

# CARS: Continuous Evolution for Efficient Neural Architecture Search

Zhaohui Yang<sup>1\*</sup>, Yunhe Wang<sup>2</sup>, Xinghao Chen<sup>2</sup>, Boxin Shi<sup>3,4</sup>,  
Chao Xu<sup>1</sup>, Chunjing Xu<sup>2</sup>, Qi Tian<sup>2†</sup>, Chang Xu<sup>5</sup>

<sup>1</sup> Key Laboratory of Machine Perception (Ministry of Education), Peking University.

<sup>2</sup> Huawei Noah's Ark Lab. <sup>3</sup> Peng Cheng Laboratory.

<sup>4</sup> National Engineering Laboratory for Video Technology, Peking University.

<sup>5</sup> School of Computer Science, University of Sydney.

{zhaohuiyang, shiboxin}@pku.edu.cn; xuchao@cis.pku.edu.cn

{yunhe.wang, xinghao.chen, tian.qil, xuchunjing}@huawei.com; c.xu@sydney.edu.cn

## Abstract

Searching techniques in most of existing neural architecture search (NAS) algorithms are mainly dominated by differentiable methods for the efficiency reason. In contrast, we develop an efficient continuous evolutionary approach for searching neural networks. Architectures in the population that share parameters within one supernet in the latest iteration will be tuned over the training dataset with a few epochs. The searching in the next evolution iteration will directly inherit both the supernet and the population, which accelerates the optimal network generation. The non-dominated sorting strategy is further applied to preserve only results on the Pareto front for accurately updating the supernet. Several neural networks with different model sizes and performance will be produced after the continuous search with only 0.4 GPU days. As a result, our framework provides a series of networks with the number of parameters ranging from 3.7M to 5.1M under mobile settings. These networks surpass those produced by the state-of-the-art methods on the benchmark ImageNet dataset.

## 1. Introduction

Convolution neural networks have made great progress in a large range of computer vision tasks, such as recognition [19, 31, 14], detection [10, 23], and segmentation [13]. Over-parameterized deep neural networks can produce impressive recognition performance but will consume huge computation resources at the same time. Besides, designing novel network architectures heavily relies on human experts' knowledge and experience, and may take many trials before achieving meaningful results [14, 31, 19, 32, 16].

\*This work was done while visiting Huawei Noah's Ark Lab.

†Corresponding author.

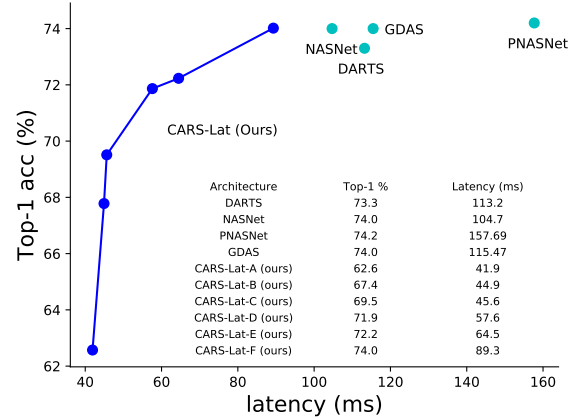


Figure 1. CARS-Lat models are searched on CIFAR-10 dataset which considers mobile device latency and validation performance. The top-1 accuracy is the searched models' performance on the ILSVRC2012 dataset.

To accelerate this process and make it automated, automatic network architecture search (NAS) has been proposed, and the learned architectures have exceeded those human-designed architectures on a variety of tasks. However, these searching methods usually require lots of computational resources to search for architectures of acceptable performance.

Techniques for searching network architectures are mainly clustered into three groups, Evolution Algorithm (EA) based, Reinforcement Learning (RL) based, and gradient-based. EA-based works, [26, 29, 28, 35] initialized a set of models and evolved for better architectures, which is time consuming, *e.g.* Real et al. takes 3150 GPU days for training [29]. RL based works, [41, 2, 39] use a controller to predict a sequence of operations, and train different architectures to gain the reward. Given an architecture, both

methods have to train it for a large number of epochs, and then evaluate its performance to guide for evolution or optimize the controller, which makes the searching stage less efficient. Gradient-based methods like DARTS [22] first train a supernet and introduce attention mechanism on the connections while searching, then remove weak connections after searching. This phase is conducted by gradient descent optimization and is quite efficient. However, it suffers from a lack of variety for the searched architectures.

Although some experiments in previous works [28] show that the evolutionary algorithm performs better than RL based approaches for searching neural architectures with better performance, the searching cost of EA is much expensive due to the evaluating procedure of each individual, *i.e.* a neural network in the evolutionary algorithm is independently evaluated. Moreover, there could be some architectures with extremely worse performance in the searching space. If we directly follow the weight sharing approach proposed by ENAS [27], the supernet has to be trained to compensate for those worse searching space. It is therefore necessary to reform existing evolutionary algorithms for an efficient yet accurate neural architecture search.

In this paper, we propose an efficient EA-based neural architecture search framework. To maximally utilize the knowledge we have learned in the last evolution iteration, a continuous evolution strategy is developed. Specifically, a supernet is first initialized with considerable cells and blocks. Individuals in the evolutionary algorithm representing architectures derived in the supernet will be generated through several benchmark operations (*i.e.* crossover and mutation). Non-dominated sort strategy is adopted to select several excellent architectures with different model sizes and accuracies, and corresponding cells in the supernet will be updated for subsequent optimizing. The evolution procedure in the next iteration is continuously executing based on the updated supernet and the multi-objective solution set obtained by the non-dominated sorting. In addition, we propose to exploit a protection mechanism to avoid the small model trap problem. The proposed continuous evolution architecture search (CARS) can provide a series of models on the Pareto front with high efficiency. The superiority of our method is verified on benchmark datasets over the state-of-the-art methods.

## 2. Related Works

### 2.1. Network Architecture Search

Network architecture search problem always contains two steps: network parameter optimization and network architecture optimization. The network parameter optimization step adjusts the parameters in the standard layers (*i.e.* convolution, batch normalization, fully connected layer) which is similar to train a standard neural network (*i.e.*

ResNet). The network architecture optimization step learns accurate network architectures with superior performance.

The parameter optimization step contains independent and sharing optimization. Independent optimization learns each network separately, *i.e.* AmoebaNet [28] takes thousands of GPU days to evaluate thousands of models. To accelerate training time, [9, 8] initialize parameters by network morphism. One-shot method [1] step further by sharing all the parameters for different architectures among one supernet. Rather than training thousands of different architectures, only one supernet is required to be optimized.

The architecture optimization step include RL-based, EA-based, and gradient-based approaches. RL-based methods [41, 42, 27] use recurrent networks as network architecture controller, and the performance of the generated architectures are utilized as the rewards for training the controller. The controller would converge during training and finally outputs architectures with superior performance. EA-based approaches [35, 28] search architectures with the help of evolutionary algorithms. The validation performance of each individual is utilized as the fitness to evolve the next generation. Gradient-based approaches [22, 36, 34] view the network architecture as a set of learnable parameters and optimize the architecture by the standard back-propagation algorithm.

### 2.2. Multi-objective Network Architecture Search

Considering multiple complementary objectives, *i.e.* performance, the number of parameters, float operations (FLOPs), and latency, there is no single architecture surpass all the others along with all the objectives. Therefore, a set of architectures within the Pareto front are desired. Many different works have been proposed to deal with multi-objective network architecture search. NEMO [17] and MNasNet [33] targets at speed and accuracy. DPP-Net and LEMONADE [7, 9] considers device-related and device-agnostic objectives. MONAS [15] targets at accuracy and energy. NSGANet [24] considers FLOPs and accuracy.

These methods are less efficient for models are optimized separately. In contrast, our architecture optimization and parameter optimization steps are conducted alternatively. Besides, the parameters for different architectures are shared, thus much more efficient during searching.

## 3. Approach

In this section, we develop a novel continuous evolutionary approach for searching neural architectures, including two procedures, *i.e.* parameters optimization and architecture optimization.

We use Genetic Algorithm (GA) for architecture evolution, because GA could provide a vast searching space. We maintain a set of architectures (a.k.a. connections)  $C =$

$\{C_1, \dots, C_N\}$ , where  $N$  is the population size. The architectures in the population gradually update according to the proposed pNSGA-III method during architecture optimization step. To make the searching stage efficient, we maintain a supernet  $\mathcal{N}$  which shares parameters  $W$  for different architectures, which could dramatically reduce the computational complexity of separately training these different architectures during searching.

### 3.1. Supernet of CARS

Different architectures are sampled from the supernet  $\mathcal{N}$ , and each network  $\mathcal{N}_i$  can be stood by a set of parameters with float precision  $W_i$  and a set of binary connection parameters (*i.e.*  $\{0, 1\}$ )  $C_i$ . The 0-element connection means the network does not contain this connection to transform data flow and 1-element connection means the network use this connection. From this point of view, each network  $\mathcal{N}_i$  could be represented as  $(W_i, C_i)$  pair.

Full precision parameters  $W$  are shared by a set of networks. If these network architectures are fixed, the optimal  $W$  could be optimized by standard back-propagation, which fits for all the networks  $\mathcal{N}_i$  to achieve higher recognition performance. After the parameters are converged, we could alternately optimize the binary connections  $C$  by GA algorithm. These two stages form the main optimization for our proposed continuous evolution algorithm and are processed in an alternative training way. We introduce these two kinds of update in the following.

### 3.2. Parameter Optimization

Parameter  $W$  is the collection of all the parameters in the network. The parameter  $W_i$  of the  $i$  th individual is  $W_i = W \odot C_i$ ,  $i \in \{1, \dots, N\}$ , where the  $\odot$  is the mask operation which keep the parameters of the complete graph only for the positions corresponding to 1-elements in the connection  $C_i$ . With input data  $X$  fed into the network, the predictions of this network is  $P_i = \mathcal{N}_i(X, W_i)$ , where  $\mathcal{N}_i$  is the  $i$  th architecture and  $W_i$  is the sampled weights. Given ground truth  $Y$ , the prediction loss can be expressed as  $L_i$ . The gradient of  $W_i$  can be calculated as

$$dW_i = \frac{\partial L_i}{\partial W_i} = \frac{\partial L_i}{\partial W} \odot C_i. \quad (1)$$

Parameters  $W$  should fit all the individuals, and thus the gradients for all networks are accumulated to calculate the gradient of parameters  $W$

$$dW = \frac{1}{N} \sum_{i=1}^N dW_i = \frac{1}{N} \sum_{i=1}^N \frac{\partial L_i}{\partial W} \odot C_i. \quad (2)$$

Any layer is only optimized by networks which use this layer during forward. By collecting the gradients of individuals in the population, the parameters  $W$  are updated through SGD algorithm.

As we have maintained a large set of architectures with shared weights in the supernet, we borrow the idea of stochastic gradient descent and use mini-batch architectures for updating parameters. Accumulating the gradients for all networks would take much time for one-step gradient descent, and thus we use mini-batch architectures for updating shared weights. We use  $N_b$  different architectures where  $N_b < N$ , and the indexes of architectures are  $\{n_1, \dots, n_b\}$  to update parameters. The efficient parameter updating of Eqn 2 is detailed as Eqn 3

$$dW \approx \frac{1}{N_b} \sum_{j=1}^{N_b} \frac{\partial L_{n_j}}{\partial W_{n_j}}. \quad (3)$$

Hence, the gradient over a mini-batch of architectures is taken as an approximation of the averaged gradients of all the  $N$  different individuals. The time cost for each update could be largely reduced, and the appropriate mini-batch size leads to a balance between efficiency and accuracy.

### 3.3. Architecture Optimization

As for the architecture optimization procedure, we use the evolution algorithm with the non-dominated sorting strategy, which was introduced in the NSGA-III [5]. Denoting  $\{\mathcal{N}_1, \dots, \mathcal{N}_N\}$  as  $N$  different networks and  $\{\mathcal{F}_1, \dots, \mathcal{F}_M\}$  as  $M$  different measurements we want to minimize. In general, these measurements, including the number of parameters, float operations, latency, and performance, could have some conflicts, which increases the difficulty in discovering an optimal solution that minimizes all these metrics.

In practice, if architecture  $\mathcal{N}_i$  dominates  $\mathcal{N}_j$ , the performance of  $\mathcal{N}_i$  is no less than that of  $\mathcal{N}_j$  on all the measurements and behaves better on at least one metric. Formally, the definition of domination can be given as below.

**Definition 1.** Consider two networks  $\mathcal{N}_i$  and network  $\mathcal{N}_j$ , and a series of measurements  $\{\mathcal{F}_1, \dots, \mathcal{F}_M\}$  we want to minimize. If

$$\begin{aligned} \mathcal{F}_k(\mathcal{N}_i) &\leq \mathcal{F}_k(\mathcal{N}_j), \quad \forall k \in \{1, \dots, M\} \\ \mathcal{F}_k(\mathcal{N}_i) &< \mathcal{F}_k(\mathcal{N}_j), \quad \exists k \in \{1, \dots, M\}, \end{aligned} \quad (4)$$

$\mathcal{N}_i$  is said to dominate  $\mathcal{N}_j$ , *i.e.*  $\mathcal{N}_i \preceq \mathcal{N}_j$ .

According to the above definition, if  $\mathcal{N}_i$  dominates  $\mathcal{N}_j$ ,  $\mathcal{N}_j$  can be replaced by  $\mathcal{N}_i$  during the evolution procedure since  $\mathcal{N}_i$  performs better in terms of at least one metric and no worse on other metrics. By exploiting this approach, we can select a series of excellent neural architectures from the population generated in the current iteration. Then, these networks can be utilized for updating the corresponding parameters in the supernet.

Although the above non-dominated sorting using NSGA-III method [5] can help us to select some better models for updating parameters, there exists a small model

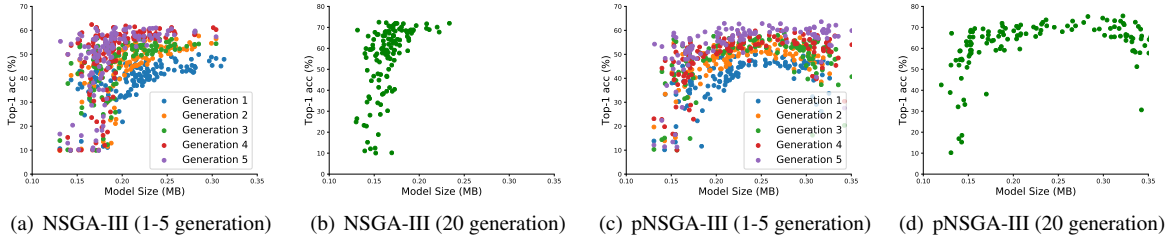


Figure 2. Comparison between different evolution strategy. supernet is trained with training set and evaluated on the validation set. Figure 2(a) shows five evolution generations using NSGA-III. Evolution with NSGA-III suffers from small model trap, which lead the distribution bias to smaller models, and Figure 2(b) shows the distribution of evolution with NSGA-III after 20 generations, where the maintained architectures are all the small models. Figure 2(c) shows the evolution generations using proposed pNSGA-III. Evolution with pNSGA-III provides protection for larger models, and Figure 2(d) shows the distribution of evolution with pNSGA-III after 20 generations, where the maintained architectures covers a large range over model size.

---

**Algorithm 1** Continuous Evolution for Efficient Neural Architecture Search

---

Input: supernet  $\mathcal{N}$ , connections  $C = \{C_1^{(0)}, \dots, C_N^{(0)}\}$ , offspring expand ratio  $t$ , evolution number  $E_{evo}$ , multi-objectives  $\{\mathcal{F}_1, \dots, \mathcal{F}_M\}$ , parameter optimization iterations  $I_{param}$  between evolution generations, and criterion  $\mathcal{C}$ .

- 1: Warmup parameters of supernet  $\mathcal{N}$ .
- 2: **for**  $e = 1, \dots, E_{evo}$  **do**
- 3:   **for**  $i = 1, \dots, I_{param}$  **do**
- 4:     Get training mini-batch data  $X$  and target  $Y$ .
- 5:     Random sample  $N_b$  different connections  $C_1, \dots, C_{n_b}$ , and mask the supernet  $\mathcal{N}$  to form sampled networks  $N_1, \dots, N_{n_b}$ .
- 6:     Calculate averaged loss by forward  $N_b$  different sampled architectures  $L = \frac{1}{N_b} \sum_{i=1}^{N_b} \mathcal{C}(\mathcal{N}_i(X), Y)$ .
- 7:     Calculate derivative to parameters  $W$  according to Eqn 3 and update network parameters.
- 8:   **end for**
- 9:   Update  $\{C_1^{(i)}, \dots, C_N^{(i)}\}$  within Pareto front using pNSGA-III.
- 10: **end for**

Output: Architectures  $C = \{C_1^{(E_{evo})}, \dots, C_N^{(E_{evo})}\}$  with various requirements of objectives.

---

trap phenomenon during the searching procedure. Specifically, since the parameters in the supernet still need optimization, the accuracy of each individual architecture in the current iteration may not always stand for its performance that can be eventually achieved, as discussed in NASBench-101 [38]. Thus, some smaller models of fewer parameters but higher test accuracy tend to dominate those larger models of lower fitness but have the potential for achieving higher accuracies as shown in Figure 3.

Therefore, we propose to improve the conventional NSGA-III for protecting these larger models, namely

pNSGA-III. More specifically, the pNSGA-III algorithm takes the increasing speed of performance into consideration. We take the validation performance and the number of parameters as an example. For NSGA-III method, the non-dominated sorting algorithm considering two different objectives and select individuals according to the sorted Pareto stages. For the proposed pNSGA-III, besides considering the number of parameters and performance, we also conduct a non-dominated sorting algorithm considering the performance increasing speed and the number of parameters. Then the two different Pareto stages are merged and gradually select individuals from the first stage to fill the generation. In this way, the large networks with slower performance increasing speed could be kept in the population.

Figure 2 illustrates the selected models in every iteration using the conventional NSGA-III and the modified pNSGA-III, respectively. It is obvious that the pNSGA-III can provide models with a wide range of model sizes and comparable performance to that of the NSGA-III.

### 3.4. Continuous Evolution for CARS

In summary, the proposed CARS for searching optimal neural architecture has two stages: 1) Architecture Optimization 2) Parameter Optimization. In addition, parameter warmup is also introduced for the few epochs.

**Parameter Warmup.** Since the shared weights of our supernet are initialized randomly, if a set of architectures are also initialized with random architectures, the most frequently used blocks for all the architectures would be trained more times compared with other blocks. Thus, by following one-shot NAS methods [1, 12, 4], we use a uniform sampling strategy to initialize the parameters in the supernet. In this way, the supernet trains each possible architecture with the same possibility which means each path in a searching space is sampled equivalently. For example, in DARTS [22] pipeline, there are 8 different operations for each node, including convolution, pooling, identity

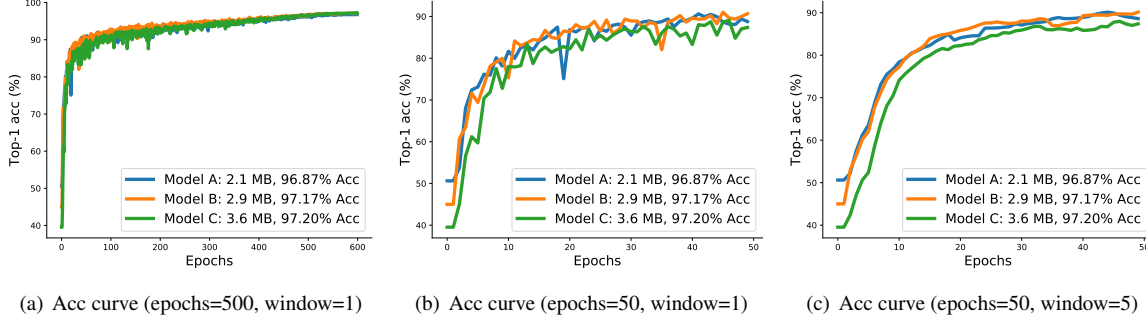


Figure 3. Comparison of accuracy curves of three models with different sizes. The left figure is the accuracy curve of training for 600 epochs, the middle figure is the accuracy curve for the first 50 epochs, and the right figure is smooth by a window size of 5.

mapping, and no connection. Each operation will be trained with a probability of  $\frac{1}{8}$ . This initialization step initialized the sharing weights of the supernet for few epochs.

**Architecture Optimization.** After initializing the parameters of the supernet, we first randomly sample  $N$  different architectures, where  $N$  is the number of maintained individuals among Pareto front using pNSGA-III, and we define  $N$  as a hyper-parameter. During the architecture evolution step, we first generate  $t \times N$  offsprings, where  $t$  is the hyper-parameter to control the number of offsprings. We then use pNSGA-III to select  $N$  individuals from  $(t + 1) \times N$  individuals.

**Parameter Optimization.** Given a set of architectures, we use the proposed mini-batch architectures update scheme for parameter optimization according to Eqn 3.

Algorithm 1 summarizes the detailed procedure of the proposed continuous evolutionary algorithm for searching neural architectures.

### 3.5. Search Time Analysis

During the searching stage of CARS, the training data is split into train/val set, and the train set is used for parameter optimization, meanwhile, the val set is used for architecture optimization stage. Assuming the average training time on the train set for one architecture is  $T_{tr}$ , and the inference time on val set is  $T_{val}$ . The first warmup stage takes  $E_{warm}$  epochs, and it needs  $T_{warm} = E_{warm} \times T_{tr}$  in this stage to initialize parameters in the supernet  $\mathcal{N}$ .

Assuming the architectures evolve for  $E_{evo}$  iterations in total. And each iteration contains parameter optimization and architecture optimization stages. The parameter optimization stage trains the supernet for  $I_{param}$  epochs on train set during one evolution iteration, thus the time cost for parameter optimization in one evolution iteration is  $T_{param} = I_{param} \times T_{tr} \times N_b$  if we consider the mini-batch size  $N_b$ . For the architecture optimization stage, all the individuals can be inferred in parallel, so the search-

ing time in this stage could be calculated as  $T_{arch} = T_{val}$ . Thus the total time cost for  $E_{evo}$  evolution iterations is  $T_{evo} = E_{evo} \times (T_{param} + T_{arch})$ . All the searching time cost in CARS is,

$$\begin{aligned} T_{total} &= T_{warm} + T_{evo} \\ &= E_{warm} \times T_{tr} + \\ &\quad E_{evo} \times (I_{param} \times T_{tr} \times N_b + T_{arch}) \end{aligned} \quad (5)$$

## 4. Experiments

In this section, we first introduce the datasets, backbone, and evolution details we used in our experiments. We then search a set of architectures using our proposed CARS on CIFAR-10 [18] dataset and show the necessity of proposed pNSGA-III over popularly used NSGA-III [5] during continuous evolution. We consider three objectives. Besides performance, we separately consider device-aware latency and device-agnostic model size. After that, we evaluate our searched architectures on CIFAR-10 dataset and ILSVRC2012 [6] large scale recognition dataset to demonstrate the effectiveness of CARS.

### 4.1. Experimental Settings

**Datasets.** Our CARS experiments are performed on CIFAR-10 [18] and ILSVRC2012 [6] large scale image classification task. These two datasets are popular benchmarks on the recognition task. We search architectures on CIFAR-10 dataset, and the searched architectures are evaluated on CIFAR-10 and ILSVRC2012 datasets.

**supernet Backbones.** To illustrate the effectiveness of our method, we evaluate our CARS over state-of-the-art NAS system DARTS [22]. DARTS is a differentiable NAS system searching for cells and shares the searched cells from shallow layers to deeper layers. The searching space contains 8 different blocks, including four types of convolution, two kinds of pooling, skip connect and no connection. DARTS searches for two kinds of topology for normal cell and reduction cell. A normal cell is used for the layers

Table 1. Comparison with state-of-the-art image classifiers on CIFAR-10 dataset. The multi-objectives used for architecture optimization are performance and model size.

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
DenseNet-BC [16]	3.46	25.6	-	manual
PNAS [20]	3.41	3.2	225	SMBO
ENAS + cutout [27]	2.91	4.2	4	RL
NASNet-A + cutout [42]	2.65	3.3	2000	RL
AmoebaNet-A + cutout [28]	3.12	3.1	3150	evolution
Hierarchical evolution [21]	3.75	15.7	300	evolution
SNAS (mild) + cutout [36]	2.98	2.9	1.5	gradient
SNAS (moderate) + cutout [36]	2.85	2.8	1.5	gradient
SNAS (aggressive) + cutout [36]	3.10	2.3	1.5	gradient
DARTS (first) + cutout [22]	3.00	3.3	1.5	gradient
DARTS (second) + cutout [22]	2.76	3.3	4	gradient
Random Search [22]	3.29	3.2	4	random
RENA [40]	3.87	3.4	-	RL
NSGANet [24]	3.85	3.3	8	evolution
LEMONADE [9]	3.05	4.7	80	evolution
CARS-A	3.00	2.4	0.4	evolution
CARS-B	2.87	2.7	0.4	evolution
CARS-C	2.84	2.8	0.4	evolution
CARS-D	2.95	2.9	0.4	evolution
CARS-E	2.86	3.0	0.4	evolution
CARS-F	2.79	3.1	0.4	evolution
CARS-G	2.74	3.2	0.4	evolution
CARS-H	2.66	3.3	0.4	evolution
CARS-I	2.62	3.6	0.4	evolution

that have the same spatial size for input feature and output feature. Reduction cell is used for layers with downsampling on input feature maps. After searching for these two kinds of cells, the network is constructed by stacking a set of searched cells.

**Evolution Details.** In the DARTS method, each intermediate node in a cell is connected with two previous nodes. Thus each node has its own searching space. Crossover and mutation are conducted on the corresponding nodes with the same searching space. Both crossover ratio and mutation ratio are set as 0.25, 0.25, and we randomly generate new architectures with a probability of 0.5. For the crossover operation, each node has a ratio of 0.5 to crossover its connections, and for mutation operation, each node has a ratio of 0.5 to be randomly reassigned.

## 4.2. Experiments on CIFAR-10

Our experiments on CIFAR-10 include the demonstration of small model trap phenomenon, NSGA method comparison, searching and evaluation. For NSGA method comparison, we compare NSGA-III with proposed pNSGA-III. After that, we conduct CARS on CIFAR-10 and search for a set of architectures which separately considers latency, model size and performance. At last, we examine the searched architectures with different computing resources.

**Small Model Trap** In Figure 3, we train three models of different sizes, 2.1M, 2.9M and 3.6M, respectively. After training for 600 epochs, the accuracy on the CIFAR-10 dataset is positively correlated with the model size, which are 96.87%, 97.17% and 97.20%. We observe the accuracy curve of the first 50 epochs, and propose that there are two main reasons for the small model trap phenomenon, which are (1) small models are naturally grow with higher speed and (2) the fluctuation of the accuracy. For Model-C, its accuracy is consistently lower than model-A and model-B in the first 50 epochs, and its models is larger. Therefore, if NSGA algorithm is used, such model will be eliminated, which is the first reason we proposed. This is because larger models are more complex, thus harder to be optimized. For Model-B and Model-A, the growth rate of accuracy is similar (Figure 3 (right)), however, due to the fluctuation of the accuracy during training (Figure 3 (middle)), if the accuracy of Model-A is higher than Model-B in one epoch, and we use GA algorithm to update architectures, Model-B would be eliminated, which verify the second reason we proposed. Both two reasons may lead to the elimination of large models in the process of model updating, which reflects the necessity of using pNSGA-III in the process of alternative optimization.

**NSGA-III vs. pNSGA-III.** We train CARS with different NSGA methods during architecture optimization stage for



comparison and objectives are the model size and validation performance. We visualize the growing trend for different architecture optimization methods after parameter warmup stage. As Figure 2 shows, CARS with NSGA-III would encounter the small model trap problem, for small models tend to eliminate large models during architecture optimization stage. In contrast, architecture optimization using pNSGA-III protects larger models which have the potential to increase their accuracies during later epochs but converge slower than small models at the beginning. To search for several models with various computing resources, it is essential to maintain larger models in the population rather than dropping them during architecture optimization stage.

**Search on CIFAR-10.** To search for a set of architectures using CARS, we split the CIFAR-10 train set by 25,000 in searching which named as train set and 25,000 for evaluation, which named as val set. The split strategy is same with DARTS and SNAS. We search for 500 epochs in total, and the parameter warmup stage lasts for the first 50 epochs. After that, we initialize the population, which maintains 128 different architectures and gradually evolve them using proposed pNSGA-III. The parameter optimization stage is optimized by train set and trains for 10 epochs during one evolution iteration. The architecture optimization stage uses the val set to update architectures according to proposed pNSGA-III. We separately search by considering performance and model size/latency.

**Search Time analysis.** For the experiment of considering the model size and performance, the training time on the train set  $T_{tr}$  takes around 1 minute, and the inference time on val set  $T_{val}$  is around 5 seconds. For the first initialization stage, it trains for 50 epochs, so the time cost in this stage  $T_{warm}$  takes around an hour. The continuous evolution algorithm evolves the architectures for  $E_{evo}$  iterations. For the architecture optimization stage in one iteration, the parallel evaluation time is  $T_{arch} = T_{val}$ . For the parameter optimization stage, we set  $N_b$  to be 1 in our experiments since different batch size does not affect the overall growth trend for individuals in the population which is discussed in supplementary material, and we train the supernet  $N$  for 10 epochs in one evolution iteration. So the parameter optimization time  $T_{param}$  is about 10 minutes. Thus the time cost for total evolution iteration  $T_{evo}$  is around 9 hours, and the total searching time  $T_{total}$  is around 0.4 GPU day. For the experiment of considering the latency and performance, the running latency for each model is evaluated during architecture optimization step. Thus the searching time is around 0.5 GPU day.

**Evaluate on CIFAR-10.** After finishing the CARS, our method keeps  $N = 128$  different architectures, and we eval-

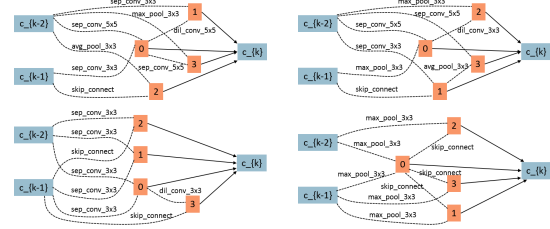


Figure 4. CARS-H and DARTS. On the top are the normal and reduction blocks of CARS-H, and the bottom are the normal and reduction blocks in DARTS (second order).

uate architectures which have a similar number of parameters with previous works [22, 36] for comparison. We retrain the searched architectures on CIFAR-10 dataset with all the training data and evaluate on the test set. All the training parameters are the same with DARTS [22].

We compare the searched architectures with state-of-the-art methods which utilize similar searching space in Table 1. All the searched architectures can be found in the supplementary material. Our searched architectures have the number of parameters vary from 2.4M to 3.6M on CIFAR-10 dataset, and the performance of these architectures are on par with the state-of-the-art NAS methods, while if we evolve with NSGA-III method, we would only search for a set of architectures with approximately 2.4M parameters without larger models, and the models perform relatively poor.

Compared to previous methods like DARTS and SNAS, our method is capable of searching for complete architectures over a large range of the searching space. Our CARS-G model achieves comparable accuracy with DARTS (second order) resulting in an approximate 2.75 error rate with smaller model size. With the same 3.3M parameters as DARTS (second order), our CARS-H achieves lower test error, 2.66% vs. 2.76%. For the small models, our searched CARS-A/C/D also achieve comparable results with SNAS. Besides, our large model CARS-I achieves lower error rate 2.62% with slightly more parameters. The overall trend from CARS-A to CARS-J is that the error rate gradually decreases while increasing the model size. These models are all Pareto solutions. Compared to other multi-objective methods like RENA [40], NSGANet [24] and LEMON-ADE [9], our searched architectures also show superior performance over these methods.

**Comparison on Searched Block.** In order to have an explicit understanding of the proposed method, we further visualize the normal and reduction blocks searched using CARS and DARTS in Figure 5, respectively. Wherein, the CARS-H and DARTS (second order) have a similar number of parameters (3.3M), but the CARS-H has higher accuracy. It can be found in Figure 5 that, there are more parameters in the CARS-H reduction block for preserving more useful

Table 2. An overall comparison on ILSVRC2012 dataset. The CARS models are the architectures searched on the CIFAR-10 dataset, then evaluated on ILSVRC2012 dataset.

Architecture	Top-1 Acc (%)	Top-5 Acc (%)	Params (M)	+ $\times$ (M)	Search Cost (GPU days)	Search Method
ResNet50 [14]	75.3	92.2	25.6	4100	-	manual
InceptionV1 [32]	69.8	90.1	6.6	1448	-	manual
MobileNetV2 (1 $\times$ ) [30]	72.0	90.4	3.4	300	-	manual
ShuffleNetV2 (2 $\times$ ) [25]	74.9	90.1	7.4	591	-	manual
PNAS [20]	74.2	91.9	5.1	588	224	SMBO
SNAS (mild) [36]	72.7	90.8	4.3	522	1.5	gradient
DARTS [22]	73.3	91.3	4.7	574	4	gradient
PARSEC [3]	74.0	91.6	5.6	548	1	gradient
NASNet-A [42]	74.0	91.6	5.3	564	2000	RL
NASNet-B [42]	72.8	91.3	5.3	488	2000	RL
NASNet-C [42]	72.5	91.0	4.9	558	2000	RL
AmoebaNet-A [28]	74.5	92.0	5.1	555	3150	evolution
AmoebaNet-B [28]	74.0	91.5	5.3	555	3150	evolution
AmoebaNet-C [28]	75.7	92.4	6.4	570	3150	evolution
RCNet [37]	72.2	91.0	3.4	294	8	gradient
CARS-A	72.8	90.8	3.7	430	0.4	evolution
CARS-B	73.1	91.3	4.0	463	0.4	evolution
CARS-C	73.3	91.4	4.2	480	0.4	evolution
CARS-D	73.3	91.5	4.3	496	0.4	evolution
CARS-E	73.7	91.6	4.4	510	0.4	evolution
CARS-F	74.1	91.8	4.5	530	0.4	evolution
CARS-G	74.2	91.9	4.7	537	0.4	evolution
CARS-H	74.7	92.2	4.8	559	0.4	evolution
CARS-I	75.2	92.5	5.1	591	0.4	evolution

information, and the size of the normal block of CARS-H is smaller than that of the DARTS (second order) to avoid unnecessary computations. This phenomenon mainly because the proposed method using EA has much larger searching space, and the genetic operations can effectively jump out of the local optimum, which demonstrates its superiority.

### 4.3. Evaluate on ILSVRC2012

For those architectures searched on CIFAR-10 dataset, we evaluate the transferability of the architectures on ILSVRC2012 dataset. We use 8 Nvidia Tesla V100 to train in parallel, and the batch size is set to 640. We train 250 epochs in total, the learning rate is set to 0.5 with linear decay scheduler, and we warmup [11] the learning rate for the first five epochs due to the large batch size we used. Momentum is set to 0.9, and weight decay is set to  $3e-5$ . Label smooth is also included with a smooth ratio of 0.1.

The evaluated results show the transferability capability of our searched architectures. Our models cover an extensive range of the parameters ranging from 3.7M to 5.1M with 430 to 590 MFLOPs. For different deploy environments, we can easily select an architecture which satisfies the restrictions.

Our CARS-I surpasses PNAS by 1% on Top1 with the same number of parameters and approximate FLOPs. The CARS-G shows superior results over DARTS by 0.9%

Top1 accuracy with the same number of parameters. Also, CARS-D surpasses SNAS (mild) by 0.6% Top1 accuracy with the same number of parameters. For different models of NASNet and AmoebaNet, our method also has various models that achieve higher accuracy using the same number of parameters. By using the proposed pNSGA-III, the larger architectures like CARS-I could be protected during architecture optimization stages. Because of the efficient parameter sharing, we could search a set of superior transferable architectures during the one-time search.

For experiments that search architectures by runtime latency and performance, we use the evaluated runtime latency on the mobile phone as an objective and the performance as another. These two objectives are used for generating the next population. We evaluated the searched architectures on ILSVRC2012 in Figure 1. The searched architectures cover an actual runtime latency from 40ms to 90ms and surpass the counterparts.

## 5. Conclusion

The evolutionary algorithm based NAS methods are able to find models with high-performance, but the searching time of these methods is extremely long, the main problem is each candidate network is trained separately. In order to make this efficient, we propose continuous evolution architecture search, namely, CARS, which maximally utilizes



the learned knowledge such as architectures and parameters in the latest evolution iteration. A supernet is constructed with considerable cells and blocks. Individuals are generated through the benchmark operations in an evolutionary algorithm. Non-dominated sort strategy is utilized to select architectures with different model sizes and high accuracies for updating the supernet. Experiments on benchmark datasets show that the proposed CARS can provide a number of architectures on the Pareto front with high efficiency, *e.g.* the searching cost on the CIFAR-10 benchmark is only 0.4 GPU days. The searched models are superior to models produced by state-of-the-art methods in terms of both model size/latency and accuracies.

## 6. Appendix

We list all the searched architectures using CARS on the CIFAR10 dataset and explore the effect of hyper-parameters utilized in CARS.

### 6.1. Network Architectures

In Table 3, we list all the searched architectures considering the model size and validation performance. In Table 4, the searched architecture of considering latency and validation performance are listed. Notations are same with DARTS [22].

### 6.2. Impact of Batch Size in pNSGA-III

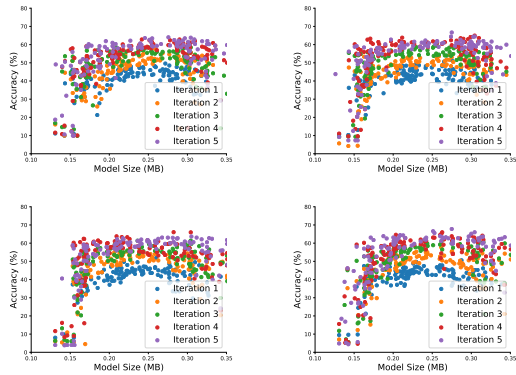


Figure 5. The evolution trend of using pNSGA-III with different batchsize  $N_b \in \{1, 2, 3, 4\}$ .

For the parameter optimization stage in the continuous evolution,  $N_b$  different architectures are sampled from the maintained population to update parameters. We evaluated different batch size  $N_b \in \{1, 2, 3, 4\}$  and find that they all grow with a similar trend. Thus we use  $N_b = 1$  in our experiment which is similar to ENAS [27].

Table 3. Searched architectures by CARS using DARTS backbone on CIFAR10 dataset considering model size and performance. Notations are same with DARTS. Sep $k$  denotes separable convolution with kernel size  $k$ . Dil $k$  denotes dilated separable convolution with kernel size  $k$ . Max/Avg3 denotes Max/Avg Pooling with size 3. Skip denotes identity connection. None denotes no connection. C(k-2) and C(k-1) denotes the previous two cells connected with current cell C(k). N1 to N4 denotes four intermediate nodes within cell C(k).

Architecture	Cell Type	N1	N2	N3	N4
CARS-A	Normal	Skip C(k-2) Sep5 C(k-1)	Max3 C(k-2) Avg3 C(k-1)	Max3 C(k-2) Max3 C(k-1)	Sep3 C(k-2) Dil5 N1
	Reduce	Avg3 C(k-2) Max3 C(k-1)	Max3 C(k-2) Skip C(k-1)	Max3 C(k-2) Dil5 C(k-1)	Dil5 C(k-2) Skip N1
CARS-B	Normal	Sep5 C(k-2) Dil3 C(k-1)	Sep3 C(k-2) Avg3 N1	Dil3 C(k-2) Max3 C(k-1)	Avg3 C(k-2) Skip C(k-1)
	Reduce	Sep5 C(k-2) Skip C(k-1)	Sep3 C(k-2) Max3 C(k-1)	Avg3 C(k-2) Avg3 C(k-1)	Dil3 N2 Max3 C(k-2)
CARS-C	Normal	Sep5 C(k-2) Skip C(k-1)	Skip C(k-2) Skip C(k-1)	Skip C(k-2) Max3 C(k-1)	Sep5 C(k-2) Sep3 C(k-1)
	Reduce	Max3 C(k-2) Max3 C(k-1)	Sep5 C(k-2) Sep5 C(k-1)	Dil5 C(k-2) Max3 C(k-1)	Sep5 C(k-2) Dil3 C(k-1)
CARS-D	Normal	Sep5 C(k-2) Dil3 C(k-1)	Skip C(k-2) Avg3 C(k-1)	Skip C(k-2) Max3 C(k-1)	Sep5 C(k-2) Sep3 C(k-1)
	Reduce	Max3 C(k-2) Max3 C(k-1)	Max3 C(k-2) Sep3 C(k-1)	Dil5 C(k-2) Max3 C(k-1)	Sep5 C(k-1) Dil3 C(k-1)
CARS-E	Normal	Sep3 C(k-2) Sep3 C(k-1)	Skip C(k-2) Sep3 N1	Avg3 C(k-1) Sep3 N1	Skip N2 Skip N3
	Reduce	Skip C(k-2) Dil3 C(k-1)	Avg3 C(k-2) Skip N1	Sep3 N1 Max3 C(k-2)	Avg3 C(k-2) Sep3 N3
CARS-F	Normal	Skip C(k-2) Sep5 C(k-1)	Sep5 C(k-2) Skip N1	Sep5 N2 Max3 C(k-2)	Skip C(k-2) Sep3 C(k-1)
	Reduce	Avg3 C(k-2) Sep5 C(k-1)	Dil3 C(k-2) Dil5 C(k-1)	Sep5 C(k-1) Skip N1	Max3 C(k-2) Max3 C(k-1)
CARS-G	Normal	Max3 C(k-2) Dil5 C(k-1)	Sep3 C(k-2) Skip C(k-1)	Dil5 C(k-2) Sep4 C(k-1)	Avg3 C(k-2) Sep3 C(k-1)
	Reduce	Max3 C(k-2) Sep3 C(k-1)	Sep3 C(k-2) Sep5 C(k-1)	Sep3 C(k-2) Skip C(k-1)	Avg3 C(k-2) Dil3 C(k-1)
CARS-H	Normal	Sep5 C(k-2) Sep3 C(k-1)	Sep3 C(k-2) Dil5 N1	Avg3 C(k-2) Skip C(k-1)	Sep5 N1 Max3 C(k-2)
	Reduce	Sep5 C(k-2) Max3 C(k-1)	Sep3 C(k-2) Skip C(k-1)	Dil3 N1 Max3 C(k-2)	Sep5 C(k-2) Avg3 N2
CARS-I	Normal	Sep3 C(k-2) Sep3 C(k-1)	Skip C(k-2) Sep5 C(k-1)	Skip N1 Sep3 N2	Sep3 C(n-2) Dil5 N3
	Reduce	Dil3 C(k-2) Skip C(k-1)	Max3 C(k-2) Max3 N1	Skip C(k-1) Sep5 N2	Dil3 C(k-1) Max3 N3

Table 4. Searched architectures by CARS using DARTS backbone on CIFAR10 dataset considering latency and performance. Notations are same with DARTS. Notations are same with DARTS. Sep $k$  denotes separable convolution with kernel size  $k$ . Dil $k$  denotes dilated separable convolution with kernel size  $k$ . Max/Avg3 denotes Max/Avg Pooling with size 3. Skip denotes identity connection. None denotes no connection. C( $k-2$ ) and C( $k-1$ ) denotes the previous two cells connected with current cell C( $k$ ). N1 to N4 denotes four intermediate nodes within cell C( $k$ ).

Architecture	Top-1 (%)	Latency (ms)	Cell Type	N1	N2	N3	N4
CARS-Lat-A	62.6	41.9	Normal	Skip C( $k-2$ )	Avg3 C( $k-2$ )	Skip N1	Skip N1
			Reduce	Max3 C( $k-1$ ) Skip C( $k-2$ ) Sep5 C( $k-1$ )	Max3 C( $k-1$ ) Dil5 C( $k-2$ ) Max3 N1	Skip N2 Skip C( $k-2$ ) Max3 C( $k-1$ )	Skip N2 Skip C( $k-1$ ) Avg3 N3
CARS-Lat-B	67.8	44.9	Normal	Skip C( $k-2$ )	Skip C( $k-1$ )	Skip N1	Max3 C( $k-2$ )
			Reduce	Skip C( $k-1$ ) Max3 C( $k-2$ ) Max3 C( $k-1$ )	Dil3 N1 Skip C( $k-1$ ) Max3 C( $k-2$ )	Skip N2 Sep3 C( $k-2$ ) Max3 C( $k-1$ )	Max3 N1 Dil5 C( $k-2$ ) Avg3 N1
CARS-Lat-C	69.5	45.6	Normal	Skip C( $k-2$ )	Skip C( $k-2$ )	Max3 C( $k-1$ )	Dil3 N1
			Reduce	Avg3 C( $k-1$ ) Skip C( $k-2$ ) Sep5 C( $k-1$ )	Skip C( $k-1$ ) Avg3 C( $k-1$ ) Sep5 N1	Skip N2 Max3 C( $k-2$ ) Max3 C( $k-1$ )	Skip N3 Dil5 N1 Skip N3
CARS-Lat-D	71.9	57.6	Normal	Sep3 C( $k-2$ )	Skip C( $k-2$ )	Skip C( $k-1$ )	Dil3 N1
			Reduce	Skip C( $k-1$ ) Dil5 C( $k-2$ ) Sep5 C( $k-1$ )	Skip C( $k-1$ ) Avg3 C( $k-1$ ) Sep5 N1	Avg3 N2 Max3 C( $k-2$ ) Max3 C( $k-1$ )	Skip N3 Dil5 N1 Skip N3
CARS-Lat-E	72.0	60.8	Normal	Dil5 C( $k-2$ )	Skip C( $k-1$ )	Skip C( $k-1$ )	Skip C( $k-2$ )
			Reduce	Skip C( $k-1$ ) Skip C( $k-2$ ) Dil3 C( $k-1$ )	Avg3 N1 Sep3 C( $k-2$ ) Sep3 N1	Avg3 N1 Dil3 C( $k-2$ ) Avg3 N2	Max3 C( $k-1$ ) Sep3 C( $k-1$ ) Sep5 N1
CARS-Lat-F	72.2	64.5	Normal	Sep5 C( $k-2$ )	Skip C( $k-2$ )	Dil3 C( $k-1$ )	Skip C( $k-2$ )
			Reduce	Skip C( $k-1$ ) Sep5 C( $k-2$ ) Max3 C( $k-1$ )	Avg3 C( $k-1$ ) Sep5 C( $k-1$ ) Skip N1	Max3 C( $k-2$ ) Sep5 C( $k-2$ ) Sep5 C( $k-1$ )	Skip C( $k-1$ ) Sep5 N2 Dil3 N3
CARS-Lat-G	74.0	89.3	Normal	Sep5 C( $k-2$ )	Sep3 C( $k-2$ )	Dil3 C( $k-1$ )	Skip C( $k-2$ )
			Reduce	Skip C( $k-1$ ) Sep5 C( $k-2$ ) Max3 C( $k-1$ )	Sep5 N1 Sep3 C( $k-2$ ) Avg3 N1	Max3 C( $k-2$ ) Sep5 C( $k-2$ ) Sep5 C( $k-1$ )	Skip C( $k-1$ ) Sep5 N2 Dil3 N3

## References

- [1] Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc V. Le. Understanding and simplifying one-shot architecture search. *ICML*, 2018.
- [2] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. *AAAI*, 2018.
- [3] Francesco Paolo Casale, Jonathan Gordon, and Nicolo Fusi. Probabilistic neural architecture search. *arXiv*, 2019.
- [4] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv*, 2019.
- [5] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *TEC*, 18(4), 2014.
- [6] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. *CVPR*, 2009.
- [7] Jin-Dong Dong, An-Chieh Cheng, Da-Cheng Juan, Wei Wei, and Min Sun. Dpp-net: Device-aware progressive search for pareto-optimal neural architectures. *ECCV*, 2018.
- [8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Simple and efficient architecture search for convolutional neural networks. *ICLR*, 2018.
- [9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *ICLR*, 2019.
- [10] Ross Girshick. Fast r-cnn. *ICCV*, 2015.
- [11] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. Accurate, large minibatch sgd: Training imagenet in 1 hour. *arXiv*, 2017.
- [12] Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. Single path one-shot neural architecture search with uniform sampling. *arXiv*, 2019.
- [13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. *ICCV*, 2017.
- [14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CVPR*, 2016.
- [15] Chi-Hung Hsu, Shu-Huan Chang, Da-Cheng Juan, Jia-Yu Pan, Yu-Ting Chen, Wei Wei, and Shih-Chieh Chang. Monas: Multi-objective neural architecture search using reinforcement learning. *arXiv*, 2018.
- [16] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. *CVPR*, 2017.
- [17] Ye-Hoon Kim, Bhargava Reddy, Sojung Yun, and Chanwon Seo. Nemo : Neuro-evolution with multiobjective optimization of deep neural network for speed and accuracy. *ICML Workshop*, 2017.
- [18] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Cite-seer, 2009.
- [19] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *NeurIPS*, 2012.
- [20] Chenxi Liu, Barret Zoph, Maxim Neumann, Jonathon Shlens, Wei Hua, Li-Jia Li, Li Fei-Fei, Alan Yuille, Jonathan Huang, and Kevin Murphy. Progressive neural architecture search. *ECCV*, 2018.
- [21] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *ICLR*, 2018.
- [22] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.
- [23] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. *ECCV*, 2016.
- [24] Zhichao Lu, Ian Whalen, Vishnu Boddeti, Yashesh Dhebar, Kalyanmoy Deb, Erik Goodman, and Wolfgang Banzhaf. Nsga-net: A multi-objective genetic algorithm for neural architecture search. *GECCO*, 2019.
- [25] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *ECCV*, 2018.
- [26] Risto Mikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. Evolving deep neural networks. *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, 2019.
- [27] Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *ICML*, 2018.
- [28] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019.
- [29] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. Large-scale evolution of image classifiers. *ICML*, 2017.
- [30] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. *CVPR*, 2018.
- [31] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.
- [32] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. *CVPR*, 2015.
- [33] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *CVPR*, 2018.
- [34] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *CVPR*, 2019.
- [35] Lingxi Xie and Alan Yuille. Genetic cnn. *ICCV*, 2017.

- [36] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *ICLR*, 2019.
- [37] Yunyang Xiong, Ronak Mehta, and Vikas Singh. Resource constrained neural network architecture search. *ICCV*, 2019.
- [38] Chris Ying, Aaron Klein, Esteban Real, Eric Christiansen, Kevin Murphy, and Frank Hutter. Nas-bench-101: Towards reproducible neural architecture search. *ICML*, 2019.
- [39] Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. Practical block-wise neural network architecture generation. *CVPR*, 2018.
- [40] Yanqi Zhou, Siavash Ebrahimi, Sercan Ö Arık, Haonan Yu, Hairong Liu, and Greg Diamos. Resource-efficient neural architect. *arXiv*, 2018.
- [41] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.
- [42] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.