrate some implementation details for the different networks. This part mainly includes several differences in training, model fine-tuning, fitness evaluation functions and so on.

a) Implementation Details of ResNet-18 and ResNet-34: As the feature map dimension of the network is increased at every unit, we use projection shortcuts conducted by 1×1 convolutions. The two layers of a residual block are changed separately, that is, there is no identity. For ResNet-18, we conduct an additional experiment to show that this operation will not lead to lower accuracy of the network and may even be higer. But for ResNet-34, this operation may lead to slightly lower accuracy than the original network. Besides Cutout and SGDR may have some effect, the crucial cause of this problem has been discussed by He et al. [25]. A projection shortcut

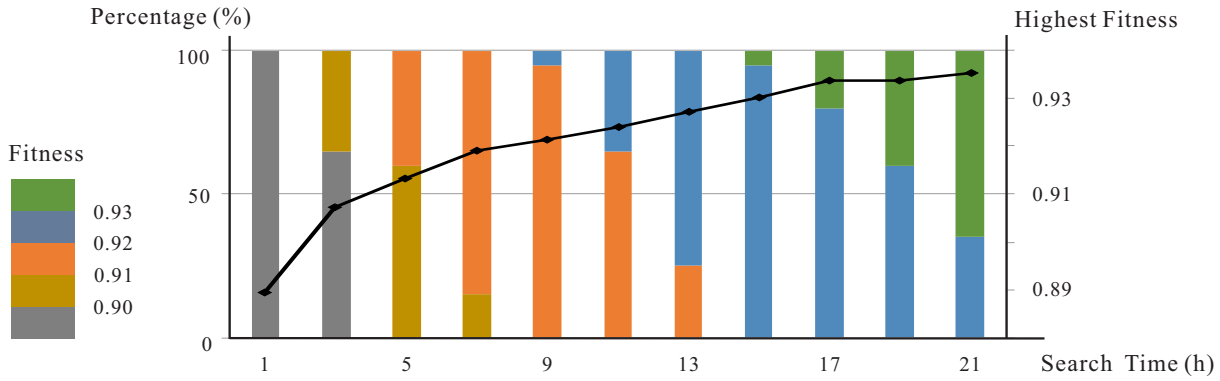

Fig. 4. The schematic diagram visualized one search process of ResNet-18 on CIFAR-10. The horizontal axis represents the search time. The vertical axis on the left represents the percentage of individuals in each range of fitness and that on the right represents the highest fitness of the population. The bar chart represents the percentage and the line chart represents the change of the highest fitness.

TABLE I
COMPARISON AGAINST THE RESULTS OF CLASSICAL NETWORKS ON CIFAR-10 AND CIFAR-100 WHEN THE FITNESS EVALUATION FUNCTION IS DEFINED AS ACCURACY ON THE VALIDATION SET. THE NUMBERS IN BRACKETS DENOTE THE IMPROVEMENT OVER THE BASELINES.

Networks	**Original Networks**			**Modify the Number of Channels**			
	Params (Mil.)	**C10 Test Error** (%)	**C100 Test Error** (%)	**Params** (Mil.)	**C10 Test Error** (%)	**C100 Test Error** (%)	**Search Time** (GPU-days)
ResNet-18 [2]	11.54	3.86	22.76	6.98	$3.69_{(0.17)}$	$20.60_{(2.16)}$	0.48
				9.94	$3.40_{(0.46)}$	$20.74_{(2.02)}$	0.92
				10.57	$3.43_{(0.43)}$	$20.46_{(2.30)}$	0.66
ResNet-34 [2]	22.22	4.71	24.81	13.00	$4.39_{(0.32)}$	$24.44_{(0.37)}$	1.29
VGG-16 [1]	15.00	5.95	28.38	7.24	$5.43_{(0.52)}$	$27.85_{(0.53)}$	0.41
SE-ResNet-50 [4]	26.12	3.81	22.30	11.96	$3.38_{(0.43)}$	$19.97_{(2.33)}$	0.79

can hamper information propagation and lead to optimization problems, especially for very deep networks.

b) Implementation Details of VGG-16: For VGG-16, there are several differences with the original network. We change the number of fully-connected layers to 2, which contain a hidden layer with 512 units and a softmax layer. Dropout [26] layers are added to the convolutional ($drop\_rate = 0.3$ or 0.4) and fully-connected ($drop\_rate = 0.5$) layers. The weights are initialized as Xavier uniform distribution and the L2 regularization of 0.0005 is applied to the weights.

c) Implementation Details of SE-ResNet-50: Not exactly the same as ResNet-34, we use projection shortcuts conducted by 1×1 convolutions but we take the layers in an interval separated by the downsampling operation as a whole. In addition, the 3 layers of the bottleneck block are identical and the ratio of the number of channels is fixed to 1:1:4.

d) Implementation Details of ResNet-20 and ResNet-32: As ResNet mentioned above, we also use projection shortcuts conducted by 1×1 convolutions and the effects of this operation are also similar. Here, we add the index FLOPs into the fitness evaluation function to ensure the accuracy and the amount of computations of the model simultaneously.

e) Implementation Details of MobileNetV2: For simplicity, we use ReLU and normal convolution instead of ReLU6 and depthwise convolution although function-preserving transformation can be used to these cases.

D. Training Methods and Results

The training methods we mentioned here refer to that used for finally training the networks to convergence for comparison of their performances. To be fair, we used exactly the same training methods to train the original classical networks and the best networks discovered by our functionally incremental search. Based on the training results, the comparative analysis proves that our method is effective and efficient.

1) Training Methods: Networks are trained on the full training dataset until convergence using Cutout [27]. All the networks are trained with a batch size of 128 using SGDR [24] with Nesterov's momentum for 511 epochs (several networks may be trained for extra epochs with SGD first). More hyper-parameters are as follows: the cutout size is 16×16 for Cutout, $momentum = 0.9$, $l_{max} = 0.1$,$T_0 = 1$ and $T_{mult} = 2$ for SGDR. Finally, the error on the test dataset will be reported.

2) Comparison of Just Focusing on Accuracy: We define the fitness evaluation function as accuracy on the validation set and the comparison against the results of original networks on CIFAR-10 and CIFAR-100 is presented in Table I. We show the comparison of the number of parameters and the test error on the datasets. In addition, we add the search time of

TABLE II
COMPARISON AGAINST THE RESULTS OF CLASSICAL NETWORKS ON CIFAR-10 WHEN THE FITNESS EVALUATION FUNCTION CONSIDERS THE INDEX FLOPS BESIDES ACCURACY.

Networks	**Original Netwroks**			**Modify the Number of Channels**			
	Params (M)	**FLOPs** (M)	**Test Error** (%)	**Params** (M)	**FLOPs** (M)	**Test Error** (%)	**Search Time** (GPU-days)
ResNet-20 [2]	0.29	42.69	6.08±0.18	0.19	43.49	5.95±0.12	0.51
ResNet-32 [2]	0.49	72.61	6.97±0.04	0.47	71.44	5.89±0.05	1.06
MobileNetV2 [28]	2.29	83.19	6.32±0.19	2.05	85.14	6.15±0.21	0.88

TABLE III
COMPARISON WITH OTHER METHODS.

Method	**Compression Rate** (%)	**Acc. Improved** (%)
Variational Convolutional [29]	20.41	-0.35
Soft Filter Pruning [30]	15.20	+0.04
Ours (Search-Widths)	**34.48**	**+0.13**

TABLE IV
COMPARISON OF RESULTS FOR DIFFERENT RULES OF THE NUMBER OF CHANNELS FOR PYRAMIDNET-110 ON CIFAR-10.

Method	**FLOPs** (G)	**Params** (M)	**C10 Test Error** (%)
Original PyramidNet-110	0.76	3.87	3.80
Modification Considered Accuracy & FLOPs	0.72	2.87	3.65
Modification just Focused on Accuracy	-	3.86	3.14

our method to show the efficiency of the search. Specifically, the number of parameters of the networks used to compare the test error on CIFAR-100 are slightly more than that on CIFAR-10 because of the last fully-connected layer (10-way and 100-way). Since the number of parameters are almost the same, we only show the one for CIFAR-10 in the table. Our method is suitable for exploring the number of channels of almost any network rapidly and we select several classical networks for experiments. We can notice that our method using minimal computational resources (0.41∼1.29 GPU-days) can discover more effective widths of networks (the accuracy can be improved by about 0.5% on CIFAR-10 and 0.37%∼2.33% on CIFAR-100 with fewer number of parameters).

3) Comparison of Considering Accuracy and FLOPs: We add the index FLOPs into the fitness evaluation function and the comparison against baselines on CIFAR-10 is presented in Table II. We show the comparison of the number of parameters and the mean error with standard deviation on test dataset in the case of the similar amount of computations. We can notice that our method using minimal computational resources (0.51∼1.06 GPU-days) can discover more effective widths of networks (the accuracy can be improved by 0.13%∼1.08% with fewer number of parameters and similar FLOPs).

4) Comparison with Other Methods: Most neural architecture search algorithms pay more attention to exploring architectures instead of specially focusing on the number of channels, because it's easier to make a difference. For this reason, it is obviously unfair to directly compare the results discovered by our method with theirs. However, there are some approaches that only focuses on the exploration about the widths of networks for comparison, such as pruning on ResNet-20. As is shown in Table III, our method achieves better accuracy while compressing more parameters.

E. Supplementary Experiments and Discussions

Based on the rules searched by our method for different networks, we can notice that the number of channels discovered are quite different from those designed manually by predecessors. In this section, we conducted a series of additional supplementary experiments for PyramidNet to further analyse the rationality and transferability of the search results.

1) Additional Supplementary Experiments: We conducted a series of supplementary experiments by transferring the search results for ResNet to the similar architecture PyramidNet. For PyramidNet-110, we select the additive PyramidNet (the performance is slightly better than multiplicative PyramidNet) whose widening factor $\alpha = 84$ and the feature map dimension of the first convolutional layer is 16. We consider the option zero-padded identity mapping shortcuts used in original PyramidNet and only use projection shotcuts when decreasing feature map dimensions. As comparisons, we change the original number of channels for PyramidNet to that designed according to the rules of the number of channels we discovered by our method for ResNet. As is shown in Table IV, we can obviously notice that search with the both fitness evaluation functions can finally discover a network with better performance which we attribute to the more reasonable rules of widths.

2) Discussions on the number of channels: Combined with our experiment results, we can notice that the performance of increasing the feature map dimension sharply at downsampling locations is not very excellent. The rule for the widths of convolutional neural networks is still worth further exploring and it is even possible that the most suitable rule on a particular dataset is not necessarily continuously increasing. On the basis of our search results, the rules for the number of channels may be different according to the performance we focus more on. Our search results turned out to be more competitive and it seems like a good solution to use our search method to explore

TABLE V
THE NUMBER OF CHANNELS USED TO COMPARE THE CLASSICAL NETWORKS WITH THE ONES MODIFIED BY OUR METHOD.

Method	Original networks	Modify the Number of Channels
RenNet-20	16×6,32×6,64×6	16,18,24,24,22,22,24,24,24,26,34,34,40,40,44,48,50,64
RenNet-32	16×10,32×10,64×10	16,18,18,18,16,16,18×4,26×6,30,30,32,32,60,60,62,62,64,64,66,66,68,68
MobileNetV2	16×1,24×2,32×3,64×4,96×3,160×3,320×1	16×1,32×2,36×3,48×4,90×3,156×3,288×1
PyramidNet-110	{16,17,19,20,22,23,25,···,100}×2	30×4,32×8,34×4,{40,36}×12,40×16,42×8,{54,64}×12,72×8,78×6,84×4,100×4
		60×2,{62,70,78,58,62,66,68,56,60,60,60,56,56}×8,54×4

a complex and fluctuating rule of the number of channels which is suitable for any given dataset.

V. CONCLUSIONS

We proposed an efficient method based on function-preserving to search the number of channels for convolutional neural networks. The classical convolutional networks with the widths explored by the functionally incremental search perform better than the original ones. The networks modified by our search method contain fewer parameters and fewer FLOPs but may achieve higher accuracy on the test datasets.

APPENDIX

In Table V, we publish the number of channels used for the baselines and the ones searched by our methods.

REFERENCES

[1] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[2] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, 2016, pp. 770–778.

[3] D. Han, J. Kim, and J. Kim, "Deep pyramidal residual networks," in *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pp. 6307–6315.

[4] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 7132–7141.

[5] C. Yang, Z. An, H. Zhu, X. Hu, K. Zhang, K. Xu, C. Li, and Y. Xu, "Gated convolutional networks with hybrid connectivity for image classification," *CoRR*, vol. abs/1908.09699, 2019. [Online]. Available: http://arxiv.org/abs/1908.09699

[6] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," *CoRR*, vol. abs/1802.01548, 2018.

[7] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, 2018, pp. 4092–4101.

[8] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. Li, L. Fei-Fei, A. L. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part I*.

[9] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," *CoRR*, vol. abs/1806.09055, 2018.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States.*, 2012.

[11] S. Zagoruyko and N. Komodakis, "Wide residual networks," in *Proceedings of the British Machine Vision Conference 2016, BMVC 2016, York, UK, September 19-22, 2016*, 2016.

[12] J. Fang, Y. Sun, Q. Zhang, Y. Li, W. Liu, and X. Wang, "Densely connected search space for more flexible neural architecture search," *CoRR*, vol. abs/1906.09607, 2019.

[13] M. Tan and Q. V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, 2019, pp. 6105–6114.

[14] T. Chen, I. J. Goodfellow, and J. Shlens, "Net2net: Accelerating learning via knowledge transfer," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[15] T. Wei, C. Wang, Y. Rui, and C. W. Chen, "Network morphism," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, pp. 564–572.

[16] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang, "Efficient architecture search by network transformation," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, 2018, pp. 2787–2794.

[17] H. Zhu, Z. An, C. Yang, K. Xu, and Y. Xu, "EENA: efficient evolution of neural architecture," *CoRR*, vol. abs/1905.07320, 2019.

[18] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*, 2018.

[19] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[20] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 8697–8710.

[21] K. Kandasamy, W. Neiswanger, J. Schneider, B. Póczos, and E. P. Xing, "Neural architecture search with bayesian optimisation and optimal transport," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 2020–2029.

[22] R. Luo, F. Tian, T. Qin, E. Chen, and T. Liu, "Neural architecture optimization," in *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pp. 7827–7838.

[23] D. E. Goldberg and K. Deb, "A comparative analysis of selection schemes used in genetic algorithms," in *Proceedings of the First Workshop on Foundations of Genetic Algorithms. Bloomington Campus, Indiana, USA, July 15-18 1990.*, 1990.

[24] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with warm restarts," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017.

[25] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, 2016, pp. 630–645.

[26] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[27] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *CoRR*, vol. abs/1708.04552, 2017.

[28] M. Sandler, A. G. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, 2018, pp. 4510–4520.

[29] C. Zhao, B. Ni, J. Zhang, Q. Zhao, W. Zhang, and Q. Tian, "Variational convolutional neural network pruning," in *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, 2019, pp. 2780–2789.

[30] Y. He, G. Kang, X. Dong, Y. Fu, and Y. Yang, "Soft filter pruning for accelerating deep convolutional neural networks," in *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI 2018, July 13-19, 2018, Stockholm, Sweden*, 2018, pp. 2234–2240.