Multinomial Distribution Learning for Effective Neural Architecture Search

Xiawu Zheng [1,2] , Rongrong Ji[1,2*], Lang Tang[1,2], Baochang Zhang[3], Jianzhuang Liu[4], Qi Tian[5]

[1]Media Analytics and Computing Lab, Department of Artificial Intelligence,
School of Informatics, Xiamen University, 361005, China,
[2]Peng Cheng Laboratory, Shenzhen, China, [3]Beihang University, China,
[4]Huawei Noah's Ark Lab, [5]Department of Computer Science, University of Texas at San Antonio

{zhengxiawu,langt}@stu.xmu.edu.cn, rrji@xmu.edu.cn

bczhang@buaa.edu.cn, liu.jianzhuang@huawei.com, qitian@cs.utsa.edu

Abstract

Architectures obtained by Neural Architecture Search (NAS) have achieved highly competitive performance in various computer vision tasks. However, the prohibitive computation demand of forward-backward propagation in deep neural networks and searching algorithms makes it difficult to apply NAS in practice. In this paper, we propose a Multinomial Distribution Learning for extremely effective NAS, which considers the search space as a joint multinomial distribution, i.e., the operation between two nodes is sampled from this distribution, and the optimal network structure is obtained by the operations with the most likely probability in this distribution. Therefore, NAS can be transformed to a multinomial distribution learning problem, i.e., the distribution is optimized to have high expectation of the performance. Besides, a hypothesis that the performance ranking is consistent in every training epoch is proposed and demonstrated to further accelerate the learning process. Experiments on CIFAR-10 and ImageNet demonstrate the effectiveness of our method. On CIFAR-10, the structure searched by our method achieves 2.55% test error, while being 6.0× (only 4 GPU hours on GTX1080Ti) faster compared with state-of-the-art NAS algorithms. On ImageNet, our model achieves 75.2% top-1 accuracy under MobileNet settings (MobileNet V1/V2), while being 1.2× faster with measured GPU latency. Test code with pre-trained models are available at https: //github.com/tanglang96/MDENAS

1. Introduction

Given a dataset, Neural architecture search (NAS) aims to discover high-performance convolution architectures with a searching algorithm in a tremendous search space.

---

*Corresponding Author.

NAS has achieved much success in automated architecture engineering for various deep learning tasks, such as image classification [19, 34, 32], language modeling [20, 33] and semantic segmentation [18, 6]. As mentioned in [9], NAS methods consist of three parts: search space, search strategy, and performance estimation. A conventional NAS algorithm samples a specific convolutional architecture by a search strategy and estimates the performance, which can be regarded as an objective to update the search strategy. Despite the remarkable progress, conventional NAS methods are prohibited by intensive computation and memory costs. For example, the reinforcement learning (RL) method in [34] trains and evaluates more than 20,000 neural networks across 500 GPUs over 4 days. Recent work in [20] improves the scalability by formulating the task in a differentiable manner where the search space is relaxed to a continuous space, so that the architecture can be optimized with the performance on a validation set by gradient descent. However, differentiable NAS still suffers from the issued of high GPU memory consumption, which grows linearly with the size of the candidate search set.

Indeed, most NAS methods [34, 18] perform the performance estimation using standard training and validation over each searched architecture, typically, the architecture has to be trained to converge to get the final evaluation on validation set, which is computationally expensive and limits the search exploration. However, if the evaluation of different architectures can be ranked within a few epochs, why do we need to estimate the performance after the neural network converges? Consider an example in Fig. 1, we randomly sample different architectures (LeNet [17], AlexNet [16], ResNet-18 [11] and DenseNet [14]) with different layers, the performance ranking in the training and testing is consistent (*i.e*, the performance ranking is ResNet-18 > DenseNet-BC > AlexNet > LeNet on different networks and training epochs). Based on this observation, we state the following hypothesis for performance ranking:
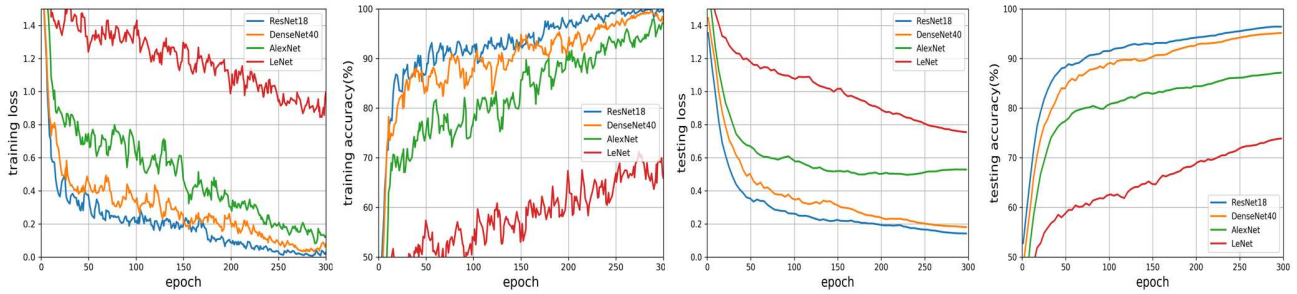
Figure 1. We randomly choose widely used LeNet [17], AlexNet [15], ResNet-18[11] and DenseNet-BC($k = 40$) [14] to illustrate the proposed *Performance Ranking Hypothesis*. The training and testing are conducted on CIFAR-10. We report the top1 error and loss learning curves on both training and testing set. As we can see in the figure, the *ranking* of the test loss and accuracy keeps consistent in every training epoch, *i.e.*, a good architecture tends to have better performance in the whole training process.

Performance Ranking Hypothesis. *If Cell A has higher validation performance than Cell B on a specific network and a training epoch, Cell A tends to be better than Cell B on different networks after the trainings of these netwoks converge.*

Here, a cell is a fully convolutional directed acyclic graph (DAG) that maps an input tensor to an output tensor, and the final network is obtained through stacking different numbers of cells, the details of which are described in Sec. 3.

The hypothesis illustrates a simple yet important rule in neural architecture search. The comparison of different architectures can be finished at early stages, as the ranking of different architectures is sufficient, whereas the final results are unnecessary and time-consuming. Based on this hypothesis, we propose a simple yet effective solution to neural architecture search, termed as Multinomial distribution for efficient Neural Architecture Search (MdeNAS), which directly formulates NAS as a distribution learning process. Specifically, the probabilities of operation candidates between two nodes are initialized equally, which can be considered as a multinomial distribution. In the learning procedure, the parameters of the distribution are updated through the current performance in every epoch, such that the probability of a bad operation is transferred to better operations. With this search strategy, MdeNAS is able to fast and effectively discover high-performance architectures with complex graph topologies within a rich search space.

In our experiments, the convolutional cells designed by MdeNAS achieve strong quantitative results. The searched model reaches 2.55% test error on CIFAR-10 with less parameters. On ImageNet, our model achieves 75.2% top-1 accuracy under MobileNet settings (MobileNet V1/V2 [12, 26]), while being 1.2× faster with measured GPU latency. The contributions of this paper are summarized as follows:

- We introduce a novel algorithm for network architec-

ture search, which is applicable to various large-scale datasets as the memory and computation costs are similar to common neural network training.

- We propose a performance ranking hypothesis, which can be incorporated into the existing NAS algorithms to speed up its search.

- The proposed method achieves remarkable search efficiency, *e.g.*, 2.55% test error on CIFAR-10 in 4 hours with 1 GTX1080Ti (6.0× faster compared with state-of-the-art algorithms), which is attributed to using our distribution learning that is entirely different from RL-based [2, 34] methods and differentiable methods [20, 29].

2. Related Work

As first proposed in [33, 34], automatic neural network search in a predefined architecture space has received significant attention in the last few years. To this end, many search algorithms have been proposed to find optimal architectures using specific search strategies. Since most hand-crafted CNNs are built by stacked *reduction* (*i.e.*, the spatial dimension of the input is reduced) and *norm* (*i.e.* the spatial dimensionality of the input is preserved) cells [14, 11, 13], the works in [33, 34] proposed to search networks under the same setting to reduce the search space. The works in [33, 34, 2] use reinforcement learning as a meta-controller, to explore the architecture search space. The works in [33, 34] employ a recurrent neural network (RNN) as the policy to sequentially sample a string encoding a specific neural architecture. The policy network can be trained with the policy gradient algorithm or the proximal policy optimization. The works in [3, 4, 19] regard the architecture search space as a tree structure for network transformation, *i.e.*, the network is generated by a farther network with some predefined operations, which reduces the search space and
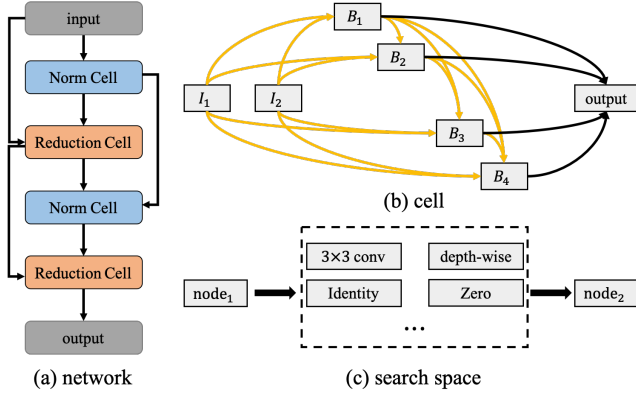
Figure 2. Searching networks with different scales. (a) A network consists of stacked cells, and each cell takes the output of two previous cells as input. (b) A cell contains 7 nodes, two input nodes I_1 and I_2, four intermediate nodes B_1, B_2, B_3, B_4 that apply sampled operations on the input nodes and upper nodes, and an output node that concatenates the outputs of the four intermediate nodes. (c) The edge between two nodes denotes a possible operation according to a multinomial distribution in the search space.

speeds up the search. An alternative to RL-based methods is the *evolutionary* approach, which optimizes the neural architecture by evolutionary algorithms [28, 24].

However, the above architecture search algorithms are still computation-intensive. Therefore some recent works are proposed to accelerate NAS by *one-shot* setting, where the network is sampled by a hyper representation graph, and the search process can be accelerated by parameter sharing [23]. For instance, DARTS [20] optimizes the weights within two node in the hyper-graph jointly with a continuous relaxation. Therefore, the parameters can be updated via standard gradient descend. However, *one-shot* methods suffer from the issue of large GPU memory consumption. To solve this problem, ProxylessNAS [5] explores the search space without a specific agent with path binarization [7]. However, since the search procedure of ProxylessNAS is still within the framework of *one-shot* methods, it may have the same complexity, *i.e.,* the benefit gained in ProxylessNAS is a *trade-off* between exploration and exploitation. That is to say, more epochs are needed in the search procedure. Moreover, the search algorithm in [5] is similar to previous work, either differential or RL based methods [20, 34].

Different from the previous methods, we encode the path/operation selection as a distribution sampling, and achieve the optimization of the controller/proxy via distribution learning. Our learning process further integrates the proposed hypothesis to estimate the merit of each operation/path, which achieves an extremely efficient NAS search.

3. Architecture Search Space

In this section, we describe the architecture search space and the method to build the network. We follow the same settings as in previous NAS works [20, 19, 34] to keep the consistency. As illustrated in Fig. 2, the network is defined in different *scales*: network, cell, and node.

3.1. Node

Nodes are the fundamental elements that compose cells. Each node x^i is a specific tensor (*e.g.*, a feature map in convolutional neural networks) and each directed edge (i, j) denotes an operation $o^{(i,j)}$ sampled from the operation search space to transform node x^i to another node x^j, as illustrated in Fig. 2(c). There are three types of nodes in a cell: input node x_I, intermediate node x_B, and output node x_O. Each cell takes the previous output tensor as an input node, and generates the intermediate nodes x_B^i by applying sampled operations $o^{(i,j)}$ to the previous nodes (x_I and $x_B^j, j \in [1, i)$). The concatenation of all intermediate nodes is regarded as the final output node.

Following [20] set of possible operations, denoted as $\mathcal{O}$, consists of the following 8 operations: (1) 3×3 max pooling. (2) no connection (zero). (3) 3×3 average pooling. (4) skip connection (identity). (5) 3×3 dilated convolution with rate 2. (6) 5×5 dilated convolution with rate 2. (7) 3×3 depth-wise separable convolution. (8) 5×5 depth-wise separable convolution.

We simply employ element-wise addition at the input of a node with multiple operations (edges). For example, in Fig. 2(b), B_2 has three operations, the results of which are added element-wise and then considered as B_2.

3.2. Cell

A cell is defined as a tiny convolutional network mapping an $H \times W \times F$ tensor to another $H' \times W' \times F'$. There are two types of cells, norm cell and reduction cell. A norm cell uses the operations with stride 1, and therefore $H' = H$ and $W' = W$. A reduction cell uses the operations with stride 2, so $H' = H/2$ and $W' = W/2$. For the numbers of filters F and F', a common heuristic in most human designed convolutional neural networks [11, 14, 16, 27, 10, 31] is to double F whenever the spatial feature map is halved. Therefore, $F' = F$ for stride 1, and $F' = 2F$ for stride 2.

As illustrated in Fig. 2(b), the cell is represented by a DAG with 7 nodes (two input nodes I_1 and I_2, four intermediate nodes B_1, B_2, B_3, B_4 that apply sampled operations on the input and upper nodes, and an output node that concatenates the intermediate nodes). The edge between two nodes denote a possible operation according to a multinomial distribution $p_{(node_1, node_2)}$ in the search space. In training, the input of an intermediate node is obtained by element-wise addition when it has multiple edges (opera-
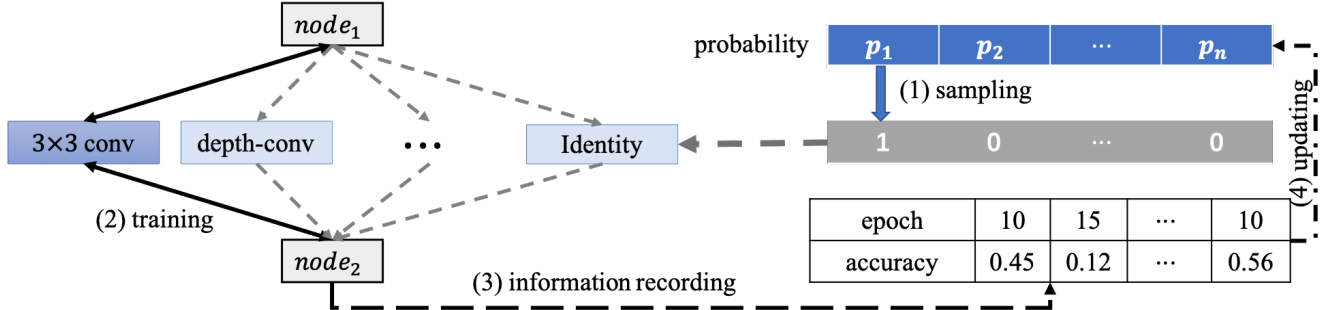
Figure 3. The overall search algorithm: (1) Sample one operation in the search space according to the corresponding multinomial distribution with parameters θ. (2) Train the generated network with one forward and backward propagation. (3) Test the network on the validation set and record the feedback (epoch and accuracy). (4) Update the distribution parameters according to the proposed distribution learning algorithm. In the right table, the epoch number of operation 1 is 10, which means that this operation is selected 10 times among all the epochs.

tions). In testing, we select the top K probabilities to generate the final cells. Therefore, the size of the whole search space is $2 \times 8^{|\mathcal{E}_\mathcal{N}|}$, where $\mathcal{E}_\mathcal{N}$ is the set of possible edges with N intermediate nodes. In our case with $N = 4$, the total number of cell structures is $2 \times 8^{2+3+4+5} = 2 \times 8^{14}$, which is an extremely large space to search, and thus requires efficient optimization methods.

3.3. Network

As illustrated in Fig. 2(a), a network consists of a predefined number of stacked cells, which can be either norm cells or reduction cells each taking the output of two previous cells as input. At the top of the network, global average pooling followed by a softmax layer is used for final output. Based on the *Performance Ranking Hypothesis*, we train a small (*e.g.*, 6 layers) stacked model on the relevant dataset to search for norm and reduction cells, and then generate a deeper network (*e.g.*, 20 layers) for evaluation. The overall CNN construction process and the search space are identical to [20]. But note that our search algorithm is different.

4. Methodology

In this section, our NAS method is presented. We first describe how to sample the network mentioned in Sec. 3 to reduce GPU memory consumption during training. Then, we present a multinomial distribution learning to effectively optimize the distribution parameters using the proposed hypothesis.

4.1. Sampling

As mentioned in Sec. 3.1, the diversity of network structures is generated by different selections of M possible paths (in this work, $M = 8$) for every two nodes. Here we initialize the probabilities of these paths as $p_i = \frac{1}{M}$ in the beginning for exploration. In the sampling stage, we follow the

work in [5] and transform the M real-valued probabilities $\{p_i\}$ with binary gates $\{g_i\}$:

$$g = \begin{cases} \underbrace{[1, 0, ..., 0]}_{M} \text{ with probability } p_1 \\ ... \\ \underbrace{[0, 0, ..., 1]}_{M} \text{ with probability } p_M \end{cases} \quad (1)$$

The final operation between nodes i and j is obtained by:

$$o^{(i,j)} = o^{(i,j)} * g = \begin{cases} o_1 \text{ with probability } p_1 \\ ... \\ o_M \text{ with probability } p_M. \end{cases} \quad (2)$$

As illustrated in the previous equations, we sample only one operation at run-time, which effectively reduces the memory cost compared with [20].

4.2. Multinomial Distribution Learning

Previous NAS methods are time and memory consuming. The use of reinforcement learning further prohibits the methods with the delay reward in network training, *i.e.,* the evaluation of a structure is usually finished after the network training converges. On the other hand, as mentioned in Sec. 1, according to the *Performance Ranking Hypothesis*, we can perform the evaluation of a cell when training the network. As illustrated in Fig. 3, the training epochs and accuracy for every operation in the search space are recorded. Operations A is better than B, if operation A has fewer training epochs and higher accuracy.

Formally, for a specific edge between two nodes, we define the operation probability as p, the training epoch as $\mathcal{H}^e$, and the accuracy as $\mathcal{H}^a$, each of which is a real-valued column vector of length $M = 8$. To clearly illustrate our learn-

ing method, we further define the differential of epoch as:

$$\Delta \mathcal{H}^e = \begin{bmatrix} (\vec{1} \times \mathcal{H}_1^e - \mathcal{H}^e)^T \\ ... \\ (\vec{1} \times \mathcal{H}_M^e - \mathcal{H}^e)^T \end{bmatrix}, \qquad (3)$$

and the differential of accuracy as:

$$\Delta \mathcal{H}^a = \begin{bmatrix} (\vec{1} \times \mathcal{H}_1^a - \mathcal{H}^a)^T \\ ... \\ (\vec{1} \times \mathcal{H}_M^a - \mathcal{H}^a)^T \end{bmatrix}, \qquad (4)$$

where $\vec{1}$ is a column vector with length 8 and all its elements being 1, $\Delta \mathcal{H}^e$ and $\Delta \mathcal{H}^a$ are 8×8 matrices, where $\Delta \mathcal{H}_{i,j}^e = \mathcal{H}_i^e - \mathcal{H}_j^e$, $\Delta \mathcal{H}_{i,j}^a = \mathcal{H}_i^a - \mathcal{H}_j^a$. After one epoch training, the corresponding variables $\mathcal{H}^e$, $\mathcal{H}^a$, $\Delta \mathcal{H}^e$ and $\Delta \mathcal{H}^a$ are calculated by the evaluation results. The parameters of the multinomial distribution can be updated through:

$$p_i \leftarrow p_i + \alpha * \left(\sum_j \mathbb{1}(\Delta \mathcal{H}_{i,j}^e < 0, \Delta \mathcal{H}_{i,j}^a > 0) - \right.$$
$$\left. \sum_j \mathbb{1}(\Delta \mathcal{H}_{i,j}^e > 0, \Delta \mathcal{H}_{i,j}^a < 0) \right), \qquad (5)$$

where α is a hyper-parameter, and $\mathbb{1}$ denotes as the indicator function that equals to one if its condition is true.

As we can see in Eq. 5, the probability of a specific operation i is enhanced with fewer epochs ($\Delta \mathcal{H}_{i,j}^e < 0$) and higher performance ($\Delta \mathcal{H}_{i,j}^a > 0$). At the same time, the probability is reduced with more epochs ($\Delta \mathcal{H}_{i,j}^e > 0$) and lower performance ($\Delta \mathcal{H}_{i,j}^a < 0$). Since Eq. 5 is applied after every training epoch, the probability in the search space can be effectively converge and stabilize after a few epochs. Together with the proposed performance ranking hypothesis (demonstrated latter in Section 5), our multinomial distribution learning algorithm for NAS is extremely efficient, and achieves a better performance compared with other state-of-the-art methods under the same settings. Considering the performance ranking is consisted of different layers according to the hypothesis, to further improve the search efficiency, we replace the search network in [20] with another shallower one (only 6 layers), which takes only 4 GPU hours of searching on CIFAR-10.

To generate the final network, we first select the operations with highest probabilities in all edges. For nodes with multi-input, we employ element-wise addition with top K probabilities. The final network consists of a predefined number of stacked cells, using either norm or reduction cells. Our multinomial distribution learning algorithm is presented in Alg. 1.

5. Experiment

In this section, we first conduct some experiments on the CIFAR-10 to demonstrate the proposed hypothesis. Then,

---

Algorithm 1: Multinomial Distribution Learning

Input: Training data: $\mathcal{D}_t$; Validation data: $\mathcal{D}_v$; CNN model: $\mathcal{F}$

1 . **Output:** Cell operation probabilities: $\mathcal{P}$
2 . **for** t= 1,...,T **epoch do**
3   Sample the operation according to Equation 1;
4   Train the network with 1 epoch;
5   Validate the network on $\mathcal{D}_v$;
6   Caculate the differential of epoch and accuracy according to Equation 3 and Equation 4;
7   Update the probabilities with Equation 5;
8 **end**

---

we compare our method with state-of-the-art methods on both search effectiveness and efficiency on two widely-used classification datasets including CIFAR-10 and ImageNet.

5.1. Experiment Settings

5.1.1 Datasets

We follow most NAS works [20, 4, 34, 19] in their experiment datasets and evaluation metrics. In particular, we conduct most experiments on CIFAR-10 [15] which has $50,000$ training images and $10,000$ testing images. In architecture search, we randomly select $5,000$ images in the training set as the validation set to evaluate the architecture. The color image size is 32×32 with 10 classes. All the color intensities of the images are normalized to $[-1, +1]$. To further evaluate the generalization, after discovering a good cell on CIFAR-10, the architecture is transferred into a deeper network, and therefore we also conduct classification on ILSVRC 2012 ImageNet [25]. This dataset consists of $1,000$ classes, which has 1.28 million training images and $50,000$ validation images. Here we consider the *mobile* setting where the input image size is 224×224 and the number of multiply-add operations in the model is restricted to be less than 600M.

5.1.2 Implementation Details

In the search process, according to the hypothesis, the layer number is irrelevant to the evaluation of a cell structures. We therefore consider in total $L = 6$ cells in the network, where the reduction cells are inserted in the second and third layers, and 4 nodes for a cell. The network is trained for 100 epochs, with a batch size as 512 (due to the shallow network and few operation sampling), and the initial number of channels as 16. We use SGD with momentum to optimize the network weights w, with an initial learning rate of 0.025 (annealed down to zero following a cosine schedule), a momentum of 0.9, and a weight decay of 3×10^{-4}. The learning rate of the multinomial parameters is set to 0.01.
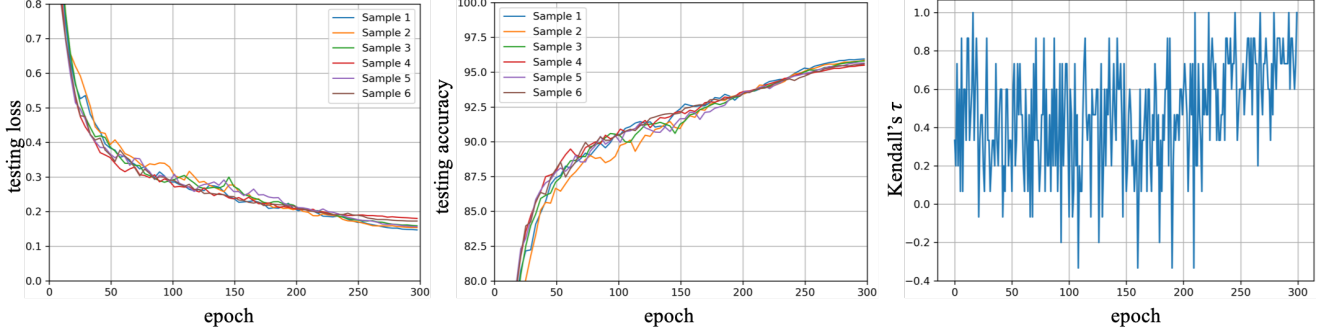
Figure 4. The test error (left), top 1 accuracy (middle), and Kendall's τ (right) of different architectures. The error and accuracy curves are entangled, since they are sampled from the same search space defined in Section 3. Therefore, we further calculate the Kendall's τ between every epoch and the final result. Note that the Kendall's $\tau > 0$ can be considered as a high value, which means more than half of the rankings are consistent.

The search takes only 4 GPU hours with only one NVIDIA GTX 1080Ti on CIFAR-10.

In the architecture evaluation step, the experimental setting is similar to [20, 34, 23]. A large network of 20 cells is trained for 600 epochs with a batch size of 96, with additional regularization such as cutout [8], and path dropout of probability of 0.3 [20]. All the experiments and models of our implementation are in PyTorch [22].

On ImageNet, we keep the same search hyperparameters as on CIFAR-10. In the training procedure, we follow previous NAS methods [20, 34, 23] with the same experimental settings. The network is trained for 250 epochs with a batch size of 512, a weight decay of 3×10^{-5}, and an initial SGD learning rate of 0.1 (decayed by a factor of 0.97 in every epoch).

5.1.3 Baselines

We compare our method with both human designed networks and other NAS networks. The manually designed networks include ResNet [11], DenseNet [14] and SENet [13]. For NAS networks, we classify them according to different search methods, such as RL (NASNet [34], ENAS [23] and Path-level NAS [4]), evolutionary algorithms (AmoebaNet [24]), Sequential Model Based Optimization (SMBO) (PNAS [19]), and gradient-based (DARTS [20]). We further compare our method under the mobile setting on ImageNet to demonstrate the generalization. The best architecture generated by our algorithm on CIFAR-10 is transferred to ImageNet, which follows the same experimental setting as the works mentioned above. Since our algorithm takes less time and memory, we also directly search on ImageNet, and compare it with another similar baseline (low computation consumption) of proxy-less NAS [5].

5.2. Evaluation of the Hypothesis

We first conduct experiments to verify the correctness of the proposed performance ranking hypothesis. To get some intuitive sense of the hypothesis, we introduce the Kendall rank correlation coefficient, *a.k.a.* Kendall's τ [1]. Given two different ranks of m items, the Kendall's τ is computed as follows:

$$\tau = \frac{P - Q}{P + Q}, \qquad (6)$$

where P is the number of pairs that are concordant (in the same order in both rankings) and Q denotes the number of pairs that are discordant (in the reverse order). $\tau \in [-1, 1]$, with 1 meaning the rankings are identical and -1 meaning a rank is in reverse of another. The probability of a pair in two ranks being consistent is $p_\tau = \frac{\tau+1}{2}$. Therefore, a $\tau = 0$ means that 50% of the pairs are concordant.

We randomly sample different network architectures in the search space, and report the loss, accuracy and Kendall's τ of different epochs on the testing set. The performance ranking in every epoch is compared with the final performance ranking of different network architectures. As illustrated in Fig. 4, the accuracy and loss are hardly distinguished due to the homogeneity of the sampled networks, *i.e.,* all the networks are generated from the same space. On the other hand, the Kendall coefficient keeps a high value ($\tau > 0, p_\tau > 0.5$) in most epochs, generally approaching 1 as the number of epochs increases. It indicates that the architecture evaluation ranking has highly convincing probabilities in every epoch and generally becomes more close to the final ranking. Note that, the mean value of Kendall's τ for each epoch is 0.474. Therefore, the hypothesis holds with a probability of 0.74. Moreover, we discover that the combination of the hypothesis with the multinomial distribution learning can enhance each other. The hypothesis guarantees the high expectation when selecting a good architecture, and the distribution learning decreases the prob-

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	Search Method
ResNet-18 [11]	3.53	11.1	-	manual
DenseNet [14]	4.77	**1.0**	-	manual
SENet [13]	4.05	11.2	-	manual
NASNet-A [34]	2.65	3.3	1800	RL
AmoebaNet-A [24]	3.34	3.2	3150	evolution
AmoebaNet-B [24]	2.55	2.8	3150	evolution
PNAS [19]	3.41	3.2	225	SMBO
ENAS [23]	2.89	4.6	0.5	RL
Path-level NAS [4]	2.49	5.7	8.3	RL
DARTS(first order) [20]	2.94	3.1	1.5	gradient-based
DARTS(second order) [20]	2.83	3.4	4	gradient-based
Random Sample [20]	3.49	3.1	-	-
MdeNAS (Ours)	**2.55**	3.61	**0.16**	**MDL**

Table 1. Test error rates of our discovered architecture, human-designed network and other NAS architectures on CIFAR-10. To be fair, we select the architectures and results with similar parameters ($< 10M$) and training conditions (same epochs and regularization).
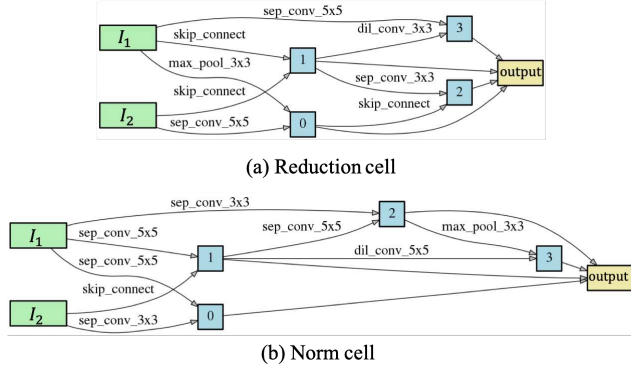

(a) Reduction cell

(b) Norm cell

Figure 5. Detailed structure of the best cells discovered on CIFAR-10. The definition of the operations on the edges is in Section 3.1. In the reduction cell (up) the stride of operations on 2 input nodes is 2, and in the norm cell (down), the stride is 1.

ability of sampling a bad architecture.

5.3. Results on CIFAR-10

We start by finding the optimal cell architecture using the proposed method. In particular, we first search neural architectures on an over-parameterized network, and then we evaluate the best architecture with a deeper network. To eliminate the random factor, the algorithm is run for several times. We find that the architecture performance is only slightly different with different times, as well as comparing to the final performance in the deeper network (<0.2), which indicates the stability of the proposed method. The best architecture is illustrated in Fig. 5.

The summarized results for convolutional architectures on CIFAR-10 are presented in Tab. 1. It is worth not-ing that the proposed method outperforms the state-of-the-art [34, 20], while with extremely less computation consumption (only 0.16 GPU days $<<$ 1,800 in [34]). Since the performance highly depends on different regularization methods (*e.g.*, cutout [8]) and layers, the network architectures are selected to compare equally under the same settings. Moreover, other works search the networks using either differential-based or black-box optimization. We attribute our superior results based on our novel way to solve the problem with distribution learning, was well as the fast learning procedure: The network architecture can be directly obtained from the distribution when the distribution converges. On the contrary, previous methods [34] evaluate architectures only when the training process is done, which is highly inefficient. Another notable phenomena observed in Tab. 1 is that, even with randomly sampling in the search space, the test error rate in [20] is only 3.49%, which is comparable with the previous methods in the same search space. We can therefore reasonable conclude that, the high performance in the previous methods is partially due to the good search space. At the same time, the proposed method quickly explores the search space and generates a better architecture. We also report the results of hand-crafted networks in Tab. 1. Clearly, our method shows a notable enhancement, which indicates its superiority in both resource consumption and test accuracy.

5.4. Results on ImageNet

We also run our algorithm on the ImageNet dataset [25]. Following existing works, we conduct two experiments with different search datasets, and test on the same dataset. As reported in Tab. 1, the previous works are time consuming on CIFAR-10, which is impractical to search on

Architecture	Accuracy (%)		Params (M)	Search Cost (GPU days)	Search Method
	Top1	Top5			
MobileNetV1 [12]	70.6	89.5	6.6	-	manual
MobileNetV2 [26]	72.0	91.0	3.4	-	manual
ShuffleNetV1 2x (V1) [30]	70.9	90.8	∼5	-	manual
ShuffleNetV2 2x (V2) [21]	73.7	-	∼5	-	manual
NASNet-A [34]	74.0	91.6	5.3	1800	RL
AmoebaNet-A [24]	74.5	92.0	5.1	3150	evolution
AmoebaNet-C [24]	75.7	92.4	6.4	3150	evolution
PNAS [19]	74.2	91.9	5.1	225	SMBO
DARTS [20]	73.1	91.0	4.9	4	gradient-based
MdeNAS (Ours)	74.5	92.1	6.1	0.16	MDL

Table 2. Comparison with state-of-the-art image classification methods on ImageNet with the mobile setting. All the NAS networks are searched on CIFAR-10, and then directly transferred to ImageNet.

Model	Top-1	Search time GPU days	GPU latency
MobileNetV2	72.0	-	6.1ms
ShuffleNetV2	72.6	-	7.3ms
Proxyless (GPU) [5]	74.8	4	5.1ms
Proxyless (CPU) [5]	74.1	4	7.4ms
MdeNAS (GPU)	75.2	2	4.9ms
MdeNAS (CPU)	74.1	2	7.1ms

Table 3. Comparison with state-of-the-art image classification on ImageNet with the mobile setting. The networks are directly searched on ImageNet with the MobileNetV2 [26] backbone.

ImageNet. Therefore, we first consider a transferable experiment on ImageNet, *i.e.*, the best architecture found on CIFAR-10 is directly transferred to ImageNet, using two initial convolution layers of stride 2 before stacking 14 cells with scale reduction (reduction cells) at 1, 2, 6 and 10. The total number of flops is decided by choosing the initial number of channels. We follow the existing NAS works to compare the performance under the *mobile* setting, where the input image size is 224×224 and the model is constrained to less than 600M FLOPS. We set the other hyper-parameters by following [20, 34], as mentioned in Sec. 5.1.2. The results in Tab. 2 show that the best cell architecture on CIFAR-10 is transferable to ImageNet. Note that, the proposed method achieves comparable accuracy with state-of-the-art methods, while using much less computation resource.

The extremely minimal time and GPU memory consumption makes our algorithm on ImageNet feasible. Therefore, we further conduct a search experiment on ImageNet. We follow [5] to design network setting and the search space. In particular, we allow a set of mobile convolution layers with various kernels $\{3, 5, 7\}$ and expanding ratios $\{1, 3, 6\}$. To further accelerate the search, we directly use the network with the CPU and GPU structure

obtained in [5]. In this way, the zero and identity layer in the search space is abandoned, and we only search the hyper-parameters related to the convolutional layers. The results are reported in Tab. 3, where we have found that our MdeNAS achieves superior performance compared to both human-designed and automatic architecture search methods, with less computation consumption.

6. Conclusion

In this paper, we have presented MdeNAS, the first distribution learning-based architecture search algorithm for convolutional networks. Our algorithm is deployed based on a novel *performance rank hypothesis* that is able to further reduce the search time which compares the architecture performance in the early training process. Benefiting from our hypothesis, MdeNAS can drastically reduce the computation consumption while achieving excellent model accuracies on CIFAR-10 and ImageNet. Furthermore, MdeNAS can directly search on ImageNet, which outperforms the human-designed networks and other NAS methods.

References

[1] Hervé Abdi. The kendall rank correlation coefficient. *Encyclopedia of Measurement and Statistics. Sage, Thousand Oaks, CA*, pages 508–510, 2007. 6

[2] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016. 2

[3] Han Cai, Tianyao Chen, Weinan Zhang, Yong Yu, and Jun Wang. Efficient architecture search by network transformation. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018. 2

[4] Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. Path-level network transformation for efficient architecture search. *arXiv preprint arXiv:1806.02639*, 2018. 2, 5, 6, 7

[5] Han Cai, Ligeng Zhu, and Song Han. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*, 2018. 3, 4, 6, 8

[6] Liang-Chieh Chen, Maxwell Collins, Yukun Zhu, George Papandreou, Barret Zoph, Florian Schroff, Hartwig Adam, and Jon Shlens. Searching for efficient multi-scale architectures for dense image prediction. In *Advances in Neural Information Processing Systems*, pages 8713–8724, 2018. 1

[7] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015. 3

[8] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017. 6, 7

[9] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*, 2018. 1

[10] Deng-Ping Fan, Wenguan Wang, Ming-Ming Cheng, and Jianbing Shen. Shifting more attention to video salient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8554–8564, 2019. 3

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 1, 2, 3, 6, 7

[12] 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. 2, 8

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7132–7141, 2018. 2, 6, 7

[14] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017. 1, 2, 3, 6, 7

[15] Alex Krizhevsky and Geoff Hinton. Convolutional deep belief networks on cifar-10. *Unpublished manuscript*, 40(7), 2010. 2, 5

[16] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 1, 3

[17] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 1, 2

[18] Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan Yuille, and Li Fei-Fei. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. *arXiv preprint arXiv:1901.02985*, 2019. 1

[19] 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. In *Proceedings of the European Conference on Computer Vision*, pages 19–34, 2018. 1, 2, 3, 5, 6, 7, 8

[20] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018. 1, 2, 3, 4, 5, 6, 7, 8

[21] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision*, pages 116–131, 2018. 8

[22] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017. 6

[23] 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. 3, 6, 7

[24] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*, 2018. 3, 6, 7, 8

[25] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015. 5, 7

[26] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 2, 8

[27] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 3

[28] Lingxi Xie and Alan Yuille. Genetic cnn. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1379–1388, 2017. 3

[29] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*, 2018. 2

[30] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018. 8

[31] Jia-Xing Zhao, Yang Cao, Deng-Ping Fan, Ming-Ming Cheng, Xuan-Yi Li, and Le Zhang. Contrast prior and fluid pyramid integration for rgbd salient object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019. 3

[32] Xiawu Zheng, Rongrong Ji, Lang Tang, Yan Wan, Baochang Zhang, Yongjian Wu, Yunsheng Wu, and Ling Shao. Dynamic distribution pruning for efficient network architecture search. *arXiv preprint arXiv:1905.13543*, 2019. 1

[33] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016. 1, 2

[34] Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710, 2018. 1, 2, 3, 5, 6, 7, 8