You Only Search Once: Single Shot Neural Architecture Search via Direct Sparse Optimization

Xinbang Zhang, Zehao Huang, Naiyan Wang, Shiming Xiang, and Chunhong Pan

Abstract—Recently neural architecture search (NAS) has raised great interest in both academia and industry. However, it remains challenging because of its huge and non-continuous search space. Instead of applying evolutionary algorithm or reinforcement learning as previous works, this paper proposes a direct sparse optimization NAS (DSO-NAS) method. The motivation behind DSO-NAS is to address the task in the view of model pruning. To achieve this goal, we start from a completely connected block, and then introduce scaling factors to scale the information flow between operations. Next, sparse regularizations are imposed to prune useless connections in the architecture. Lastly, an efficient and theoretically sound optimization method is derived to solve it. Our method enjoys both advantages of differentiability and efficiency, therefore it can be directly applied to large datasets like ImageNet and tasks beyond classification. Particularly, on the CIFAR-10 dataset, DSO-NAS achieves an average test error 2.74 percent, while on the ImageNet dataset DSO-NAS achieves 25.4 percent test error under 600M FLOPs with 8 GPUs in 18 hours. As for semantic segmentation task, DSO-NAS also achieve competitive result compared with manually designed architectures on the PASCAL VOC dataset. Code is available at https://github.com/XinbangZhang/DSO-NAS

Index Terms—Neural architecture search(NAS), convolution neural network, sparse optimization

1 Introduction

With no doubt, Deep Neural Networks (DNN) have been the engines for the Artificial Intelligence (AI) renaissance in recent years. Dating back to 2012, DNN based methods have refreshed the records for many AI applications, such as image classification [1], [2], [3], speech recognition [4], [5] and Go Game [6], [7]. Considering its amazing representation power, DNNs have shifted the paradigm of these applications from manually designing the features and stagewise pipelines to end-to-end learning. Although DNNs have liberated researchers from such feature engineering, another tedious work has emerged–"network engineering". In most cases, the neural networks need to be designed based on specific tasks, which again leads to endless hyperparameters tuning. Therefore, designing a suitable neural network architecture still requires considerable amounts of expertise and experience.

To democratize the techniques, Neural Architecture Search (NAS) or more broadly, Automated Machine Learning (AutoML) has been proposed. There are mainly three streams for NAS, as shown in Fig. 1. The first popular algorithm for architecture search is the evolutionary algorithm (EA), which iteratively evaluates and proposes new models [8], [9]. The second one is based on reinforcement learning (RL). Several works [10], [11], [12] adopt a recurrent network as controller to generate architectures and optimize the controller by reinforcement learning. The accuracy of trained network structure is used as the reward signal. Despite their impressive performance, the search processes are incredibly resource-hungry and unpractical for large datasets like ImageNet, though some acceleration methods have been proposed [12], [13]. The last algorithm searches suitable architectures by gradient-based optimization. The recently proposed method, DARTS [14], is one of the pioneering methods. Based on a continuous relaxation search space, DARTS formulates NAS as a bilevel optimization problem [15] and optimizes architecture by gradient descent. Although DARTS achieves impressive performance with great acceleration, they search the structure of blocks on a smaller dataset with a proxy network that consists of fewer blocks, then the obtained block structure is shared among all blocks in the target network. However, searching in the proxy task is inferior due to the gap between the proxy and the target task. Besides, the search space of DARTS is also limited to single operation for single node, due to the fixed length coding.

In this work, we take another view to address these problems. We formulate NAS as a pruning problem. Our method finds suitable architectures by pruning the useless connections from a single super-network which contains a large network architecture hypothesis space. A sparse

- *Xinbang Zhang and Shiming Xiang are with the Department of National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Science, Beijing 100190, China, and also with the School of Artificial Intelligence, University of Chinese Academy of Sciences, Beijing 100049, China. E-mail: {xinbang.zhang, smxiang}@nlpr.ia.ac.cn.*
- *Zehao Huang and Naiyan Wang are with Tusimple, Beijing 100020, China. E-mail: {zehaohuang18, winsty}@gmail.com.*
- *Chunhong Pan is with the Department of National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Science, Beijing 100190, China. E-mail: chpan@nlpr.ia.ac.cn.*

Manuscript received 17 Oct. 2019; revised 15 June 2020; accepted 12 Aug. 2020. Date of publication 31 Aug. 2020; date of current version 4 Aug. 2021.
Recommended for acceptance by H. J. Escalante, J. Vanschoren, W.-W. Tu, Y. Yu, S. Escalera, N. Pillay, R. Qu, N. Houlsby, and T. Zhang.
Digital Object Identifier no. 10.1109/TPAMI.2020.3020300

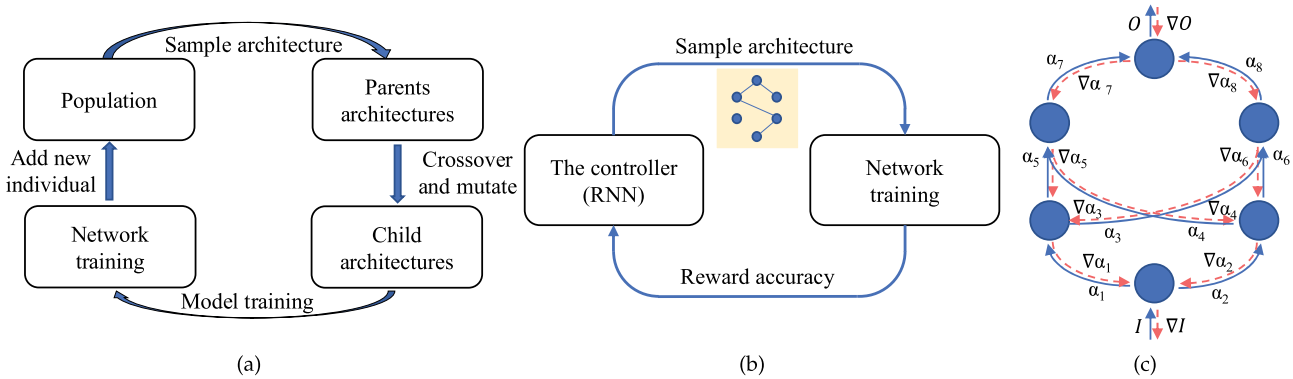


Fig. 1. Three mainstream methods of neural architecture search: (a) Evolution, where a population of architectures is updated based on their fitness, e.g., performance on the validation set. New individuals are produced by crossover and mutate operation.Reinforcement learning, where an RNN controller is applied to sample architectures and trained based on the performance of the architectures. (b) Reinforcement learning, where an RNN controller is applied to sample architectures and optimized based on the performance of the sampled architectures. (c) Gradient based method, where all paths in hypenetwork are parameterized with architecture parameters, both weights and architecture parameters are updated with gradients.

regularization is adopted to removing the unimportant connections. Since the network structure is directly optimized based on the target network and datasets with no proxy task, we name our method Direct Sparse Optimization NAS (DSO-NAS). We further demonstrate that this sparse regularized problem can be efficiently optimized by a modified accelerated proximal gradient method opposed to the inefficient reinforcement learning or evolutionary search. DSO-NAS does not need any controller [10], [11], [12] or performance predictor [16] or relaxation of the search space [10], [11], [12], [14]. It can also incorporate the search process with different hardware budgets easily. As a result of the efficiency and simplicity, *DSO-NAS demonstrates that block structure sharing is not necessary for architecture search and NAS can be directly applied on large scale datasets*. Our experiments show that DSO-NAS can achieve 2.74 percent average test error on CIFAR-10, as well as top-1 error 25.4 percent on ImageNet with FLOPs (the number of multiply-adds) under 600M and comparable result on the PASCAL VOC semantic segmentation task. In summary, our contributions can be summarized as follows:

- We propose a model pruning formulation for neural architecture search based on sparse optimization. Only one model needs to be trained during the search.
- A modified stochastic Accelerated Proximal Gradient (APG) method is adopted to solve this challenging optimization problem with higher efficiency.
- Extensive experiments demonstrate that our method achieves competitive results on CIFAR, ImageNet for classification and PASCAL VOC for semantic segmentation. Specifically, DSO-NAS achieve 2.74 error rate with only one GPU day on CIFAR-10.

2 Related Work

In this section, we briefly review two research fields related to our work. We will first review the related work about network pruning, followed by the three steams of neural architecture search methods, i.e., evolution-based NAS, reinforcement learning-based NAS and gradient-based NAS.

2.1 Network Pruning

Network pruning is a widely used technique for model acceleration and compression. The early works focus on removing unimportant connections in neural networks, such as Optimal Brain Damage [17], Optimal Brain Surgeon [18], Dynamic Network Surgery [19] and so on [20], [21]. Though connection-level pruning yields effective compression, it is hard to harvest actual computational savings because modern GPU cannot utilize the irregular weights well. To tackle this issue, a significant amount of works focusing on structure pruning have been proposed [22], [23], [24]. For neuron-level pruning, several works [25], [26], [27], [28], [29] prune the neurons directly by evaluating the importance of neurons based on specific criteria. More generally, [30], [31] propose sparse structure learning. They adopt group sparsity on multiple structures of networks, including filter shapes, channels and layers. Recently, [32] proposes a simpler way for structure pruning. They introduce scaling factors to the outputs of specific structures (neurons, groups or blocks) and apply sparse regularizations on them. After training, structures with zero scaling factors can be safely removed. Similarly, [33] introduces scaling factors to indicate the importance of every neuron and prunes the neurons with the minimum scalers. Unfortunately, the pruning ratio for every layer is required to be predefined, limiting the board application of it. More recently, [34], [35] integrate reinforcement learning into model compression. For efficient search, the action space is constrained as pruning ratio. Though they only apply this method on channel selection, it also can be seen as an architecture search approach. As shown by [36], the pruned architecture is more important than inheriting obtained weights. In other words, pruning is naturally a special kind of architecture search that focuses on searching specific filter configuration of each layer. However, the process of pruning is strictly based on specific architecture, the search space of it is insufficient to explore new network topology and operations. Therefore, we extend the idea of pruning into a more general and harder case.

2.2 Evolution-Based NAS

Recently, there has been growing interest in developing methods to generate neural architecture automatically. One heavily investigated direction is evolutionary algorithm.

Historically, evolutionary algorithms have already been heavily investigated and applied in evolving neural architectures [37], [38], [39], [40]. Recently, [8], [9] bring this idea back by applying evolutionary algorithms in a CNN search space. Modifications like changing filter sizes, inserting layers or adding identity mapping are designed as the mutations in evolution. These evolution-based NAS methods differ in how sampling parents, updating populations and generating offspring. [8] removes the individual with the worst performance from the population while [9] removes the oldest one. To get suitable architectures with low resource-consumption, [41], [42] incorporate hardware constraint, i.e., model size, inference time, into the evaluation of child architectures and propose a multi-objective genetic algorithm. Though they achieve remarkable results, training and evaluating the generated offspring are extremely computationally expensive. As a result, Evolution-based NAS are source-hunger and less practical at large scale even some acceleration methods like weight inheritance [8], [41] are applied.

2.3 Reinforcement Learning-Based NAS

Several works use reinforcement learning [43] to optimize the neural architecture. The pioneering work [10] adopts an RNN network as the controller to sequentially decide the type and parameters of layers. Then the controller is trained by reinforcement learning with the reward designed as the accuracy of the proposed model. Although it achieves remarkable results, it needs 800 GPUs to get such results, which is not affordable for broad applications. Based on this work, several methods have been proposed to accelerate the search process by limiting the search space [11], early stopping with performance prediction [13], progressive searching [16] or weight sharing among child architectures [12]. Specifically, One Shot NAS [44] gets rid of controller by randomly sampling architecture candidates and evaluating them on a pre-trained over-parameters network that contains all candidate paths. To achieve a good trade-off between architecture latency and performance, [45] builds its search space with mobile inverted bottleneck convolution [46] and explicitly incorporates model latency into the reward of reinforcement learning. Unfortunately, [45] is still resource hunger and suffers from limited search space due to the fixed-length coding of architecture.

2.4 Gradient-Based NAS

Contrary to treating architecture search as a black-box optimization problem, gradient-based NAS methods utilize gradients to optimize neural architecture. Subjecting to the discreteness of architecture, [14] applies a continuous relaxation to the search space and optimizes the proposed architecture parameters with gradient directly. After searching, the operations with the highest architecture parameter will be selected to generate the new architecture. Similar relaxation is also applied to search for the feature extractor for multi-task learning [47] and hardware-friendly architecture [48]. As a great improvement of [14], [49] applies an early stopping strategy to prevent the collapse of the search process. [48] proposes to optimize the pre-defined supernet directly. Though the building of the search space is similar to our method, their network structure is limited to ResNet fabric and convolutional neural fabrics. Another stream of researchers optimize discrete architecture directly with discrete optimization methods like Gumbel-Softmax [50], [51], concrete optimization [52] and binary optimization [53]. Despite the remarkable results they achieve, their search spaces are also limited, the number of inputs to every operation is determined compulsorily according to their discretion rules, leading to the strict restraint on search space. As a consequence, various gradient-based NAS methods have to design their search spaces elaborately to achieve higher performance [53].

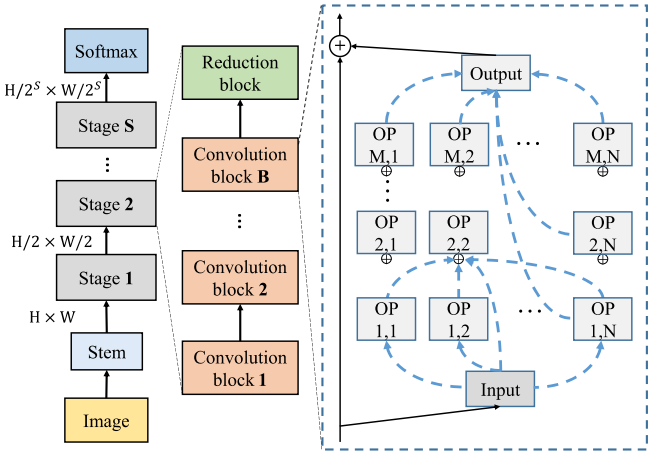


Fig. 2. We divide a network into **S** stages according to the size of feature maps. A stage consists of **B** convolution blocks and ends up with a reduction block. A block composed of M levels and every level contains N different operations. Following [14], we adopt differnet stem structure for CIFAR and ImageNet.

3 Proposed Method

In this section, we will describe the details of our proposed method. We will start with the design of the search space, followed by the motivations and the formulation of our method. Lastly, we will describe the optimization and training details.

3.1 Search Space

Instead of searching structure on a proxy network with fewer building blocks, we directly search on the target network. In the following, we will discuss two different search spaces adopted in out method, the novel normal setting and the hardware-friendly mobile setting introduced by [53].

The structure of whole network is shown in Fig. 2: a network consists of S stages with B convolution blocks in every stage. Reduction block that reduces the feature map into half is located at the end of each stage except for the last stage. All operations in the convolution block are of stride one to maintain the size of feature map. We define any block in the network consists of M sequential levels that are composed of N different kinds of operations. In each block, every operation has connections with all the operations in the former levels and the input of the block. Also, the output of the block is connected with all the operations in the block. Formally, the output of the jth operation in the ith layer of the bth block $\mathbf{h}_{(b,i,j)}$ is computed as

$$\mathbf{h}_{(b,i,j)} = \mathcal{O}_{(b,i,j)}\left(\sum_{m=1}^{i-1}\sum_{n=1}^{N}\mathbf{h}_{(b,m,n)} + \mathbf{O}_{(b-1)}\right), \tag{1}$$

where $\mathcal{O}_{(b,i,j)}$ is the transformation of the jth operation in the ith layer of the bth block and $\mathbf{O}_{(b-1)}$ is the output of the $(b-1)$th block. Here we denote $\mathbf{h}_{(b,0,0)} = \mathbf{O}_{(b-1)}$ as the input node and $\mathbf{h}_{(b,M+1,0)} = \mathbf{O}_{(b)}$ as the output node of the bth block, respectively. The operation in the mth layer may have $(m-1)N+1$ inputs. Note that the connections between operations and the output of block are also learnable. The output of the bth block $\mathbf{O}_{(b)}$ is obtained by applying a reduction operation (concatenation followed by a convolution with kernel size 1×1) $\mathcal{R}$ to all the nodes that have contribution to the output, we also apply identity mapping in case all operations are pruned.

$$\begin{aligned}\mathbf{O}_{(b)} = \mathcal{R}\big([\mathbf{h}_{(b,1,1)}, \mathbf{h}_{(b,1,2)}, \ldots, \mathbf{h}_{(b,m,n)}, \ldots \mathbf{h}_{(b,M,N)}]\big) \\ + \mathbf{O}_{(b-1)}, m \in [1, M], n \in [1, N].\end{aligned} \tag{2}$$

Next, we will introduce two detailed settings under our definition of search space.

Normal Setting. We adopt four kinds of operations in convolution block following [12]: separable convolution with kernel 3×3 and 5×5, average pooling with kernel 3×3 and max pooling with kernel 3×3. Each convolution is applied in the order of conv-batchnorm-ReLU. As for reduction block, we simply use convolutions with kernel size 1×1 and 3×3, and apply them with a stride of 2 to reduce the size of feature map and double the number of filters. The output of the reduction block is the sum of these two convolutions. Although we only applied four types of operation in our search space, the connections between them can be extremely complicated. As for a block consisting of M levels and N operations, the number of individual connections can be as large as $(M^2-M)N^2/2 + 2MN - N$. Specifically, for a block structure with 4 levels and 4 operations, it contains 124 independent connections, indicating a huge search space with 2^{124}, roughly 10^{37} candidates. We try two search scopes: (1) the share search scope, where structure is shared among blocks. (2) the full search scope, where different blocks have different structures.

Mobile Setting. Although the normal setting is huge and can cover most of architecture candidates, it is unfriendly for realtime applications due to the fragmentation of network [54]. Therefore, we incorporate DSO-NAS with a different search space proposed by [53]. This search space is built with MobilenetV2 as the backbone. Every block contains only one level, mobile inverted bottleneck convolutions (MBConv) with different kernel size and expansion ratios are applied as operations. Following [53], we apply the GPU and CPU network structure and the six operations: MBConv layers with kernel sizes 3×3, 5×5, 7×7, expansion ratios 3 and 6. As there is only one level in each block, the output of the $i-$th operation in the bth block $\mathbf{h}_{(b,i)}$ is computed as

$$\mathbf{h}_{(b,i)} = \mathcal{O}_{(b,i)}\big(\mathbf{O}_{(b-1)}\big), \tag{3}$$

where $\mathcal{O}_{(b,i)}$ is the transformation of the ith operation of the bth block, $\mathbf{O}_{(b-1)}$ is the output of the $(b-1)$th block, and the output of the bth block $\mathbf{O}_{(b)}$ is obtained by applying a reduction operation (sum) to all the operations and the residual connection in this block, as illustrated in Fig. 3. The output of the bth block is computed according to the following equation:

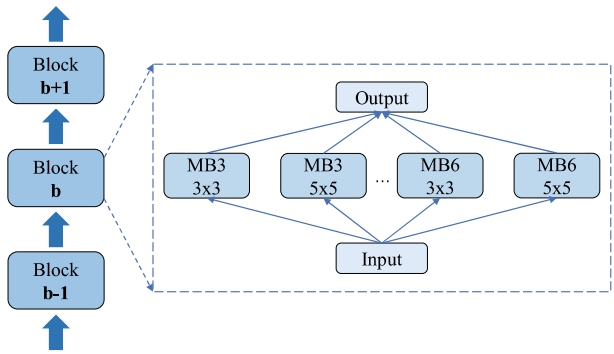


Fig. 3. The search space under mobile setting, where six operations are applied, namely, MBconv block with kernel sizes 3×3, 5×5, 7×7, expansion ratios with 3 and 6.

$$\mathbf{O}_{(b)} = \sum_{i=1}^{N}\mathbf{h}_{(b,i)} + \mathbf{O}_{(b-1)}, \tag{4}$$

where N is the number of operations in the bth block.

Based on our definition of search space, the task of searching suitable architecture can be converted to learning the connections between nodes, which will be introduced later. As the search space under normal setting is widely used and more challenging, most experiments in this paper are based on the normal setting if not stated particularly.

3.2 Problem Formulation

The idea of DSO-NAS follows the observation that the process of NAS can be regarded as obtaining a sub-graph from a super-graph through pruning useless nodes and edges. In other words, the problem of searching for suitable architecture can be reformulated as pruning useless connections and nodes in the full graph. For an individual block with M levels and N operations, a fully connected architecture, or supernet, that keeps all possible connections can be built. The topology of this supernet can be represented by a completely connected Directed Acyclic Graph (DAG) consisting of M levels with $N(m)$ nodes in the mth level.

In this DAG, nodes and edges represent local computations $\mathcal{O}$ (e.g., convolution, pooling) and information flows (e.g., feature map), respectively. The output of jth node in the ith level $\mathbf{h}_{(i,j)}$ can be calculated by transforming the sum of all the outputs of the predecessors $\mathbf{h}_{(m,n)}, m < i$, and the input node $\mathbf{h}_{(0,0)}$, by the local operation $\mathcal{O}_{(i)}$, namely

$$\mathbf{h}_{(i,j)} = \mathcal{O}_{(i,j)}\left(\sum_{m=0}^{i-1}\sum_{n=1}^{N(m)}\mathbf{h}_{(m,n)}\right), \tag{5}$$

Fig. 4 illustrates an example DAG of a specific block. With our representation of individual block, the final network architecture can be represented by stacking blocks with residual connections.

Without loss of generality, any sub-graph can be represented by selecting a subset of connections of the full graph

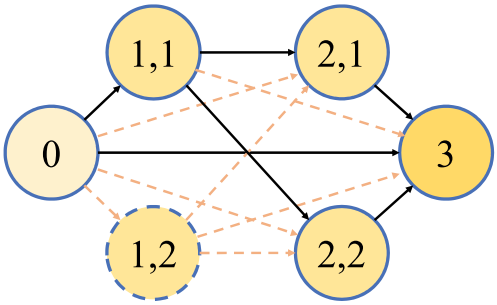


Fig. 4. The whole search space can be represented by a completely connected DAG. Here node 0 and 3 are the input and output node, respectively. The dashed line and dashed circle represent that the corresponding connections and nodes are removed. For example, the initial output of the first node $(2,1)$ in the second level can be calculated by $\mathbf{h}_{(2,1)} = \mathcal{O}_{(5)}(\mathbf{h}_{(0)} + \mathbf{h}_{(1,1)} + \mathbf{h}_{(1,2)})$, while it becomes $\mathbf{h}_{(5)} = \mathcal{O}_{(5)}(\mathbf{h}_{(2)} + \mathbf{h}_{(4)})$ for the pruned sub-graph.

$$\mathbf{h}_{(i,j)} = \mathcal{O}_{(i,j)}\left(\sum_{m=0}^{i-1}\sum_{n=1}^{N(m)} \lambda_{(m,n)}^{(i,j)}\mathbf{h}_{(m,n)}\right), \lambda_{(m,n)}^{(i,j)} \in \{0,1\}, \tag{6}$$

where $\lambda_{(m,n)}^{(i,j)}$ is the scaling factor applied on the information flow from node j to i, signifying whether this connection is utilized in this sub-graph. For a specific architecture, it corresponds to a series of $\boldsymbol{\lambda}$, represented as $\mathcal{F}(\boldsymbol{\lambda}, \mathbf{W}_{\boldsymbol{\lambda}})$ with network weights $\mathbf{W}_{\boldsymbol{\lambda}}$. Intrinsically, the task of neural architecture search can be defined as finding the best $\boldsymbol{\lambda}^*$ that minimizes the validation loss, where corresponding weights are obtained by minimizing the training loss, i.e.,

$$\begin{aligned} &\min_{\boldsymbol{\lambda}} \ \mathcal{L}_{val}(\mathcal{F}(\boldsymbol{\lambda}, \mathbf{W}_{\boldsymbol{\lambda}}^*)), \\ &s.t. \ \mathbf{W}_{\boldsymbol{\lambda}}^* = \arg\min_{\mathbf{W}_{\boldsymbol{\lambda}}} \mathcal{L}_{train}(\mathcal{F}(\boldsymbol{\lambda}, \mathbf{W}_{\boldsymbol{\lambda}})). \end{aligned} \tag{7}$$

Benefiting our binary representation of architecture, the task of NAS can therefore be converted to learning the optimal binary code $\boldsymbol{\lambda}$. However, it is a NP-hard problem which is difficult to solve directly. To solve this problem, we relax the discrete search space to a continuous one by relaxing the scaling factors on every edge to be continuous. Then Eq. (6) can be modified to

$$\mathbf{h}_{(i,j)} = \mathcal{O}_{(i,j)}\left(\sum_{m=0}^{i-1}\sum_{n=1}^{N(m)} \lambda_{(m,n)}^{(i,j)}\mathbf{h}_{(m,n)}\right), \lambda_{(m,n)}^{(i,j)} \geq 0, \tag{8}$$

with this strategy, structure parameters $\boldsymbol{\lambda}$ can be optimized efficiently with backward gradients. Unfortunately, this will induce another obstacle, it is hard to obtain sparse coding of $\boldsymbol{\lambda}$ with normal optimization strategy, resulting in the difficulty of binarizing $\boldsymbol{\lambda}$ without changing the behavior of sub-network. To overcome this issue, we apply sparse regularizations on the scaling parameters to force some of them to be zero during search. Intuitively, if $\lambda_{(m,n)}^{(i,j)}$ is zero, the corresponding edge can be removed safely and isolated nodes can also be pruned as no contribution is made. In this way, the search space is more flexible as the number of selected paths is not limited. Besides, the final architecture can be directly obtained by removing useless connections, without discretization. The whole search process is illustrated in Fig. 5. Considering the regulations for $\boldsymbol{\lambda}$ and $\mathbf{W}_{\boldsymbol{\lambda}}$, the optimization problem can be modified to

$$\begin{aligned} &\min_{\boldsymbol{\lambda}} \ \mathcal{L}_{val}(\mathcal{F}(\boldsymbol{\lambda}, \mathbf{W}_{\boldsymbol{\lambda}}^*)) + \gamma\|\boldsymbol{\lambda}\|_1, \\ &s.t. \ \mathbf{W}_{\boldsymbol{\lambda}}^* = \arg\min_{\mathbf{W}_{\boldsymbol{\lambda}}} \mathcal{L}_{train}(\mathcal{F}(\boldsymbol{\lambda}, \mathbf{W}_{\boldsymbol{\lambda}})) + \delta\|\mathbf{W}\|_F^2. \end{aligned} \tag{9}$$

where δ and γ represent the weights of regularizations for $\mathbf{W}_{\boldsymbol{\lambda}}$ and $\boldsymbol{\lambda}$, respectively. This implies that the essence of NAS is to solve a bi-level optimization problem, which is challenging to optimize as different type of regulations are applied on the architecture parameters and network weights.

3.3 Optimization and Training

The sparse regularization of $\boldsymbol{\lambda}$ induces great difficulties in optimization, especially in the stochastic setting in DNN. Several works [28], [32], [55] have been proposed to solve the sparse optimization problem in DNN. In this paper, we adopt a recently proposed method APG-NAG [32] to solve this problem by modifying a theoretically sound optimization method Accelerated Proximal Gradient (APG) [56].

For better illustration, we shorten $\mathcal{L}_{val}(\mathcal{N}(\boldsymbol{\lambda}, \mathbf{W}_{\boldsymbol{\lambda}}^*))$ in Eq. (9) as $\mathcal{G}(\boldsymbol{\lambda})$ and represent the soft-threshold operator

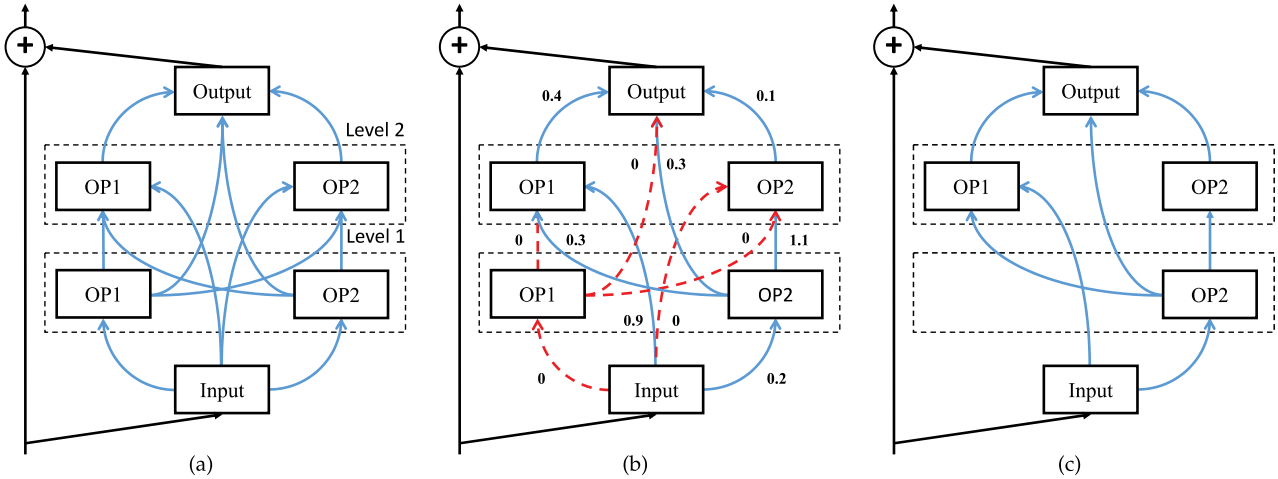


Fig. 5. An example of search block, which has two levels with two operations: (a) We start with a completely connected block in which all connections, and all nodes are kept. (b) In the search process, we jointly optimize the weights of neural network and the structure parameters $\boldsymbol{\lambda}$ (shown by the value on every connection) associated with each edge. (c) The final model after removing useless connections and operations.

$\mathcal{S}_{\alpha}(\mathbf{z})_i = \text{sign}(z_i)(|z_i| - \alpha)_+$ as $\mathcal{S}_{\eta_{(t)}\gamma}$. The original formulation of APG are

$$\mathbf{d}_{(t)} = \boldsymbol{\lambda}_{(t-1)} + \frac{t-2}{t+1}(\boldsymbol{\lambda}_{(t-1)} - \boldsymbol{\lambda}_{(t-2)}) \tag{10}$$

$$\mathbf{z}_{(t)} = \mathbf{d}_{(t)} - \eta_{(t)}\nabla\mathcal{G}(\mathbf{d}_{(t)}) \tag{11}$$

$$\boldsymbol{\lambda}_{(t)} = \mathbf{prox}_{\eta_{(t)}\mathcal{R}_s}(\mathbf{z}_{(t)}), \tag{12}$$

where $\eta_{(t)}$ is the gradient step size at iteration t and $\mathbf{prox}_{\eta_{(t)}\mathcal{R}_s}(\cdot) = \mathcal{S}_{\eta_{(t)}\gamma}(\cdot)$ since the regularization for $\boldsymbol{\lambda}$ is $\gamma\|\boldsymbol{\lambda}\|_1$. However, this formulation is not friendly for deep learning since we need to do extra forward-backward computation for obtaining $\nabla\mathcal{G}(\mathbf{d}_{(t)})$. Following the modification of NAG proposed in [32], [57] reformulated APG as a momentum based method to avoid redundant forward and backward by replacing the update of $\boldsymbol{\lambda}_{(t-1)}$ with

$$\boldsymbol{\lambda}'_{(t-1)} = \boldsymbol{\lambda}_{(t-1)} + \mu_{(t-1)}\mathbf{v}_{(t-1)}, \tag{13}$$

where $\mathbf{v}_{(t-1)} = \boldsymbol{\lambda}_{(t-1)} - \boldsymbol{\lambda}_{(t-2)}$ and $\mu_{(t-1)} = \frac{t-2}{t+1}$. The final original APG can be reformulated as

$$\mathbf{z}_{(t)} = \boldsymbol{\lambda}'_{(t-1)} - \eta_{(t)}\nabla\mathcal{G}(\boldsymbol{\lambda}'_{(t-1)}) \tag{14}$$

$$\mathbf{v}_{(t)} = S_{\eta_{(t)}\gamma}(\mathbf{z}_{(t)}) - \boldsymbol{\lambda}'_{(t-1)} + \mu_{(t-1)}\mathbf{v}_{(t-1)} \tag{15}$$

$$\boldsymbol{\lambda}'_{(t)} = \mathcal{S}_{\eta_{(t)}\gamma}(\mathbf{z}_{(t)}) + \mu_{(t)}\mathbf{v}_{(t)}. \tag{16}$$

In optimization process, all $\boldsymbol{\lambda}$ are initialized to 1, the weights $\mathbf{W}$ and structure parameters $\boldsymbol{\lambda}$ are updated using NAG [58] and APG-NAG jointly on the same training set. However, APG-NAG cannot be directly to solve the objective function Eq. (9) since the architecture parameters are optimized based on the validation set, training them jointly with weights in the same dataset will lead to overfitting just shown by the experiments in Section 4.5.3. Different from pruning, whose search space is usually quite limited, the search space in NAS is much more diverse. If the structure is learned on such an overfitting model, it will generalize badly on the test set. Besides, optimizing $\boldsymbol{\lambda}$ directly is costly as it is expensive to obtain the optimal network weights $\mathbf{W}^*_{\boldsymbol{\lambda}}$ for every step of $\boldsymbol{\lambda}$.

To solve these problems, we divide training data into two parts and update $\mathbf{W}$ and $\boldsymbol{\lambda}$ in these two separate sets. As $\mathbf{W}$ is shared for all child networks, we approximate $\mathbf{W}^*_{\boldsymbol{\lambda}}$ under the architecture $\boldsymbol{\lambda}$ by optimizing $\mathbf{W}_{\boldsymbol{\lambda}}$ for one step on the data set for weights and update $\boldsymbol{\lambda}$ for one step on the data set based on $\mathbf{W}_{\boldsymbol{\lambda}}$, iteratively. To help $\mathbf{W}_{\boldsymbol{\lambda}}$ approach $\mathbf{W}^*_{\boldsymbol{\lambda}}$ in the beginning of search process, we train the completely connected network for several epochs on the dataset for $\mathbf{W}$ to get a good initialization of weights. The importance of pretraining the completely connected network is also proven of great importance in Section 4.5. The pipeline of our method is shown in Algorithm 1.

Our method guarantees that the structure parameters $\boldsymbol{\lambda}$ are learned on a different subset of training data which is not seen during the learning of $\mathbf{W}_{\boldsymbol{\lambda}}$. Therefore, the sample distribution in the structure learning is more similar to that in testing, which may lead to better performance. Besides, the sparse optimization problem of $\boldsymbol{\lambda}$ is solved in an efficient APG method, which can be easily incorporated with deep learning framework.

Algorithm 1. Direct Sparse Optimization Neural Architecture Search

Require: Training data D
1: Split training data D into two parts D_w and D_λ;
2: Initialize $\mathbf{W}_{\boldsymbol{\lambda}}$ and $\boldsymbol{\lambda}$;
3: Training $\mathbf{W}_{\boldsymbol{\lambda}}$ for several epochs on D_w;
4: **repeat**
5: Update $\boldsymbol{\lambda}^t$ by optimizing loss on dataset D_λ, $\mathcal{L}_{D_\lambda}(\mathbf{W}_{\boldsymbol{\lambda}^{t-1}}, \boldsymbol{\lambda}^{t-1})$ based on Eqs. (14), (15), and Eq. (16);
6: Update $\mathbf{W}_{\boldsymbol{\lambda}^t}$ by optimizing loss on dataset D_w, $\mathcal{L}_{D_w}(\mathbf{W}_{\boldsymbol{\lambda}^t}, \boldsymbol{\lambda}^{t-1})$ with NAG;
7: **until** converged

3.4 Incorporating Different Budgets

Hand-crafted network usually incorporates many domain knowledge. For example, as highlighted in [54], memory access may be the bottleneck for lightweight network on GPU because of the use of separable convolution. Our method can easily consider these priors in the search by adaptively adjusting γ for each connection.

The first example is to balance the FLOPs for each block. As indicated in [59], most intense changes of the main branch flow of ResNet are concentrated after reduction block. In our experiments, we empirically find that the complexity of the block after each reduction block is much higher than the others' if all γ are fixed. Consequently, to balance the FLOPs among different blocks, the regularization weight $\boldsymbol{\lambda}$ is adjusted for every block, namely γ^t at iteration t adaptively according to the FLOPs of the block.

$$\gamma^t = \frac{\text{FLOPs}^t}{\text{FLOPs}_{block}}\gamma, \tag{17}$$

where FLOPs_{block} represents the FLOPs of the completely connected block and FLOPs^t, which can be calculated based on $\boldsymbol{\lambda}$, represents the FLOPs of the kept operations at iteration t for a block. Using this simple strategy, the distribution of FLOPs is smoothed by penalizing $\boldsymbol{\lambda}$ of every block according to the FLOPs of the block. This method is called *Adaptive FLOPs* in the following.

The second example is to incorporate a specific computation budget such as Memory Access Cost (MAC). Similarly, the γ applied on the nth operation in mth level at iteration t is calculated according to the following equation:

$$\gamma^t_{(m,n)} = \frac{\text{MAC}^t_{(m,n)}}{\text{MAC}_{max}}\gamma, \tag{18}$$

where $\text{MAC}^t_{(m,n)}$ represents the MAC of the nth operation in the mth level, and MAC_{max} represents the maximum MAC in the network. Using this strategy, DSO-NAS can generate architectures with better performance under the same budget of MAC. We call this method *Adaptive MAC* in the following.

Besides the FLOPs and MACs, latency is another important criterion for neural network. Unfortunately, unlike FLOPs or MACs that can be calculated, the latency of connection is hard to measure directly. In this work, we assume the expected latency of a network is the sum of every connection's latency. The latency of specific connection can be defined by the latency deduction of the whole network when this connection is removed. The regularization weights applied on every connection are adjusted every epoch based on its effect to the latency. Specifically, The γ applied on the nth operation in the mth level at epoch t is calculated by

$$\gamma^t_{(m,n,m',n')} = \frac{\mathrm{LAT}^t_{(m,n,m',n')} - \mathrm{LAT}^t_{min}}{\mathrm{LAT}^t_{max} - \mathrm{LAT}^t_{min}}\gamma + \gamma_0, \tag{19}$$

where γ_0 represents basic regularization weight, $\mathrm{LAT}^t_{(m,n,m',n')}$ represents the latency of network when the connection between the nth operation in mth level and the n'th operation in m'th level is pruned. LAT^t_{max} and LAT^t_{min} represent the maximum and minimum of $\mathrm{LAT}^t_{(i,j,i',j')}, i, i' \in [0, M], j, j' \in [0, N]$ respectively. We call this method *Adaptive Latency* in the following. We believe other types of budgets can be similarity incorporated.

4 Experiments

In this section, we first introduce the implementation details of our method, then followed by the results of classification tasks on CIFAR-10 [60] and ImageNet datasets [61] and segmentation task on PASCAL VOC datasets [62]. Later, we conduct experiments to analyze the effectiveness of budget aware techniques. At last, we analyze each design component of our method in detail.

4.1 Implementation Details

The pipeline of our method consists of three stages:

1) Training the completely connected network for several epochs to get a good initialization.
2) Searching architecture from the pre-trained model.
3) Re-training the final architecture from scratch with all training data.

In the first two stages, the scaling parameters in batch normalization layers are fixed to one to prevent affecting the learning of λ. After step two, we adjust the number of filters in each operation by a global width multiplier to satisfy the computation budget, and then train the network from scratch as done in [12].

The model searched with and without block structure sharing are denoted as *DSO-NAS-share* and *DSO-NAS-full*, respectively. In each block, we set the number of levels $M = 4$, the number of operations $N = 4$ as four kinds of operations are applied for all experiments indicated in Section 3.1. For the hyper-parameters of optimization algorithm and weight initialization, we follow the setting of [32]. All the experiments are conducted in MXNet [66] with NVIDIA GTX 1080Ti GPUs. As the fully connected supernet is extremely complicated and large in size, a GPU memory-saving technique [67] is applied to guarantee that the whole network can be accommodated by the GPUs. As for latency measurement, the CPU latency is measured on a server with Intel i5-6600K CPU. For the GPU platforms, the latency is measured on a single NVIDIA GeForce GTX 1080Ti with MXNet + CUDNN7.0. The input image size is 224 $\times$ 224 and batch size is set to 1 if not stated particularly. We report the latency by averaging 5 runs.

4.2 CIFAR

The CIFAR-10 dataset consists of 50K training images and 10K testing images. We divide the training data into two parts: 25K for training of weights and rest 25K for structure. During training, standard data pre-processing and augmentation techniques [68] including subtracting the channel mean and dividing the channel standard deviation, centrally padding the training images to 40$\times$40 and randomly cropping them back to 32$\times$32 are adopted. The mini-batch size is 128 on 2 GPUs and weight decay is set to 3e-4. First, we pre-train the full network for 120 epochs, then search network architecture until convergence (about 120 epochs). Constant learning rate 0.1 is applied in both pre-training and search stages. The network adopted in CIFAR-10 experiments consists of three stages, and each stage has eight convolution blocks and one reduction block. Adaptive FLOPS (introduced in Section 3.4) is applied during the search process. The GPU memory cost in these two stages is reduced to 2.3G with the help of [67]. The pre-training and search stage costs about 12 and 10 hours respectively. After search, we train the final model from scratch with the same setting of [12]. The searched models are trained for 630 epochs with NAG, where we change the learning rate following the cosine scheduler [69] with $l_{max} = 0.05, l_{min} = 0.001, T_0 = 10, T_{mul} = 2$. Additional improvements are applied including dropout technique [70] with probability 0.6, cutout [71] with size 16, drop path [72] with probability 0.5, auxiliary towers [73] located at the end of second stage with weight 0.4.

Table 1 shows the performance of our searched models, including DSO-NAS-full and DSO-NAS-share, where results with * are obtained by retraining model with our training hyperparameters. Adaptive FLOPS technique introduced in Section 3.4 is applied in DSO-NAS-full search space. We report the mean and standard deviation of five independent runs. Due to limited space, we only show the block structure of DSO-NAS-share in Fig. 6a. We also compare the simplest yet still effective baseline–the random structure and the full network that keeps all operations and connections. As the full network is extremely large in parameters, we rescale the width of the full network for fair comparison. Both our DSO-NAS-share and DSO-NAS-full yield much better performance with fewer parameters. Comparing with other state-of-the-art methods, our method demonstrates competitive results with similar or fewer parameters while costing only one GPU day.

4.3 ILSVRC 2012

To demonstrate the searching efficiency of our method, we apply our method on a larger and more complex task, the ImageNet ILSVRC 2012 classification task. The dataset consists of 1.28M images for training and 50K images for validation. In our ILSVRC 2012 experiments, we conduct data

TABLE 1
Comparison With State-of-the-Art NAS Methods on CIFAR-10

Architecture	Test Error	Parameter	Search Cost (GPU days)	Number of operations	Search Method	NAS Pipeline Completeness
ResNet(pre-activation) [63]	4.62	10.2M	-	-	manual	-
DenseNet [64]	3.46	25.6M	-	-	manual	-
NAS v1 [10]	5.50	4.2M	22400	-	RL	complete
NAS v2 [10]	6.01	2.5M	22400	-	RL	complete
NAS v3 [10]	4.47	7.1M	22400	-	RL	complete
NASNet-A [11]+cutout	2.65	3.3M	1800	13	RL	complete
NASNet-B [11]	3.73	2.6M	1800	13	RL	complete
AmoebaNet-A [9]	3.34	3.2M	3150	19	evolution	complete
AmoebaNet-B [9]+cutout	2.55	2.8M	3150	19	evolution	complete
PNAS [16]	3.41	3.2M	150	8	SMBO	complete
ENAS [12]+cutout	2.89	4.6M	0.5	5	RL	complete
Hierarchical Evo [65]	3.75	15.7M	300	6	evolution	complete
DARTS(1st order) [14]+cutout	2.94	2.9M	1.5	7	gradient-based	incomplete
DARTS(2nd order) [14]+cutout	2.83	3.4M	4	7	gradient-based	incomplete
SNAS(mild constraint) [52]+cutout	2.98	2.9M	1.5	7	gradient-based	complete
SNAS(moderate constraint) [52]+cutout	2.85 ± 0.02	2.8M	1.5	7	gradient-based	complete
SNAS(aggressive constraint) [52]+cutout	3.10 ± 0.04	2.3M	1.5	7	gradient-based	complete
GDAS [50]+cutout*	2.96	3.4M	0.2	7	gradient-based	complete
DARTS+ [49]+cutout*	2.67	3.7M	0.5	7	gradient-based	incomplete
DSO-NAS-share+cutout	2.74 ± 0.07	3.0M	1	4	gradient-based	complete
DSO-NAS-full+cutout	2.83 ± 0.12	3.0M	1	4	gradient-based	complete
random-share+cutout	3.48 ± 0.21	3.4 ± 0.1M	-	4	-	-
random-full+cutout	3.46 ± 0.19	3.5 ± 0.1M	-	4	-	-
full network+cutout	3.32 ± 0.05	3.0M	-	4	-	-

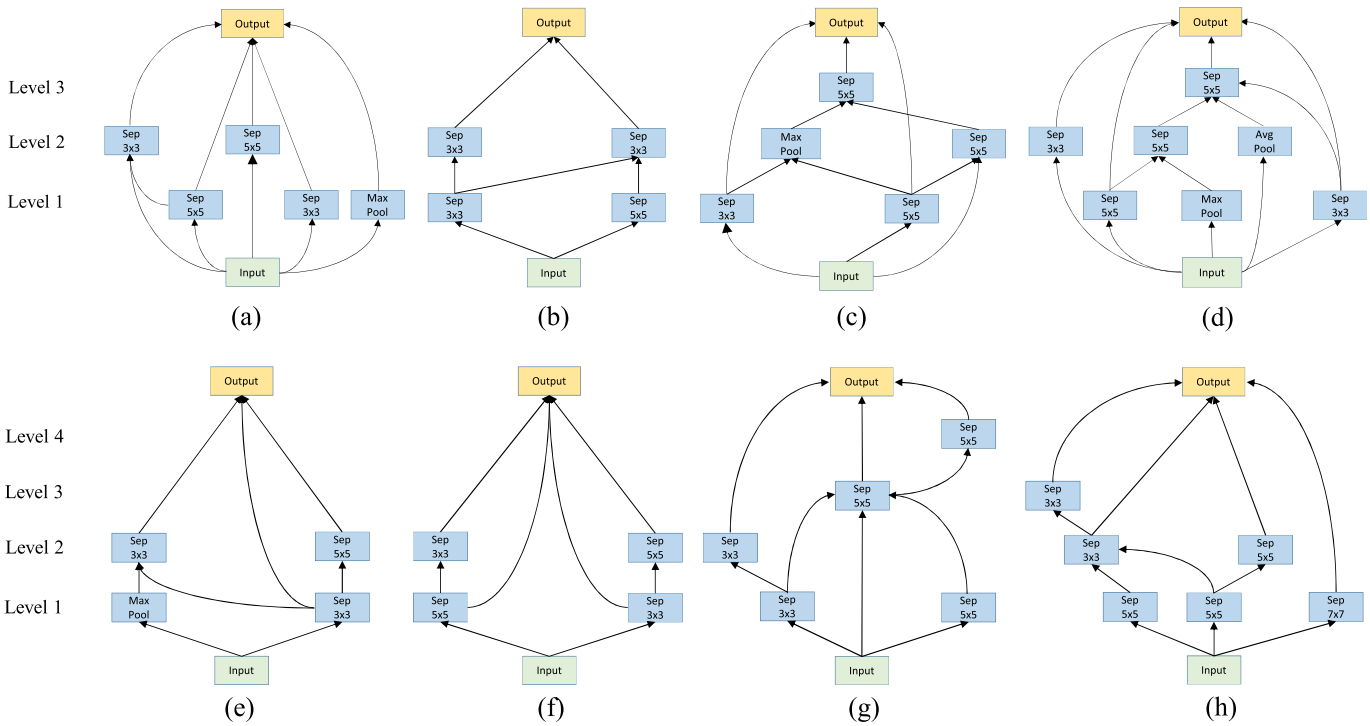


Fig. 6. Block structures learned on different datasets. (a) The block learned on CIFAR-10. (b) The block learned on ImageNet with the Adaptive Latency technique. (c) The block learned on COCO. (d) The block learned on ImageNet without the Adaptive Latency technique. (e) The block learned on CIFAR-10 with the Adaptive MAC technique. (f) The block learned on CIFAR-10 with fewer number of levels ($M = 2$). (g) The block learned on CIFAR-10 with more number of levels ($M = 5$). (h) The block learned on CIFAR-10 with more number of operations ($N = 5$).

augmentation based on the publicly available implementation of 'fb.resnet'.[1] Since this dataset is much larger than CIFAR-10, the training dataset is divided into two parts: $4/5$ for training weights and $1/5$ for training structure. In the pre-training stage, we train the whole network for 30 epochs with constant learning rate 0.1, weight decay 4×10^{-5}. The pre-training stage takes about 18 hours with mini-batch size

1. https://github.com/facebook/fb.resnet.torch

TABLE 2
Comparison With State-of-the-Art Image Classifiers on ImageNet

Architecture	Top-1/5 Error rate	Para	FLOPs	Search Cost (GPU days)
Inception-v1 [2]	30.2 /10.1	6.6M	1448M	-
MobileNet [75]	29.4 /10.5	4.2M	569M	-
ShuffleNet-v1 2x [76]	26.3 /10.2	5M	524M	-
ShuffleNet-v2 2x [54]	25.1 /-	5M	591M	-
NASNet-A† [11]	26.0 / 8.4	5.3M	564M	1800
NASNet-B† [11]	27.2 / 8.7	5.3M	488M	1800
NASNet-C† [11]	27.5 / 9.0	4.9M	558M	1800
AmoebaNet-A† [9]	25.5 / 8.0	5.1M	555M	3150
AmoebaNet-B† [9]	26.0 / 8.5	5.3M	555M	3150
AmoebaNet-C† [9]	24.3 / 7.6	6.4M	570M	3150
PNAS† [16]	25.8 / 8.1	5.1M	588M	150
DARTS† [14]	26.9 / 9.0	4.9M	595M	4
OSNAS [44]	25.8 /-	5.1M	-	-
MNAS-92 [45]	25.2 / 8.0	4.4M	388M	1600
GDAS† [50]	26.0 / 8.5	5.3M	581M	0.2
DARTS+ [49]*	25.3 / 8.1	5.1M	582M	6.8
DSO-NAS†	26.2 / 8.6	4.7M	571M	1
DSO-NAS-full	25.7 / 8.1	4.6M	608M	6
DSO-NAS-share	25.4 / 8.3	4.7M	567M	6
random-full	26.6 / 9.8	4.3M	583M	-
random-share	26.5 / 9.7	4.7M	564M	-

256 on 8 GPUs. The same setting is adopted in the search stage, which costs about 0.75 days with 8 GPUs. We apply the Adaptive Latency technique introduced in Section 3.4 for the share search scope in the search process. The GPU memory cost of these two stages is around 3.2G.

After the search, we train the final model from scratch using NAG for 240 epochs, with batch size 1,024 on 8 GPUs. We set weight decay to 4×10^{-5} and adopt linear-decay learning rate schedule (linearly decreased from 0.5 to 0). Additional techniques including Label smoothing [74] and auxiliary loss [73] are utilized in the training process. There are four stages in the network for ImageNet, and the number of convolution blocks in these four stages is set to 2, 2, 13, 6, respectively, according to the distribution of blocks [3].

We first transfer the block structure searched on CIFAR-10. We also directly search the network architecture on ImageNet. The final structure generated by DSO-NAS-share is shown in Fig. 6b. The quantitative results for ImageNet are shown in Table 2, where results with † are obtained by transferring the generated CIFAR-10 blocks to ImageNet and results with * are obtained with our training hyperparameters.

It is notable that given similar computation budget, DSO-NAS achieves competitive or better performance than other state-of-the-art methods with less search cost, except for MnasNet [45] whose search space is carefully designed and different from other methods. The block structure transferred from CIFAR-10 also achieves impressive performance, proving the generalization capability of the searched architecture. Moreover, directly searching on the target dataset (ImageNet) brings additional improvements. Comparing to random search, our method yields better results.

As shown by the studies [11], [65], the designing of search space plays an important role in neural architecture search. Benefiting from the development of manually designed architecture like Mobilnet series [46], [75], many novel search spaces composed of resource-efficiency structures are proposed [45], [53] to further reduce the cost of computation. For fair comparison and proving the transferability of our method, we incorporate DSO-NAS with the search space under mobile setting.

TABLE 3
Results of Transforming Our Method to Different Search Space

Method	Top-1/5 Error rate	Latency / ms GPU / CPU	Search Cost (GPU hours)
MobileNet [75]	29.4 /10.5	5.4 / 11.7	-
MobileNet v2 [46]	28.0 / 9.0	6.2 / 20.5	-
Proxyless(GPU) [53]*	25.4 / 8.4	6.2 / 15.3	200
Proxyless(CPU) [53]*	25.1 / 8.1	7.3 / 21.7	200
DSO-NAS(GPU)	25.2 / 8.2	6.3 / 15.5	168
DSO-NAS(CPU)	24.8 / 8.0	7.1 / 21.9	168

The searching and training process is the same as the experiments in Section 4.3, we also take latency into consideration. We measure the latency of every operation in the search space and apply regularization weight on every connection according to the latency of the corresponding operation. The λ applied on every connection is calculated by Eq. (19) introduced in Section 5. We keep training until there is only one operation in every block. During testing, the batch size is set to 8 for the measure of GPU latency and 1 for the measure of CPU latency. The result is shown in Table 3, where results of the models with * are obtained with our training hyperparameters. Table 3 shows the models searched by our method achieve better performance while keeping a similar latency, besides, our method is more efficient in search cost. Our experiments show that the designing of search space is extremely important for neural architecture search and our method can be easily transformed to different search spaces. The searched architectures are shown in Fig. 7.

4.4 Pascal VOC

Besides image classification task, we also study neural architecture search for semantic segmentation. In semantic segmentation, backbones play an important role. The performance of segmentation is highly correlated with the representational ability of features extracted by backbones. However, many semantic segmentation networks are usually equipped with backbone network designed for image classification. It may be sub-optimal as the classification task mainly requires features contain information about *what* the main object the image is while segmentation task requires information about *what and where* each object is, demonstrated by [77], [78], [79]. Therefore, we directly conduct architecture search experiments on semantic segmentation task to further improve the performance of segmentation network.

In the search process, we apply the head adopted in Deeplab v3 [80] and search for the architecture of feature extractor with block sharing. We select the 20 semantic class contained by Pascal VOC and divide the training dataset of COCO [81] into two parts, $4/5$ for training weights and rest for structure. In the pre-training stage, the whole feature

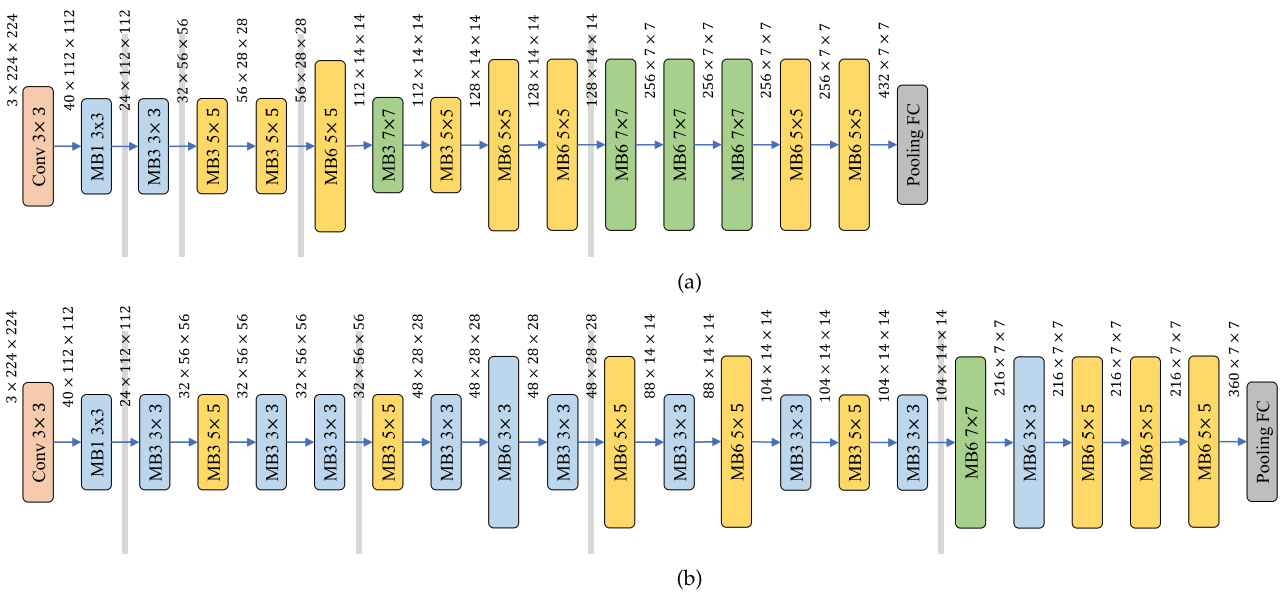


Fig. 7. Network structure learned on the search space proposed by Proxyless NAS. (a) Efficient GPU architecture obtained by DSO-NAS. (b) Efficient CPU architecture obtained by DSO-NAS.

extractor is trained for 30 epochs using NAG with learning rate fixed to 0.1 on ImageNet. In the search stage, we apply the same network structure used in Section 4.3 and adopt APG-NAG to search architecture with fixed learning rate 0.014 for 50 epochs, the batch size is set to 80. The search stage costs about 3 days with 8 GPUs while the pre-training takes about 2 days. The GPU memory cost of the pre-training and search stage is 3.2G and 4.3G respectively.

After search, we train the network from scratch. The searched architecture is pre-trained for 120 epochs on ImageNet and then trained on the semantic segmentation task using SGD with initiate learning rate 0.028 and weight decay 4e-5. Following [80], the training setup can be divided into three stages, in the first stage the architecture is trained for 100 epochs (on COCO+BSD+VOC) with learning rate 0.014 followed by the second stage - for 160 (BSD+VOC) with learning rate 0.008, and the last one - for 350 (VOC only) with learning rate 0.008. Other hyperparameters are set following [80]. Table 4 shows the performance of our model on PASCAL VOC dataset [62] where DSO-NAS-seg represents the architecture obtained in the segmentation task and DSO-NAS-cls represents the architecture searched on the Imagenet dataset with block sharing.

All above models employ the ratio of input image spatial resolution to final output resolution $OS = 16$ and have been pre-trained on COCO following [46]. It's notable that DSO-NAS achieves similar performance with fewer parameters compared with MobileNet. Besides, the architecture searched directly on segmentation dataset has better performance than the one searched on classification task, which shows DSO-NAS can perform task-specific architecture search.

TABLE 4
Comparison With State-of-the-Art Architecture on PASCAL VOC

Architecture	mIOU	Para	Search Cost (GPU days)
Mobilenet v1 [46]	75.29	11.2M	-
Mobilenet v2 [46]	75.70	4.5M	-
DSO-NAS-seg	76.40	6.7M	24
DSO-NAS-cls	76.00	6.5M	24

4.5 Ablation Study

In this section, we conduct ablation experiments to verify the importance of different components proposed in our method.

4.5.1 Effectiveness of Budget Aware Technique

With Adaptive FLOPS and Adaptive MAC techniques, the weight of sparse regularization will be changed adaptively according to Eqs. (17) and (18). We also show the error rates of different settings in Fig. 8. It is clear that the networks searched with adaptive FLOPs technique are consistently better under the same computational constraints.

DSO-NAS can also search for architecture based on other computational targets, such as latency described in Section 3.4. We compare the latency of several state-of-the-art NAS methods, including DARTS, NasNet and our DSO-NAS in different hardware architectures and platforms. Table 5 shows the results of network search with the *Adaptive Latency* technique in the shared search space, where 'DSO-NAS-share+AL' represents applying Adaptive Latency technique in the search process. In the experiments, weight of sparse regularization for each connection is calculated based on its influence on latency according to Eq. (19). The latency of each operation is measured on GPU. During testing, the batch-size is set to 1. The block structures learned with and without adaptive latency technique are shown in Figs. 6b and 6d respectively. Obviously, DSO-NAS can generate architecture with similar accuracy with less latency, proving the effectiveness of our proposed Adaptive Latency technique.

4.5.2 Influence of Pretraining the Supernet

To explore the influence of pre-training the completely connected network, we train the completely connected network for different epochs to get different initialization and search

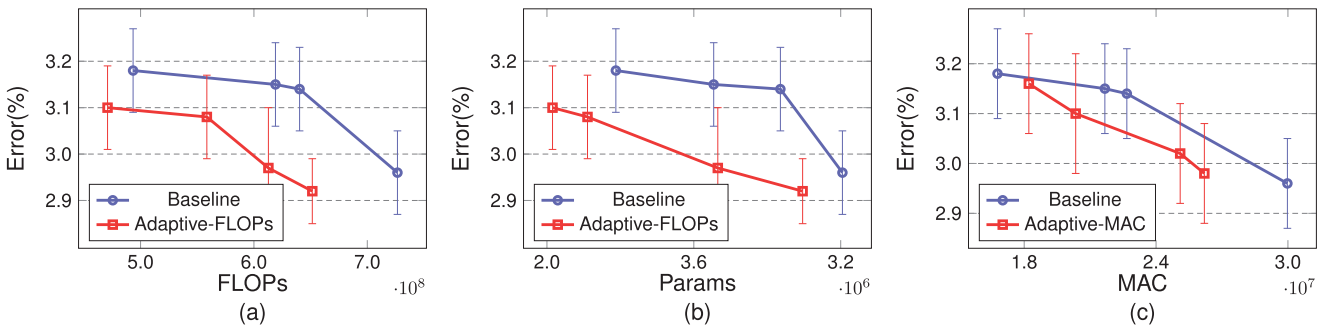


Fig. 8. Performance of adaptive FLOPs and adaptive MAC techniques. (a) Error-FLOPs curve w/o adaptive FLOPs technique. (b) Error-Parameters curve w/o adaptive FLOPs technique. (c) Error-MAC curve w/o adaptive MAC technique.

the network architecture based on these initializations. For fair comparison, we fix other hyperparameters and conduct experiments in both full search scope and share search scope introduced in Section 4.2. The results are shown in Table 6. Note that we only pre-train model on the weight learning set.

It's notable that pre-training the completely connected network plays an important role in search process and higher accuracy of the pre-trained model lead to better performance of searched architecture. Besides, a good initialization of weights may improve the performance by 0.2 percent compared with random initialization. Our experiments indicate a good initialization is of extreme importance as a good initialization can help the weights approach $\mathbf{W}^*_{\boldsymbol{\lambda}}$ from the beginning of search process and $\mathbf{W}^*_{\boldsymbol{\lambda}}$ is essential in optimizing the structure parameters indicated by Eq. (7).

4.5.3 Effectiveness of Split Training

To better understand the necessity of training weights and structure parameters in two different sets. We conduct experiments to explore the influence of splitting data set into two parts and the ratio of them. The results are shown in Table 7, where "Split training" means whether to split the whole training set into two sets for weight and structure learning separately. The "Ratio of W&S" means the ratio of training samples for weight learning and structure learning. As for the ratio of $x : y$, we update weights for x times and update $\boldsymbol{\lambda}$ for y times for every $x + y$ iterations.

It is notable that the use of a separate set for structure learning plays an important role to prevent overfitting training data, and optimizing weights and structure parameters jointly over the same data set may lead to 0.2 percent drop on accuracy under that same parameter budgets. Besides, the ratio of these two sets has minor influence. The result can be explained from two aspects: theoretically, in the task of neural architecture search, the structure parameters and weights are optimized on different datasets, just as shown by Eq. (9), training them jointly in one set will lead to inferior results. Practically, structure parameters can be viewed as a type of hyperparameter in a sense, optimizing these hyperparameters in the same set as weights may lead to overfitting on the training data and poor generalization.

4.5.4 Sensitivity to the Scale of Search Space

In order to empirically analyze the sensitivity to the scale of search space, we conduct experiments on search spaces with different numbers of levels M and operations N on the CIFAR-10 dataset. First, we fix the number of operations N to 4 and vary the number of levels M, then we fix the number of levels M to 4 and vary the number of the operations from 2 to 5. In the experiments with different operations, we adopt the following operations: separable convolution with kernel 3×3, average

TABLE 5
Performance of Adaptive Latency Techniques

Method	Top-1/5 Error rate	Flops	Latency/ms (CPU/GPU)
MobileNet	29.4 /10.5	569M	28.21/1.87
DARTS	26.9 / 9.0	595M	17.52/7.69
NASNet-A	26.0 / 8.4	564M	35.54/10.45
DSO-Nas-share	25.4 / 8.4	586M	36.19/10.34
DSO-Nas-share+AL	25.4 / 8.3	567M	15.93/6.88

TABLE 6
Influence of Different Pretrain Strategy on CIFAR-10 Dataset

Pre-train accuracy	Pre-train accuracy	Search space	Parameters	Test Error
0	0.0	Full Share	2.9M 3.0M	3.26 ± 0.08 3.22 ± 0.09
60	85.77	Full Share	3.0M 3.0M	3.08 ± 0.10 3.02 ± 0.11
120	88.63	Full Share	2.9M 3.0M	3.05 ± 0.09 2.98 ± 0.10
150	88.75	Full Share	3.0M 3.0M	3.06 ± 0.08 2.94 ± 0.12

TABLE 7
Influence of Different Split Training Strategy on CIFAR-10 Dataset

Split training	Ratio of W&S	Search space	Parameters	Test Error
No	-	Full Share	2.9M 3.0M	3.26 ± 0.08 3.20 ± 0.09
Yes	1:1	Full Share	2.9M 3.0M	3.05 ± 0.09 2.98 ± 0.10
Yes	3:1	Full Share	3.0M 3.0M	3.04 ± 0.07 2.99 ± 0.11
Yes	4:1	Full Share	2.9M 3.0M	3.06 ± 0.08 2.96 ± 0.09

TABLE 8
Influence of Number of Levels M on ImageNet

Number of levels M	Top-1/5 Error rate	Para	FLOPs	Search Cost (GPU hours)
2	26.3 / 8.8	4.9M	591M	23
3	26.2 / 8.7	4.9M	592M	24
4	26.2 / 8.6	4.7M	571M	26
5	**25.8** / **8.4**	4.8M	583M	30

TABLE 9
Influence of Number of Operations N on ImageNet

Number of operations N	Top-1/5 Error rate	Para	FLOPs	Search Cost (GPU hours)
2	26.3 / 8.6	4.9M	581M	18
3	26.3 / 8.5	4.8M	581M	21
4	26.2 / 8.6	4.7M	571M	26
5	**26.0** / **8.5**	4.6M	567M	32

pooling with kernel 3×3, max pooling with kernel 3×3, separable convolution with kernel 5×5 and separable convolution with kernel 7×7. Every level contains the first n operations in the search space.

The obtained block structures with $M = 2$, $M = 5$ and $N = 5$ are shown in Figs. 6f, 6g, and 6h respectively. Since the performance of these structures in CIFAR-10 is almost saturated, we transfer the obtained structures to the ImageNet dataset to validate the effectiveness of these structures. The results are shown in Tables 8 and 9, respectively. It can be noted that our method tends to achieve better performance with larger search space, verifying the scalable ability of DSO-NAS. Furthermore, our experiments also verify one of our insight that a broader and more flexible search space is beneficial for obtaining suitable architecture.

5 Conclusion and Future Work

Neural Architecture Search has been the core technology for realizing AutoML. In this paper, we have proposed a Direct Sparse Optimization method for NAS. Our method is appealing to both academic research and industrial practice in two aspects: First, our unified weight and structure learning method is fully differentiable in contrast to most previous works. It provides a novel model pruning view to the NAS problem. Second, the induced optimization method is both efficient and effective. We have demonstrated state-of-the-art performance on both CIFAR and ILSVRC2012 image classification datasets, with affordable cost (single machine in one day).

In the future, we would like to incorporate DSO-NAS with other tasks like detection, translation, etc. Since our experiments spotlight the different distribution of searched architectures between tasks, the exploration of the searched architecture pattern might provide meaningful insight on the interpretability of neural network. We believe DSO-NAS opens a new direction to pursue such objective. It could push a further step to realizing AutoML for everyone's use.

Acknowledgments

This work was supported by the Major Project for New Generation of AI under Grant No. 2018AAA0100400, and the National Natural Science Foundation of China under Grants 91646207 and 61976208.

References

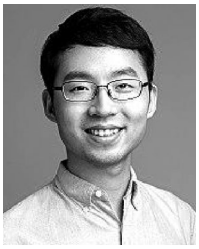
[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," in *Proc. 25th Int. Conf. Neural Inf. Process. Syst.*, 2012, pp. 1106–1114.
[2] C. Szegedy *et al.* "Going deeper with convolutions," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2015, pp. 1–9.
[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 770–778.
[4] G. Hinton *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Process. Mag.*, vol. 29, no. 6, pp. 82–97, Nov. 2012.
[5] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," in *Proc. Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 6645–6649.
[6] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
[7] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.
[8] E. Real *et al.*, "Large-scale evolution of image classifiers," in *Proc. 34th Int. Conf. Mach. Learn.*, 2017, pp. 2902–2911.
[9] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4780–4789.
[10] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2017.
[11] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 8697–8710.
[12] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 4092–4101.
[13] Z. Zhong, J. Yan, W. Wu, J. Shao, and C.-L. Liu, "Practical block-wise neural network architecture generation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 2423–2432.
[14] H. Liu, K. Simonyan, and Y. Yang, "DARTS: Differentiable architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.
[15] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Ann. Operations Res.*, vol. 153, no. 1, pp. 235–256, 2007.
[16] C. Liu *et al.*, "Progressive neural architecture search," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 19–35.
[17] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1990, pp. 598–605.
[18] B. Hassibi and D. G. Stork, "Second order derivatives for network pruning: Optimal brain surgeon," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 1993, pp. 164–171.
[19] Y. Guo, A. Yao, and Y. Chen, "Dynamic network surgery for efficient DNNs," in *Proc. Advances Neural Inf. Process. Syst.*, 2016, pp. 1379–1387.
[20] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," in *Proc. Int. Conf. Neural Inf. Process. Syst.*, 2015, pp. 1135–1143.
[21] F. H. Leung, H. Lam, S. Ling, and P. K. Tam, "Tuning of the structure and parameters of a neural network using an improved genetic algorithm," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 14, no. 1, pp. 79–88, Jan. 2003.
[22] J. Wang, C. Xu, X. Yang, and J. M. Zurada, "A novel pruning algorithm for smoothing feedforward neural networks based on group lasso method," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 5, pp. 2012–2024, May 2018.
[23] S. Chen and Q. Zhao, "Shallowing deep networks: Layer-wise pruning based on feature representations," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 12, pp. 3048–3056, Dec. 2019.
[24] F. Tung and G. Mori, "Deep neural network compression by in-parallel pruning-quantization," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 3, pp. 568–579, Mar. 2020.

[25] H. Hu, R. Peng, Y.-W. Tai, and C.-K. Tang, "Network trimming: A data-driven neuron pruning approach towards efficient deep architectures," 2016, *arXiv:1607.03250*.
[26] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," in *Proc. Int. Conf. Learn. Representations*, 2017.
[27] Z. Mariet and S. Sra, "Diversity networks," in *Proc. Int. Conf. Learn. Representations*, 2016.
[28] Z. Liu, J. Li, Z. Shen, G. Huang, S. Yan, and C. Zhang, "Learning efficient convolutional networks through network slimming," in *Proc. IEEE Int. Conf. Comput. Vis.*, 2017, pp. 2755–2763.
[29] J. Luo, H. Zhang, H. Zhou, C. Xie, J. Wu, and W. Lin, "ThiNet: Pruning CNN filters for a thinner net," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 41, no. 10, pp. 2525–2538, Oct. 2019.
[30] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li, "Learning structured sparsity in deep neural networks," in *Proc. 30th Int. Conf. Neural Inf. Process. Syst.*, 2016, pp. 2074–2082.
[31] M. Gong, J. Liu, H. Li, Q. Cai, and L. Su, "A multiobjective sparse feature learning model for deep neural networks," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 12, pp. 3263–3277, Dec. 2015.
[32] Z. Huang and N. Wang, "Data-driven sparse structure selection for deep neural networks," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 317–334.
[33] R. Yu *et al.*, "NISP: Pruning networks using neuron importance score propagation," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 9194–9203.
[34] A. Ashok, N. Rhinehart, F. Beainy, and K. M. Kitani, "N2N learning: Network to network compression via policy gradient reinforcement learning," in *Proc. Int. Conf. Learn. Representations*, 2018.
[35] Y. He, J. Lin, Z. Liu, H. Wang, L.-J. Li, and S. Han, "AMC: AutoML for model compression and acceleration on mobile devices," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 815–832.
[36] Z. Liu, M. Sun, T. Zhou, G. Huang, and T. Darrell, "Rethinking the value of network pruning," in *Proc. Int. Conf. Learn. Representations*, 2019.
[37] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *IEEE Trans. Neural Netw.*, vol. 5, no. 1, pp. 54–65, Jan. 1994.
[38] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002.
[39] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: From architectures to learning," *Evol. Intell.*, vol. 1, no. 1, pp. 47–62, 2008.
[40] L. Shao, L. Liu, and X. Li, "Feature learning for image classification via multiobjective genetic programming," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 7, pp. 1359–1371, Jul. 2014.
[41] T. Elsken, J. H. Metzen, and F. Hutter, "Efficient multi-objective neural architecture search via lamarckian evolution," in *Proc. Int. Conf. Learn. Representations*, 2019.
[42] J. Liu, M. Gong, Q. Miao, X. Wang, and H. Li, "Structure learning for deep neural networks based on multiobjective optimization," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 29, no. 6, pp. 2450–2463, Jun. 2018.
[43] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
[44] G. Bender, P.-J. Kindermans, B. Zoph, V. Vasudevan, and Q. Le, "Understanding and simplifying one-shot architecture search," in *Proc. 35th Int. Conf. Mach. Learn.*, 2018, pp. 549–558.
[45] M. Tan, B. Chen, R. Pang, V. Vasudevan, and Q. V. Le, "MnasNet: Platform-aware neural architecture search for mobile," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 2820–2828.
[46] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520.
[47] J. Ma, Z. Zhao, J. Chen, A. Li, L. Hong, and E. H. Chi, "SNR: Sub-network routing for flexible parameter sharing in multi-task learning," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 216–223.
[48] T. Véniat and L. Denoyer, "Learning time/memory-efficient deep architectures with budgeted super networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 3492–3500.
[49] H. Liang *et al.*, "DARTS+: Improved differentiable architecture search with early stopping," 2019, *arXiv: 1909.06035*.
[50] X. Dong and Y. Yang, "Searching for a robust neural architecture in four GPU hours," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 1761–1770.
[51] B. Wu *et al.*, "FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2019, pp. 10726–10734.
[52] S. Xie, H. Zheng, C. Liu, and L. Lin, "SNAS: Stochastic neural architecture search," in *Proc. Int. Conf. Learn. Representations*, 2019.
[53] H. Cai, L. Zhu, and S. Han, "ProxylessNAS: Direct neural architecture search on target task and hardware," in *Proc. Int. Conf. Learn. Representations*, 2019.
[54] N. Ma, X. Zhang, H.-T. Zheng, and J. Sun, "ShuffleNet V2: Practical guidelines for efficient CNN architecture design," in *Eur. Conf. Comput. Vis.*, 2018, pp. 122–138.
[55] J. Ye, X. Lu, Z. Lin, and J. Z. Wang, "Rethinking the smaller-norm-less-informative assumption in channel pruning of convolution layers," in *Proc. Int. Conf. Learn. Representations*, 2018.
[56] N. Parikh and S. P. Boyd, "Proximal algorithms," *Found. Trends Optim.*, vol. 1, no. 3, pp. 127–239, 2014.
[57] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu, "Advances in optimizing recurrent networks," in *Proc. IEEE Int. Conf. Acoust. Speech Signal Process.*, 2013, pp. 8624–8628.
[58] Y. E. Nesterov, "A method for solving the convex programming problem with convergence rate $\varnothing(1/k^2)$," *Dokl. Akad. Nauk*, vol. 269, no. 3, pp. 543–547, 1983.
[59] S. Jastrzebski, D. Arpit, N. Ballas, V. Verma, T. Che, and Y. Bengio, "Residual connections encourage iterative inference," in *Proc. Int. Conf. Learn. Representations*, 2018.
[60] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009. [Online]. Available: https://www.cs.toronto.edu/~kriz/cifar.html
[61] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *Int. J. Comput. Vis.*, vol. 115, no. 3, pp. 211–252, 2015.
[62] M. Everingham, S. M. A. Eslami, L. J. V. Gool, C. K. I. Williams, J. M. Winn, and A. Zisserman, "The pascal visual object classes challenge: A retrospective," *Int. J. Comput. Vis.*, vol. 111, no. 1, pp. 98–136, 2015.
[63] K. He, X. Zhang, S. Ren, and J. Sun, "Identity mappings in deep residual networks," in *Proc. Eur. Conf. Comput. Vis.*, 2016, pp. 630–645.
[64] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 2261–2269.
[65] H. Liu, K. Simonyan, O. Vinyals, C. Fernando, and K. Kavukcuoglu, "Hierarchical representations for efficient architecture search," in *Proc. Int. Conf. Learn. Representations*, 2018.
[66] T. Chen *et al.*, "MXNet: A flexible and efficient machine learning library for heterogeneous distributed systems," in *Proc. Int. Conf. Neural Inf. Process. Syst. Workshop*, 2015.
[67] T. Chen, B. Xu, C. Zhang, and C. Guestrin, "Training deep nets with sublinear memory cost," 2016, *arXiv:1604.06174*.
[68] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual formations for deep neural networks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2017, pp. 5987–5995.
[69] I. Loshchilov and F. Hutter, "SGDR: Stochastic gradient descent with warm restarts," in *Proc. Int. Conf. Learn. Representations*, 2017.
[70] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 1929–1958, 2014.
[71] T. DeVries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017, *arXiv: 1708.04552*.
[72] G. Larsson, M. Maire, and G. Shakhnarovich, "FractalNet: Ultra-deep neural networks without residuals," in *Proc. Int. Conf. Learn. Representations*, 2017.
[73] C.-Y. Lee, S. Xie, P. Gallagher, Z. Zhang, and Z. Tu, "Deeply-supervised nets," in *Proc. 18th Int. Conf. Artif. Intell. Statist.*, 2015, pp. 562–570.
[74] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception architecture for computer vision," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2016, pp. 2818–2826.
[75] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," 2017, *arXiv: 1704.04861*.
[76] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 6848–6856.
[77] K. Sun *et al.*, "High-resolution representations for labeling pixels and regions," 2019, *arXiv: 1904.04514*.

[78] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun, "DetNet: Design backbone for object detection," in *Proc. Eur. Conf. Comput. Vis.*, 2018, pp. 339–354.
[79] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 834–848, Apr. 2018.
[80] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017, *arXiv: 1706.05587*.
[81] T. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Proc. Eur. Conf. Comput. Vis.*, 2014, pp. 740–755.

Xinbang Zhang received the BS degree in automation from the University of Northeastern University, Shenyang, China, in 2017. He is currently working toward the PhD degree from the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include network pruning and Auto ML.

Zehao Huang received the BS degree in automatic control from Beihang University, Beijing, China, in 2015. He is currently an algorithm engineer at TuSimple. His research interests include computer vision and image processing.



Naiyan Wang received the BS degree from Zhejiang University, China, in 2011, and the PhD degree from CSE Department, Hong Kong University of Science and Technology, Hong Kong, in 2015. He is currently the chief scientist at TuSimple. His research interests focuses on applying statistical computational model to real problems in computer vision and data mining. Currently, he mainly works on the perception and localization part of autonomous truck. He has published more than 30 papers in top tier conferences in computer vision and machine learning, and been cited more than 4,600 times according to Google scholar.

Shiming Xiang received the BS degree in mathematics from Chongqing Normal University, Chongqing, China, in 1993, the MS degree from Chongqing University, Chongqing, China, in 1996, and the PhD degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2004. From 1996 to 2001, he was a lecturer with the Huazhong University of Science and Technology, Wuhan, China. He was a postdoctorate candidate with the Department of Automation, Tsinghua University, Beijing, China, until 2006. He is currently a professor at the National Lab of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China. His research interests include pattern recognition and machine learning.

Chunhong Pan received the BS degree in automatic control from Tsinghua University, Beijing, China, in 1987, the MS degree from the Shanghai Institute of Optics and Fine Mechanics, Chinese Academy of Sciences, China, in 1990, and the PhD degree in pattern recognition and intelligent system from the Institute of Automation, Chinese Academy of Sciences, Beijing, China, in 2000. He is currently a professor at the National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, China. His research interests include computer vision, image processing, computer graphics, and remote sensing.