Neural Architecture Search Summary

1 Introduction

Neural Architecture Search (NAS) has been successfully applied in searching for model architectures in image classification and language modeling tasks. However, the high computation cost is a great limitation in the development of NAS as each evaluation requires training of a neural network. Recent developments have shown increasing of accuracy and decreasing of computation cost.

2 Related Work

Successful NAS algorithms are basicly using reinforcement learning [1, 2, 3, 4], evolutionary algorithms [5, 6, 7, 8], and gradient-based [9, 10, 11, 12, 13, 14]. When using reinforcement learning (RL), the controller RNN recursively generates the hyperparameters to describe the model architecture; this model is trained and the validation accuracy is taken as the reward function. The controller is updated by policy gradient. When using evolutionary algorithms (EA), the controller initializes a population of neural networks, randomly mutating some architectures and inserting them back to the population; the top performing models generate "children" and keep evolution; the model with the highest validation accuracy is taken as the final model. When using gradient-based search, unlike RL and EA that perform search in a discrete space, the architecture search space is continuous. Some works [9] represent the continuous space as mixture weights of operations between each pair of nodes, some [15] encode the neural architecture into a continuous space. The following subsections explain each method in details with relevant examples.

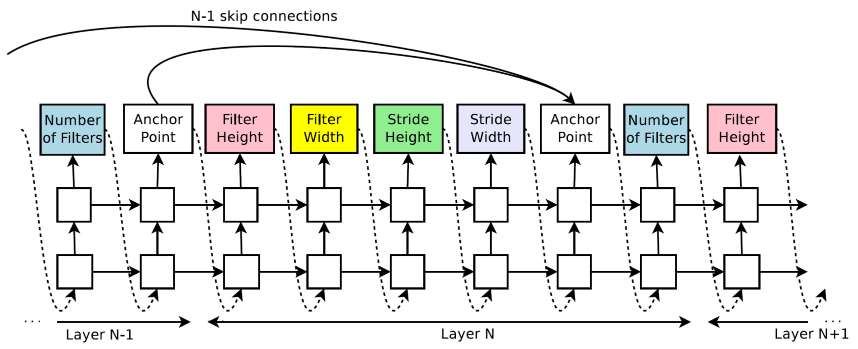
2.1 Reinforcement Learning

For **NAS with RL**, as an example in [1], on image recognition tasks, the controller recursively generates architectual hyperparameters for each convolutional layer, as shown in Figure 1(a). On language modeling tasks, the controller searches for a recurrent cell which is binary tree-structured, where on each time step t, the controller predicts the output tree node h_t and two activation functions to perform on the current input x_t and previous hidden state h_{t-1}, as shown in Figure 1(b).

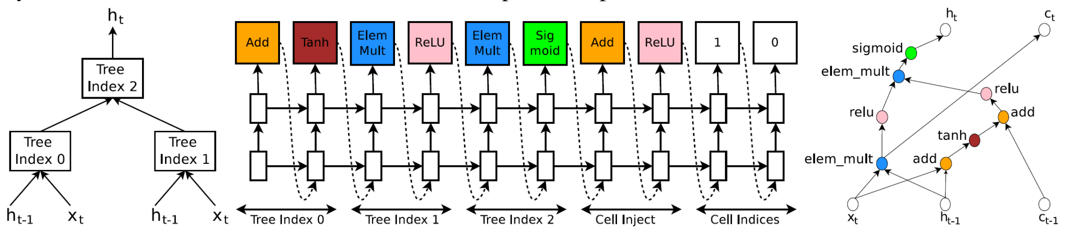
Extending from RL based NAS, **NASNet** [2] designs a cell search space that the best architecture found on small image classification datasets (such as CIFAR-10) could easily scale to larger, higher-resolution image datasets. The controller RNN searches for convolutional layers on CIFAR-10 and stacks the resulting layers to perform classification on ImageNet. The controller searches for two types of convolutional cells: $Normal\ Cell$ that keeps spatial dimensions the same, and $Reduction\ Cell$ that reduces the spatial dimensions by half. Therefore, the model can deal with images with different resolutions. For each cell, the controller determines two input hidden states, two operations applied to the hidden states separately, and a combination method to combine the two output, as shown in Figure 2(a). Then the two types of cells are placed in predefined CIFAR10 and ImageNet architectures as shown in Figure 2(b).

Another extension from RL based NAS is **ENAS** [3], which views the whole search space as a large directed acyclic graph (DAG). The nodes represent operations and the edges represent the flow of information. The controller RNN is trained using policy gradient to find a subgraph that maximizes the reward (c /validation_perplexity) on a validation set. On each node, the controller samples a previous node index and an activation function to apply on the previous node, as shown in Figure 3. Since each child model is a subgraph in the DAG, every model shares parameters during training. This allows models to train on pretrained weights rather than from scratch.

Other RL based works further reduce the computational complexity while increasing the accuracy on image classification. **PNAS** [4] extends from NASNet, but instead of constructing 5-block convolutional cells all at once, it proposed sequential model-based optimization (SMBO) that searches for convolutional cells in order of increasing number of blocks.



(a) The controller RNN model for recursively finding hyperparameters for each layer. Each output is carried out by a softmax classifier and fed into the next time step as the input.



(b) An example tree structure for a recurrent cell. *Left:* an example tree that has two leaf nodes and one internal node. The controller needs to predict operations for `Tree Index 0,1,2`. *Middle:* The controller RNN model that recursively finds operations for each node. "Cell Inject" and "Cell Indices" specify how to connect c_t and c_{t-1} to outputs of tree nodes. In this example, the controller predicts 1 for c_t, so we set c_t as output of `Tree Index 1`. The controller predict 0 for c_{t-1}, along with operations Add and $ReLU$, so $a_0^{new} = \text{ReLU}(a_0 + c_{t-1})$. *Right:* the computation graph of the constructed recurrent cell.

Figure 1: NAS with RL searching rules and architectures.

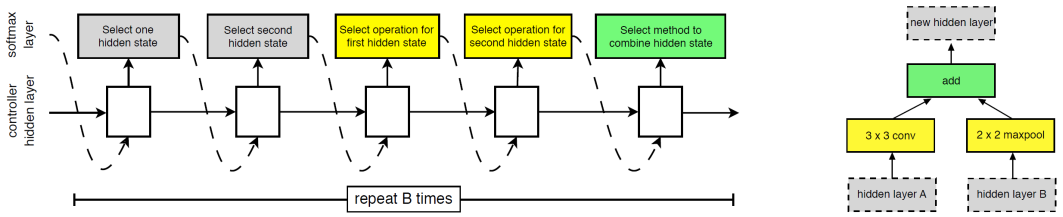
2.2 Evolutionary Algorithms

For **NAS with EA (Hier-EA)**, in [6], models are evolved to be more complex. The search space is a single directed acyclic graph (DAG), where each node corresponds to a feature map, and each edge (i, j) corresponds to a primitive operation such as convolution and pooling that transforms node i to node j. A model can be represented as $(G, \mathbf{o})$. $\mathbf{o}$ is a set of primitive operations. G is adjacency matrix, where $G_{ij} = k$ means that k-th operation is placed between nodes i and j. [6] also proposed hierarchical representations for neural network architectures that more complex models are composed of simpler building blocks. Details are shown in Figure 4.

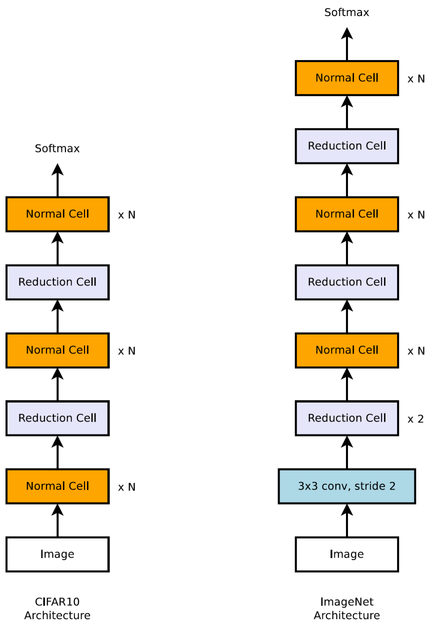
Previous methods use non-$aging\ evolution$ that during each round of evolution, the best-performance model mutates to get a child and the worst-performance model is dropped. One drawback for non-$aging\ evolution$ is zooming in on good early-generated models and explore less the later-generated/younger models. To pay more attention to younger models and explore the search space more, **AmoebaNet-A** [7] proposes an $aging\ evolution$ that at every round of evolution, the oldest model is discarded. During training, the algorithm keeps a population of P models. Then for every cycle it evolves, S candidate models are sampled to perform the tournament selection that the model with the best performance is chosen to be the $parent$ model and randomly mutates its architecture to get a $child$ model. The $child$ model is added to the population and the first-generated model in the population is removed.

2.3 Gradient-based

Although RL and EA based NAS can achieve satisfying performance, the computation cost is as high as hundreds or even thousands of GPU days. To reduce the cost, **Differentiable Architecture Search (DARTS)** [9] proposes a continuous architecture search space to search for a convolution



(a) The controller RNN model for recursively finding B blocks of a convolutional cell. *Left:* one block of a convolutional cell includes two hidden states as the inputs, two operations applied on the two hidden states separately, the method to combine the outputs of the hidden states. *Right:* an example of one block. The first hidden state is `hidden layer A`, the second is `hidden layer B`. The operation for first hidden state is `3x3 conv`, for second is `2x2 maxpool`. The combination method is `add`.



(b) Placement of $Normal\ Cells$ and $Reduction\ Cells$ in CIFAR10 and ImageNet architectures. The number of $Normal\ Cells$ N varies in experiments.

Figure 2: NASNet searching rules and architectures.

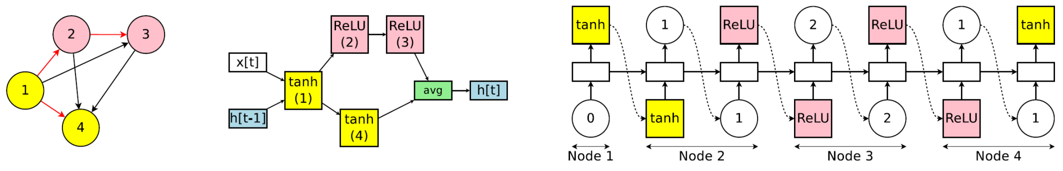


Figure 3: An example of constructing a recurrent cell. *Left:* The computational DAG of the recurrent cell. Red edges represent information flow in the graph. Node 1 is the input. Node 3 and 4 are the outputs. *Middle:* the computation graph after each node has been sampled an operation. *Right:* the controller RNN model for recursively predicting the operation and which index to connect for each node. In this example, for `Node 1`, the controller predicts $tanh$; for `Node 2`, it predicts $ReLU$ and `Node 1` to connect to. After the whole computational DAG has been predicted, the output results are averaged as the final output.

or recurrent cell. The search space is a DAG, where each node represents a feature map, and each directed edge (i, j) represents some operation $o(\cdot) \in \mathcal{O}$ that transforms node i to node j. During training, the information propagated from i to j is a weighted sum over all operations, namely, $f(x_i) = \sum_{o \in \mathcal{O}} \frac{exp\{\alpha_o^{(i,j)}\}}{\sum_{o' \in \mathcal{O}} exp\{\alpha_{o'}^{(i,j)}\}} \cdot o(\mathbf{x}_i)$, where $\alpha_o^{(i,j)}$ is the weight for operation $o(\cdot)$ on edge (i, j).

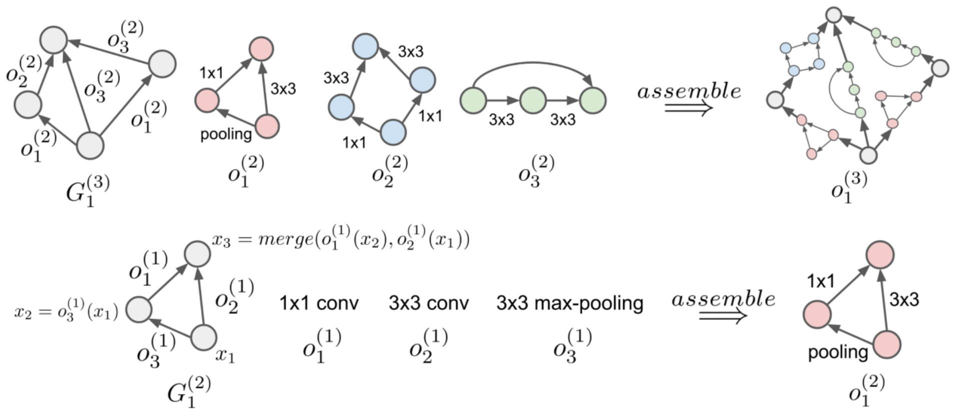


Figure 4: Hierarchical representations for neural network architectures. *Bottom row:* three level-1 primitive operations $\{o_1^{(1)}, o_2^{(1)}, o_3^{(1)}\}$ and three feature maps $\{x_1, x_2, x_3\}$ are assembled into level-2 motif $o_1^{(2)}$. *Top row:* three level-2 motifs $\{o_1^{(2)}, o_2^{(2)}, o_3^{(2)}\}$ are assembled according to adjacency matrix $G_1^{(3)}$.

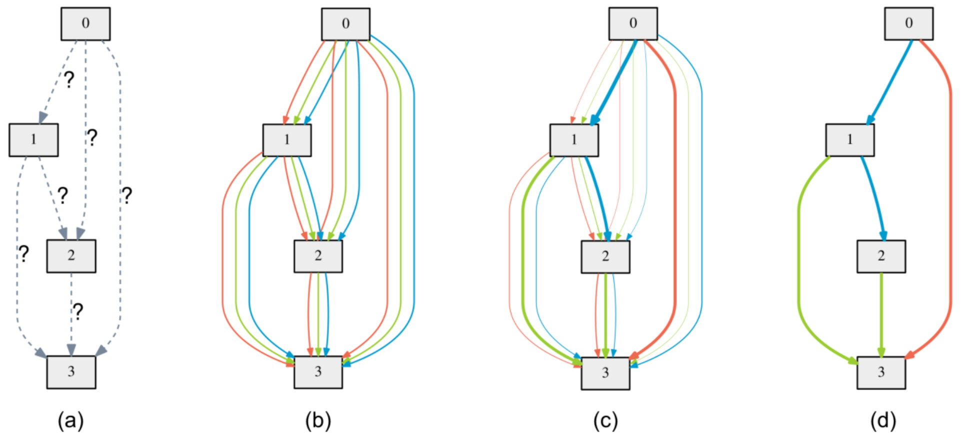


Figure 5: Learning process in the continuous relaxation of the search space in DARTS. (a) Initial weights for operations on each edge are unknown. (b) Continuously relax the search space by placing softmax probabilities of all operations on each edge. (c) Perform the bilevel optimization to update the softmax probabilities. (d) Choose an operation with the largest weight for each edge and derive the final architecture.

The controller's goal is to find α that minimizes the validation loss, where the child network weights w minimize the training loss. After training, the operation with the largest weight on each edge is used. The training process is briefly visualized in Figure 5. The derived cells are then used to construct larger architectures.

Despite of the sophisticated design of DARTS that converts the discrete selection of operations into mixture weights of operations, it still suffers from high memory and computation cost since each edge has to keep all $|\mathcal{O}|$ operations. This high memory usage further results in small batch size, making the training unstable. Another drawback of DARTS is the *depth gap* between the training network and the evaluation network that training uses shallower neural networks, so evaluation reports lower accuracy. **P-DARTS** [10] deals with the *depth gap* issue by progressively increasing the depth of searched networks while dropping the number of candidate operations during training (shown in Figure 6). This process stops when the depth is close to that used in evaluation. **PC-DARTS** [11] reduces the memory cost by sampling a subset of channels on each node to perform mixture operations, while other channels are directly copied to the output (shown in Figure 7).

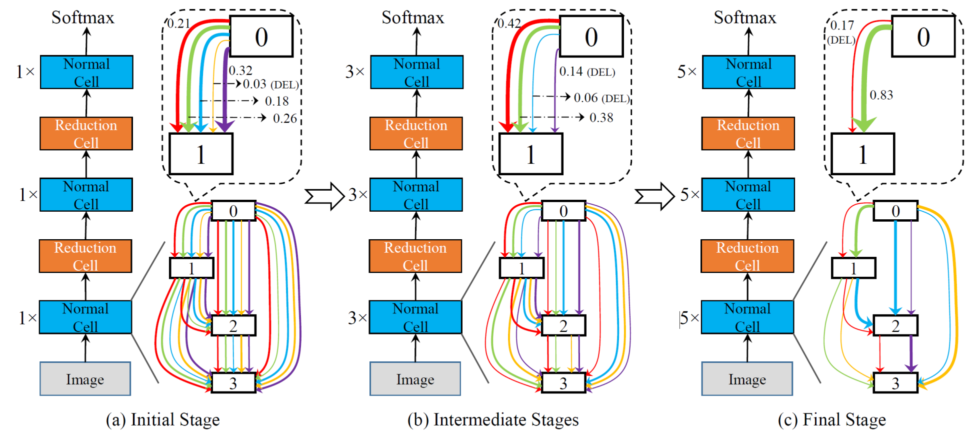
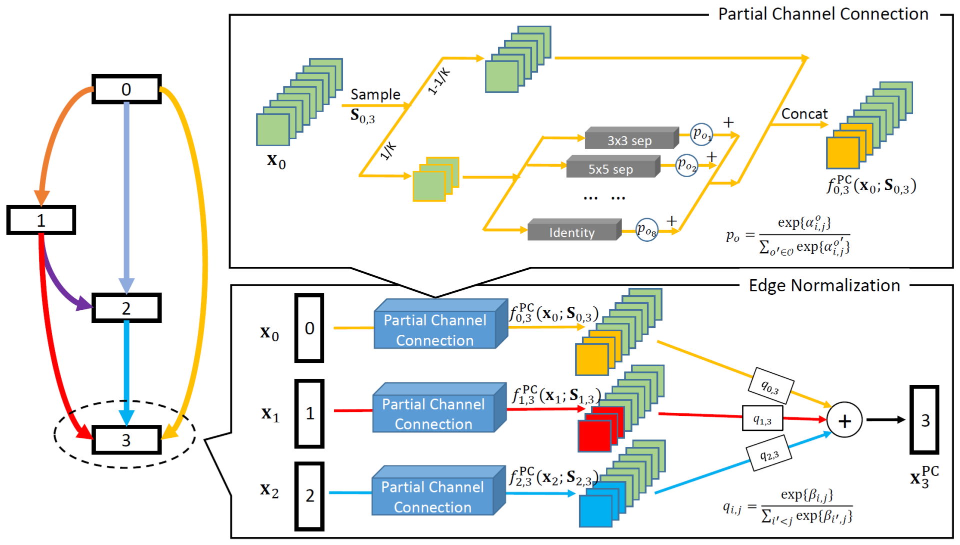


Figure 6: The overall training process of P-DARTS. For simplicity, only the computation graph of the normal cells are displayed. The depth of the search network increases from 5 at the initial stage to 11 at the intermediate stage and 17 at the final stage, while the number of candidate operations (the colored connections between each pair of nodes) reduces from 5 at the initial stage to 4 at the intermediate stage and 2 at the final stage. After each stage, the candidate operation on each edge with the lowest score is dropped.



Partial Channel Connection
Sample
$\mathbf{S}_{0,3}$
$\mathbf{x}_0$
1-1/K
1/K
3x3 sep
5x5 sep
Identity
Concat
$f^{\mathrm{PC}}_{0,3}(\mathbf{x}_0;\mathbf{S}_{0,3})$
$p_o = \frac{\exp\{\alpha^o_{i,j}\}}{\sum_{o'\in\mathcal{O}}\exp\{\alpha^{o'}_{i,j}\}}$
Edge Normalization
Partial Channel Connection
$f^{\mathrm{PC}}_{1,3}(\mathbf{x}_1;\mathbf{S}_{1,3})$
$f^{\mathrm{PC}}_{2,3}(\mathbf{x}_2;\mathbf{S}_{2,3})$
$q_{0,3}$
$q_{1,3}$
$q_{2,3}$
$\mathbf{x}^{\mathrm{PC}}_3$
$q_{i,j} = \frac{\exp\{\beta_{i,j}\}}{\sum_{i'<j}\exp\{\beta_{i',j}\}}$

Figure 7: Illustration of the information propagation in a cell in PC-DARTS. As an example, the figure only displays the information flows from all other nodes to node #3. For node #0 $\mathbf{x}_0$, the mask $\mathbf{S}_{0,3}$ samples $1/K$ of its channels to apply the mixed operations. The rest $(1-1/K)$ channels are directly copied to the output. After doing such operation for all other nodes, the resulted output channels go through an edge normalization to reduce the bias towards weight-free operations (*e.g.*, *skip-connection*), and are concatenated depth-wise to get the final output.

Unlike DARTS that represents the continuous space by mixture weights, **Neural Architecture Optimization (NAONet)** [15] encodes the neural architecture into a continuous space. NAONet has an LSTM encoder that takes the string x describing an architecture as the input, and the encoder's hidden states from all time steps $e_x = \{h_1, h_2, ..., h_T\}$ are viewed as the continuous representation of the architecture. A performance predictor feeds the mean of the continuous representation $\bar{e}_x = \frac{1}{T}\sum_t^T h_t$ into a one-layer feed-forward network to predict the performance. Gradient ascent optimization is applied to maximize the performance and update e_x. A single-layer LSTM decoder with attention mechanism takes e_x as the input, and outputs decoded architecture string x'. The process visualization can be found in Figure 8.

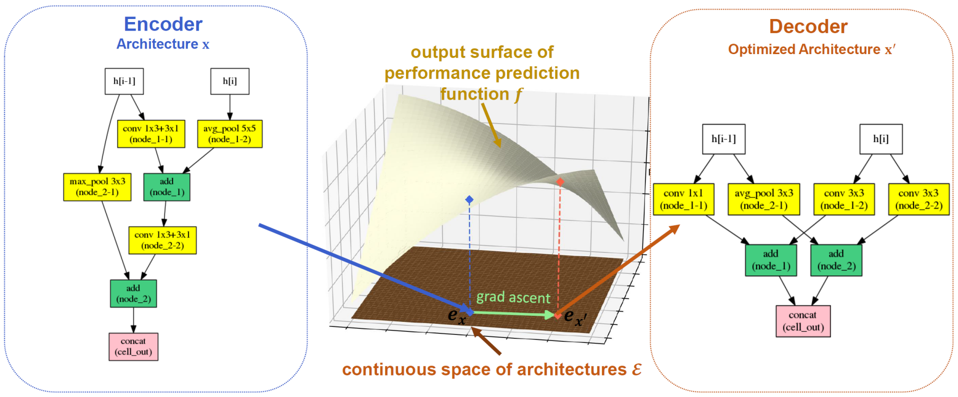


Figure 8: The general framework for NAONet. The encoder encodes the original architecture x into a continuous representation (a.k.a embedding) e_x through the blue arrow. Then by using gradient ascent to optimize the performance predictor function f, e_x is updated to $e_{x'}$ through the green arrow. Finally, $e_{x'}$ is decoded by the decoder to get the optimized architecture x' through the brown arrow.

3 Results

Table 1 shows the test error, number of parameters, search cost and search method for state-of-the-art models on CIFAR-10 and ImageNet. Models are trained on Tesla V100 GPUs.

Architecture	CIFAR-10			ImageNet				Search Method
	Test Err. (%)	Params (M)	Search Cost (GPU-days)	Test Err. (%) top-1	top-5	Params (M)	Search Cost (GPU-days)	
DenseNet-BC [16]	3.46	25.6	-	-	-	-	-	manual
Inception-v1 [17]	-	-	-	30.2	10.1	6.6	-	manual
MobileNet [18]	-	-	-	29.4	10.5	4.2	-	manual
ShuffleNet 2× (v1) [19]	-	-	-	26.4	10.2	∼5	-	manual
ShuffleNet 2× (v2) [20]	-	-	-	25.1	-	∼5	-	manual
NASNet-A + cutout [2]	2.65	3.3	1800	26	8.4	5.3	1800	RL
ENAS + cutout [3]	2.89	4.6	0.5	-	-	-	-	RL
PNAS [4]	3.41±0.09	3.2	225	25.8	8.1	5.1	225	SMBO
Hier-EA [6]	3.75±0.12	15.7	300	-	-	-	-	evolution
AmoebaNet-A [7]	3.34±0.06	3.2	3150	25.5	8.0	5.1	3150	evolution
AmoebaNet-B [7]	2.55±0.05	2.8	3150	26.0	8.5	5.3	3150	evolution
NAONet (searched on CIFAR10) [15]	3.53	3.1	0.4	25.7	8.2	11.35	-	gradient
SNAS (mild) [12]	2.98	2.9	1.5	27.3	9.2	4.3	1.5	gradient
DARTS (1st order) [9]	3.00±0.14	3.3	0.4	-	-	-	-	gradient
DARTS (2nd order) [9]	2.76±0.09	3.3	1	26.7	8.7	4.7	4.0	gradient
P-DARTS (searched on CIFAR10) [10]	2.50	3.4	0.3	24.4	7.4	4.9	0.3	gradient
PC-DARTS (searched on CIFAR10) [11]	2.57±0.07	3.6	0.1	25.1	7.8	5.3	0.1	gradient
PC-DARTS (searched on ImageNet) [11]	-	-	-	24.2	7.3	5.3	3.8	gradient

Table 1: Comparison with state-of-the-art architectures on CIFAR-10 and ImageNet.

References

[1] Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. *ICLR*, 2017.

[2] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V. Le. Learning transferable architectures for scalable image recognition. *CVPR*, 2018.

[3] Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[4] 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.

[5] Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc Le, and Alex Kurakin. Large-scale evolution of image classifiers. *ICML*, 2017.

[6] Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. Hierarchical representations for efficient architecture search. *ICLR*, 2018.

[7] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *AAAI*, 2019.

[8] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Efficient multi-objective neural architecture search via lamarckian evolution. *ICLR*, 2019.

[9] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *ICLR*, 2019.

[10] Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. *ICCV*, 2019.

[11] Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. Pc-darts: Partial channel connections for memory-efficient architecture search. *ICLR*, 2020.

[12] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: Stochastic neural architecture search. *ICLR*, 2019.

[13] Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. Smash: One-shot model architecture search through hypernetworks. *ICLR*, 2018.

[14] Jiemin Fang, Yuzhu Sun, Qian Zhang, Yuan Li, Wenyu Liu, and Xinggang Wang. Densely connected search space for more flexible neural architecture search. *CVPR*, 2020.

[15] Renqian Luo, Fei Tian, Tao Qin, Enhong Chen, and Tie-Yan Liu. Neural architecture optimization. *NIPS*, 2018.

[16] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. *CVPR*, 2017.

[17] 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.

[18] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[19] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CVPR*, 2018.

[20] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. *ECCV*, 2018.