Prioritized Architecture Sampling with Monto-Carlo Tree Search

Xiu Su[1]*, Tao Huang[2]*, Yanxi Li[1], Shan You[2,3]†,
Fei Wang[2], Chen Qian[2], Changshui Zhang[3], Chang Xu[1]†
[1]School of Computer Science, Faculty of Engineering, The University of Sydney, Australia
[2]SenseTime Research
[3]Department of Automation, Tsinghua University,
Institute for Artificial Intelligence, Tsinghua University (THUAI),
Beijing National Research Center for Information Science and Technology (BNRist)
xisu5992@uni.sydney.edu.au, {huangtao,youshan,wangfei,qianchen}@sensetime.com
yali0722@uni.sydney.edu.au, zcs@mail.tsinghua.edu.cn, c.xu@sydney.edu.au

Abstract

One-shot neural architecture search (NAS) methods significantly reduce the search cost by considering the whole search space as one network, which only needs to be trained once. However, current methods select each operation independently without considering previous layers. Besides, the historical information obtained with huge computation costs is usually used only once and then discarded. In this paper, we introduce a sampling strategy based on Monte Carlo tree search (MCTS) with the search space modeled as a Monte Carlo tree (MCT), which captures the dependency among layers. Furthermore, intermediate results are stored in the MCT for future decisions and a better exploration-exploitation balance. Concretely, MCT is updated using the training loss as a reward to the architecture performance; for accurately evaluating the numerous nodes, we propose node communication and hierarchical node selection methods in the training and search stages, respectively, making better uses of the operation rewards and hierarchical information. Moreover, for a fair comparison of different NAS methods, we construct an open-source NAS benchmark of a macro search space evaluated on CIFAR-10, namely NAS-Bench-Macro. Extensive experiments on NAS-Bench-Macro and ImageNet demonstrate that our method significantly improves search efficiency and performance. For example, by only searching 20 *architectures, our obtained architecture achieves* 78.0% *top-1 accuracy with 442M FLOPs on ImageNet. Code (Benchmark) is available at: https://github.com/xiusu/NAS-Bench-Macro.*

*Equal contributions.
†Corresponding authors.

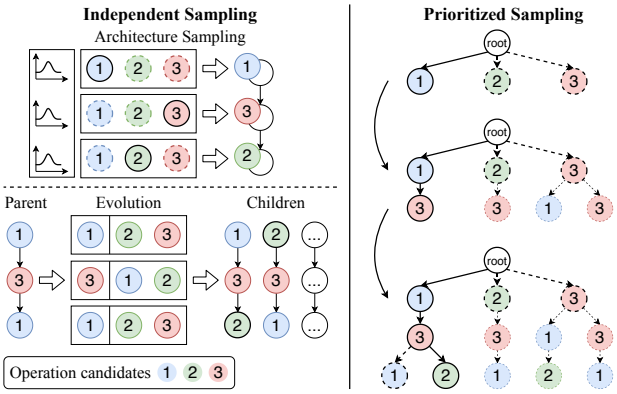


Figure 1. Comparison between existing methods (left) and our method (right). The existing method treats each layer independently in training (top-left) and search (bottom-left) stages, while our method models the search space with dependencies to a unified tree structure.

1. Introduction

Deep learning has not only thrived in various tasks as image recognition and object detection [18, 32, 11], but also achieved remarkable performance on mobile edge devices [23, 12, 36, 28, 5, 10]. Neural architecture search (NAS) makes a step further; it even liberates the reliance on expert knowledge and obtains higher performance by developing more promising and cost-effective architectures [27, 24, 7]. Despite the inspiring success of NAS, the search space of conventional NAS algorithms is extremely large, leading the exhaustive search for the optimal network will be computation prohibited. To accommodate the searching budget, heuristic searching methods are usually leveraged and can be mainly categorized into reinforcement learning-based [25, 26], evolution-based [9, 35], Bayesian optimization-

based [37, 29], and gradient-based methods [19, 1, 34, 14].

Among these methods, one-shot NAS methods enjoy significant efficiency since they only train the whole search space(*i.e.*, supernet) simultaneously. In current mainstream methods of one-shot NAS, a supernet is usually considered as a performance evaluator for all architectures within it, and the optimal architecture is obtained by evaluating the architectures with a validation set. Since it is unrealistic to evaluate all the architectures in the search space, current methods usually deal with a restricted search number, *e.g.*, 1000 vs. 13^{21}. For the sake of efficiently searching a good architecture with this limited search number, several heuristic search methods have been developed [9, 35], *e.g.*, evolutionary algorithms (EA), Bayesian optimization.

Though existing one-shot NAS methods have achieved impressive performance, they often consider each layer separately while ignoring the dependencies between the operation choices on different layers, which leads to an inaccurate description and evaluation of the neural architectures during the search. For example, Gaussian Processes (GP) in Bayesian optimization requires that the input attributes (OPs) are independent of each other [37, 29], and the cross mutations of OPs in evolutionary search are often carried out separately in each layer [9, 35]. In fact, for a feed-forward neural network, the choice of a specific layer relates to its previous layers and contributes to its post layers.

In this paper, we highlight the dependencies between operations on different layers through establishing a Monte Carlo Tree (MCT) in the architecture search space and develop an effective sampling strategy based on Monte Carlo Tree Search (MCTS) for NAS (see Figure 1), The training loss is considered as a reward representation of each node (OP) in MCT, which is used for determining which architecture to be explored. Meanwhile, for a better evaluation of numerous posterior nodes, we propose a node communication technique to share the rewards among nodes with the same operation and depth. The dependencies between different operations on different layers can be accurately modeled with MCT. During searching on the supernet, to evaluate the nodes more accurately, we propose a hierarchical node selection of MCT, which hierarchically updates the rewards on those less-visited nodes using validation data.

For a better comparison between different NAS methods, we propose a NAS benchmark on macro search space named *NAS-Bench-Macro*, which trains 6561 networks isolatedly on CIFAR-10 dataset. Experiments on NAS-Bench-Macro show our superiority in efficiently searching the optimal architectures. We also implement our MCTS-NAS on the large-scale benchmark dataset ImageNet [22] with extensive experiments. Under the same FLOPs budgets or acceleration, our method significantly improves the search efficiency with better performances compared to other one-shot NAS methods [9, 35]. For example, we decrease the search number of sub-networks (subnets) from 1000 to 20, which reduces a large amount of search cost. Besides, the obtained architecture achieves 78.0% Top-1 accuracy with the MobileNetV2 search space with only 442M FLOPs.

2. Related work

One-shot NAS. The recent emerging one-shot approaches significantly reduce the search cost of NAS by considering architectures as sub-graphs of a densely connected supernet. DARTS [19, 33] relaxes the discrete operation selection as a continuous probability distribution, which can be directly optimized with gradient descent. Training all operation candidates in parallel is memory-consuming and computationally inefficient. To address such a challenge, Single Path One-Shot [9] proposes to train each architecture alternately and adopt a two-stage search schema, where network training is first performed by uniform path sampling. Then the final architecture is searched using a validation dataset.

Targeting at minimizing the evaluation gap between a weight-sharing subnet and a standalone network, Greedy-NAS [35] introduces a progressive search space reduction strategy. A greedy path filtering technique is introduced to let the supernet pay more attention to those potentially-good paths. Similarly, SGAS [17] propose to prune operation candidates in the search space in a greedy manner. With the low-ranked candidates being removed, the search space is reduced, and the supernet can focus on the remaining ones for sufficient training and proper evaluation.

The greedy search [35] can be arbitrary and might be trapped by the local optimum. In contrast, the evolutionary algorithm [4] works as a heuristic search method that leverages the prior information of the searched subnets that have achieved profound success in NAS. For example, Single Path One-shot [9] searched the optimal subnets from supernet with the evolutionary algorithm. Afterward, many algorithms [35, 2] follow this strategy by leveraging the evolutionary algorithm to search for optimal subnets from the designed supernet. However, the evolutionary algorithm only allows mutation and crossover over operations, which fails to consider the relations over layers, leading to sub-optimal results.

To solve this issue, many methods involve tree-based MCTS [20, 30, 29, 31] into NAS. However, these methods only explore MCTS as a simple sampler with a huge search space. For example, the AlphaX [31] trains each searched architecture independently for evaluation, which is computationally expensive. LaNAS [30] only adopts MCTS on the search stage, while on training, it trains the supernet with random masks. Compared to the above methods, with a macro one-shot search space, our MCT-NAS stores much more information for efficient and accurate searching on both the training and search stage, which promotes an ac-

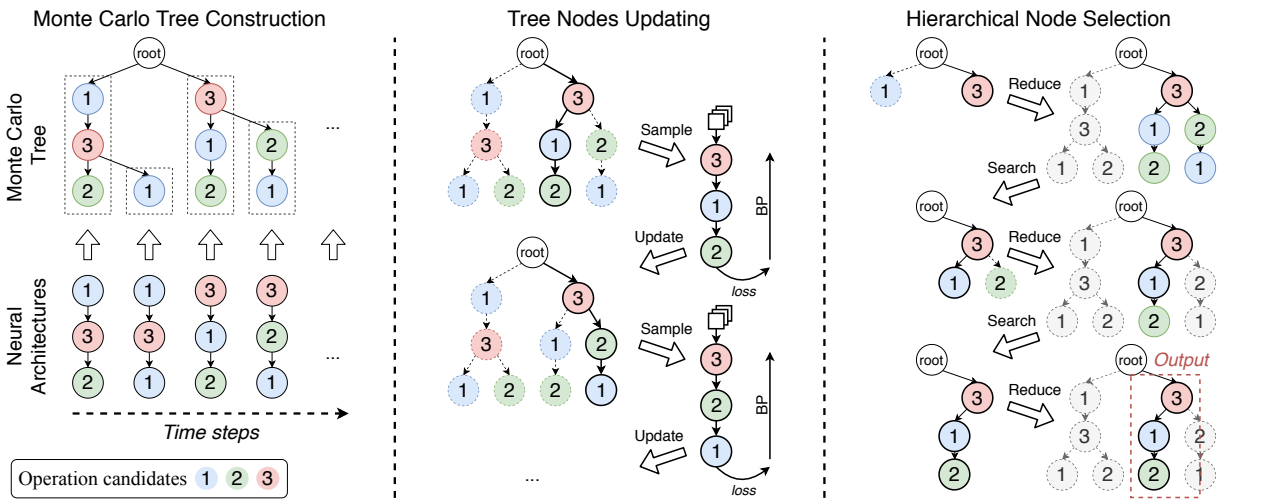


Figure 2. The overall framework of MCT-NAS, which models the search space into a MCT (left), then updates the tree with a prioritized sampling strategy during training (middle), finally searches the optimal architecture using hierarchical node selection (right).

curate evaluation of the numerous nodes.

3. Methodology

In this paper, we conduct architecture search on a macro NAS search space, in which the searching layers are stacked sequentially, and each layer selects one operation from the operation set $\mathcal{O} = \{o_i\}$ with size N. With a search space $\mathcal{A}$ of L layers, an architecture $\boldsymbol{\alpha} \in \mathcal{A}$ can be uniquely represented by a set of operations, where $\boldsymbol{\alpha} = \{o^{(l)}\}_{l \in \{1,\dots,L\}}$.

To search for the optimal architecture $\boldsymbol{\alpha}^* \in \mathcal{A}$, as illustrated in Figure 2, we use a two-stage procedure consists of training and search. First, in the training stage (see the left and middle sub-figures in Figure 2), we sample and train architectures in the search space alternately. Different from the uniform sampling strategy adopted in previous works, we sample architectures with the help of the *Monte Carlo tree* (MCT), which well balances the exploration and exploitation of the search space, and the training loss of each architecture is stored in the MCT for future decision. Second, after the supernet is well trained, we adopt a node communication strategy to evaluate the less-visited nodes in the constructed MCT using a validation set; thus, the nodes can be searched more efficiently and accurately in the search stage. Then we search architectures using a hierarchical node selection method and then obtain the final architecture with the highest validation accuracy.

3.1. Search Space Modeling with MCT

In one-shot NAS, the over-parameterized weight-sharing supernet is usually trained with a sampling strategy, in which only one subnet will be sampled and optimized each iteration. While at the search stage, the subnets are also sampled and evaluated standalone. So the sampling strategy highly determines the performance of obtained architecture. Existing NAS methods treat different layers independently on sampling; however, in this paper, we highlight the dependency modeling and propose to sample the subnets from a MCT-based distribution.

We first analyze the common sampling strategy in existing NAS methods that select each operation independently. Therefore, the probability distribution of sampling an architecture $\boldsymbol{\alpha}$ can be formulated as

$$P(\boldsymbol{\alpha}) = P(o^{(1)}, \dots, o^{(L)}) = \prod_{l=1}^{L} P(o^{(l)}), \tag{1}$$

where $P(o^{(l)})$ denotes the probability distribution of the operation selection in the layer l. In Eq. (1), the probability of selecting an operation is solely determined by the layer l independently. However, we argue that in a chain-structured network, the selection of operation at each layer should depend on operations in the previous layers.

To capture the dependency among layers and leverage the limited combinations of operations for better understanding of the search space, we replace $P(o^{(l)})$ in Eq.(1) with a conditional distribution for each $2 \leq l \leq L$. Therefore, we reformulate Eq. (1) as follows:

$$\begin{aligned} P(\boldsymbol{\alpha}) &= P(o^{(1)}, \dots, o^{(L)}) \\ &= P(o^{(1)}) \cdot \prod_{l=2}^{L} P(o^{(l)} | o^{(l)}, \dots, o^{(l-1)}), \end{aligned} \tag{2}$$

where $P(o^{(l)}|o^{(1)}, \dots, o^{(l-1)})$ is the conditional probability distribution of the operation selection in the layer l conditioned on its previous layers 1 to $l-1$. Note that $l = 1$ has no previous layer, so $P(o^{(1)})$ is still independent.

Inspired by Eq.(2), we find this conditional probability distribution of search space can be naturally modeled into a tree-based structure; the MCTS is targeting this structure for a better exploration-exploitation trade-off. As a result, we propose to model the search space with a MCT $\mathcal{T}$. In MCT, each node $v_i^{(l)} \in \mathcal{T}$ corresponds to selecting an operation $o_i^{(l)} \in \mathcal{O}$ for the layer l under the condition of its ancestor nodes, so the architecture representation $\boldsymbol{\alpha} = \{o^{(l)}\}_{l\in\{1,\dots,L\}}$ can also be uniquely identified in the MCT. As Figure 2 shows, the architectures are independently represented by paths in the MCT, and different choices of operations lead to different child trees; thus, the dependencies of all the operation selections can be naturally formed.

3.2. Training with Prioritized Sampling

With the MCT $\mathcal{T}$, we can perform a prioritized sampling of architectures and store the searching knowledge in it. For each node $v_i^{(l)}$, there are two values stored in $\mathcal{T}$, including a Q-value $Q(v_i^{(l)})$ measuring the quality of selection and a number of visits $n_i^{(l)}$ counting how many times this selection occurs. For the sake of efficiency, we use the training loss $\mathcal{L}_{tr}$ as a representation of selection quality (Q-value). However, since the supernet's network weights are continuously optimized during the search, the performance of a certain architecture will be enhanced along with the optimization procedure. As a result, it is unfair to directly compare architectures evaluated at different iterations. An intuitive and effective way to solve this issue is to introduce a normalized performance ranking factor. We use a moving average of losses $\widetilde{\mathcal{L}}_t$ as the baseline:

$$\widetilde{\mathcal{L}}_t = \beta \cdot \widetilde{\mathcal{L}}_{t-1} + (1-\beta)\cdot \mathcal{L}_{tr}(\boldsymbol{\alpha}_t), \tag{3}$$

where $\beta \in [0,1]$ is the reduction ratio of the moving average, and $\mathcal{L}_{tr}(\boldsymbol{\alpha}_t)$ denotes the training loss of the current architecture $\boldsymbol{\alpha}_t$ in iteration t. $\widetilde{\mathcal{L}}_t$ represents the convergence status of supernet at iteration t, so we can compare $\mathcal{L}_{tr}(\boldsymbol{\alpha}_t)$ with it to get a relative performance of the architecture, thus the Q-value of the selected nodes are updated with,

$$Q(v_i^{(l)}) = \frac{\widetilde{\mathcal{L}}_t}{\mathcal{L}_{tr}(\boldsymbol{\alpha}_t)}. \tag{4}$$

In the sampling, for a better exploration-exploitation balance, we select node in the MCT based on the *Upper Confidence Bounds for Trees* (UCT) [16]. Given a parent node $v_p^{(l-1)} \in \mathcal{T}$ with a number of visits $n_p^{(l-1)}$, we select its child nodes according to the UCT function. The UCT function of a child node $v_i^{(l)} \in \mathcal{T}$ is calculated by

$$\mathrm{UCT}(v_i^{(l)}) = \frac{Q(v_i^{(l)})}{n_i^{(l)}} + C_1\sqrt{\frac{\log(n_p^{(l-1)})}{n_i^{(l)}}}, \tag{5}$$

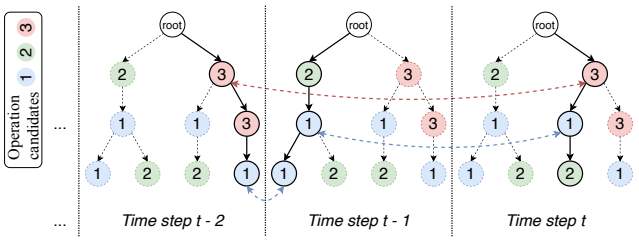


Figure 3. Node communication with the same selected operation.

where $C_1 \in \mathbb{R}_+$ is a constant controlling the trade-off between exploration and exploitation.

In general, only the node with the highest UCT score will be selected by MCT, but it prevents the sampling method from exploring more diverse architectures. Instead of directly selecting the node with the maximum UCT function, we propose MCT-NAS to relax the operation selection in one layer to a probability distribution using softmax function, *i.e.*,

$$P_t(v_i^{(l)}) = \frac{\exp\left(\mathrm{UCT}(v_i^{(l)})/\tau\right)}{\sum_{j\le N^l}\exp\left(\mathrm{UCT}(v_j^{(l)})/\tau\right)}, \tag{6}$$

where N^l denotes the total node number in depth l, and τ is a temperature term. Note that when $\tau \to 0$, it becomes an approximated categorical distribution that almost always selects the operation with the maximal UCT score. We set τ to 0.0025 in all of our experiments.

In the end, during training, the sampling distribution is changed from uniform distribution to our prioritized sampling distribution. We investigate the dependence between operations on different layers and promote the exploration and exploitation of good architectures. Note that, in this paper, we also use uniform sampling at the beginning of the training for a warm-up start.

3.3. Node Communication

In MCT, each architecture in the search space corresponds to a unique path. As the increment of the depth, the number of nodes grows exponentially. For example, the MCT will have N^L leaf nodes with a search space of size N^L. It is, therefore, impossible to explore all these numerous nodes. However, as in the supernet, the same operations in a layer share the same weights, inspiring that the nodes have some common knowledge from their operation type. For a better reward representation on the nodes in posterior depth, we propose a *node communication* technique to share the rewards for nodes with the same operation and depth.

Concretely, to represent the reward of an operation in a specific layer, we use a moving average of all the rewards of its corresponding nodes, denoted as node communication score G in Figure 3. For each operation in layer l, denoted as $o_j^{(l)}$, its score G is updated by the Q-value of correspond-

ing nodes $v_i^{(l)}$, *i.e.*,

$$G(o_j)^{(l)} \leftarrow \gamma \cdot G(o_j)^{(l)} + (1-\gamma) \cdot Q(v_i^{(l)}), \quad (7)$$

where $\gamma \in [0, 1]$ is a reduction ratio.

As a result, a node's reward can be jointly represented by the Q-value of this node and the node communication score G of its corresponding operation. By adding Eq.(7), the UCT function in Eq.(5) can be reformulate as

$$\mathrm{UCT}(v_i^{(l)}) = \frac{Q(v_i^{(l)})}{n_i^{(l)}} + C_1 \sqrt{\frac{\log(n_p^{(l-1)})}{n_i^{(l)}}} + C_2 \cdot G(o_j)^{(l)}, \quad (8)$$

where $C_2 \in \mathbb{R}_+$ is a hyperparameter controlling the weight of the node communication term.

3.4. Hierarchical Node Selection

In the search stage, the constructed MCT can be naturally used to find optimal architecture. With the stored Q-value, we can directly sample the architecture with the highest reward as the final result. However, the architecture that performs the best on the training set might not always be the best on the validation set. We thus resort to a search stage to evaluate a small pool of the architectures of higher rewards, and then the one with the highest validation accuracy will be exported. Moreover, to accurately evaluate those nodes with small numbers of visits, we propose a hierarchical node selection method to select the node hierarchically and re-evaluate those less-visited nodes.

In MCT, the number of nodes increases exponentially with the depth. It is impossible to visit all the posterior nodes sufficiently during training. Fortunately, as our conditional probability modeling of search space, the sub-trees whose parents have low rewards could be directly trimmed, and we only need to focus on those good sub-trees.

Concretely, as illustrated in the right of Figure 2, our selection starts hierarchically from the root node of MCT. If the average number of visits of its child nodes is larger than a threshold constant n_{thrd}, we think it is promising to its reward, and thus the child node can be sampled using the same probabilistic distribution as Eq.(6). On the other hand, if the average number is lower than the n_{thrd}, we randomly sample paths consisting of those child nodes and then evaluate the paths using a batch of validation data until the threshold reached. Then we have enough confidence to continue moving to the next depth. After selecting the leaf nodes, the specific architecture is obtained, we then evaluate it with the full validation set. We repeat this procedure with search number times and then report the one with the highest validation accuracy as the final architecture. Since the "exploration" of architectures is not needed in the search stage, the UCT function then becomes:

$$\mathrm{UCT}_s(v_i^{(l)}) = \frac{Q(v_i^{(l)})}{n_i^{(l)}} + C_2 \cdot G(o_j). \quad (9)$$

Algorithm 1: Architecture Search with Hierarchical Node Selection

Input: the root node $v^{(0)}$ of tree $\mathcal{T}$, layer number L, search number K, validation dataset $\mathcal{D}_{val}$.

Init $E = \{\}$, $k = 0$;
while $k \leq K$ **do**
  Init $\boldsymbol{\alpha} = \{\}$, $l = 0$;
  while $l \leq L$ **do**
    while $\frac{1}{N} \sum_{v_i^{(l)} \in \mathrm{child}(v^{(l-1)})} n_i^{(l)} \geq n_{\mathrm{thrd}}$ **do**
      sample one path $\tilde{\boldsymbol{\alpha}}$ randomly with ancestor nodes $\boldsymbol{\alpha}$;
      evaluate $\tilde{\boldsymbol{\alpha}}$ with one batch data from $\mathcal{D}_{val}$;
      update UCT scores corresponding to $\tilde{\boldsymbol{\alpha}}$;
    end
    sample a node $v_j^{(l)}$ according to Eq.(6);
    $\boldsymbol{\alpha} = \boldsymbol{\alpha} \cup \{o_j^{(l)}\}$;
  end
  evaluate $\boldsymbol{\alpha}$ with validation dataset $\mathcal{D}_{val}$;
  $E = E \cup \{\boldsymbol{\alpha}\}$;
end

Output: architecture with highest accuracy in E

Our iterative procedure of searching with hierarchical node selection is shown in Algorithm 1.

4. Experiments

In this section, we conduct extensive experiments on the proposed NAS benchmark *NAS-Bench-Macro* and ImageNet dataset. Detailed experimental settings are elaborated in supplementary materials.

4.1. Proposed benchmark: NAS-Bench-Macro

For a better comparison between our MCT-NAS and other methods, we propose an open source NAS benchmark on macro structures with CIFAR-10 dataset, named NAS-Bench-Macro. The NAS-Bench-Macro consists of 6561 networks and their test accuracies, parameter numbers, and FLOPs on CIFAR-10.

Search space. The search space of NAS-Bench-Macro is conducted with 8 searching layers; each layer contains 3 candidate blocks, marked as *Identity*, *MB3_K3* and *MB6_K5*. Thus the total size of the search space is $3^8 = 6561$. However, the architectures with the same *Identity* number in each stage exactly have the same structures and can be mapped together so that the total search space can be mapped to 3969 architectures. Detailed structure configurations can be found in Supplementary Materials.

Benchmarking architectures on CIFAR-10. We train all architectures isolatedly on CIFAR-10 dataset. Each ar-

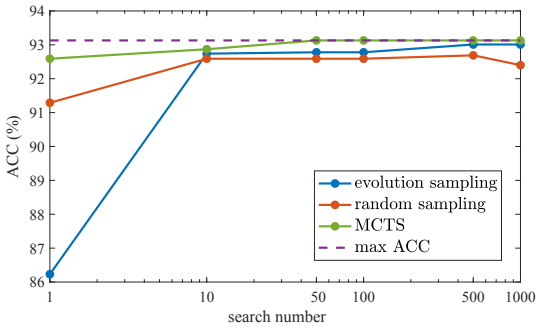
Table 1. Rank correlations between each supernet and NAS-Bench-Macro.

training method	Spearman rho (%)	Kendall tau (%)
uniform	88.96	72.41
MCTS	90.63	74.66
uniform + MCTS	91.87	76.22

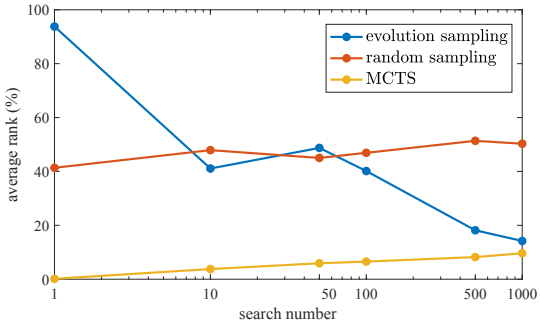
chitecture is trained with a batch size of 256 and SGD optimizer, a cosine learning rate strategy that decays 50 epochs is adopted with an initial value 0.1. We train each architecture 3 times with different random seeds and report their mean accuracies on the test set.

Rank correlations between each search method and NAS-Bench-Macro. In MCT-NAS, during training, the MCTS also participates in the architecture sampling; with UCT function in MCTS, the good architectures can be efficiently explored and trained. Thus, the supernet trained with MCTS should have a more accurate ranking on subnets. We adopt experiments to measure the ranking confidences of these two evaluators trained by uniform sampling and MCTS. Concretely, we calculate the ranking correlation coefficients between the validation accuracies on weight-sharing subnets and their ground-truth performances in NAS-Bench-Macro; the supernets are trained with uniform sampling and MCTS, respectively. The results illustrated in Table 1 indicates the supernet trained with our MCTS is more promising to the validation accuracies. However, using MCTS in the whole training stage performs worse than adding uniform sampling for warm-up(*uniform+MCTS*) since the subnets are not converged initially. To adopt an effective exploration in MCTS, we use uniform sampling for warm-up in all our experiments.

Comparison between different search numbers. Before searching, the MCT has already stored the reward of each subnet, so we can directly use the subnet with the highest reward as the final architecture. However, these rewards are restricted to the training loss, thus not accurate enough to the performances, so the subnets still need to be evaluated on the validation set for better results. We adopt experiments with different search numbers and report the top accuracy and average percentile rank with NAS-Bench-Macro. The result illustrated in Figure 4 shows that, by directly using the max-rewarded architecture in MCT, our MCT-NAS can also obtain higher performance compared to random search and evolutionary algorithm. However, more explorations obtain even better results, and only with a search number of 50, our MCT-NAS can find the best architecture in NAS-Bench-Macro. Meanwhile, as Figure 4 (b) shows, the lower average percentile rank indicates the better average performance of networks. Since our MCT-NAS stores an accurate ranking of architectures, good architectures are usually explored in the early stage; with the search number's increment, the average performance will



(a) Top ACCs



(b) Average Percentile Rank

Figure 4. Top accuracies (a) and average percentile rank (b) of searched architectures with different search numbers in NAS-Bench-Macro.

drop. However, the evolution shows a reverse trend since it starts with a random population and utilizes the new population's evaluation results. According to the results, we set the search number to 20 in all experiments to a better trade-off between performance and search efficiency.

4.2. Search on ImageNet

Dataset. We conduct the architecture search on the large-scale dataset ImageNet (ILSVRC-12), consisting of 1.28 million training images and 50k validation images from 1000 categories. To facilitate the search, we randomly sample 50k images from the training dataset as the validation dataset, and the rest of the data is used for training. For comparison with other methods, we use the original validation dataset as a test dataset to report the accuracy.

Search space. For a fair comparison, we conduct our MCTS-NAS with the same space as [35, 2, 1] to examine the performance of other one-shot NAS methods with macro search space. We construct the supernet with 21 search blocks, and each block is a MobileNetV2 inverted bottleneck [23] with an optional SE [13] module. For each search block, the convolutional kernel size is within $\{3, 5, 7\}$, and the expansion ratio is selected in $\{3, 6\}$, and each block can choose to use SE module or not. With an additional identity block for network depth search, the total

Table 2. Comparison of searched architectures w.r.t. different state-of-the-art NAS methods. Search number means the number of evaluated architectures during searching; our MCT-NAS involves additional evaluation cost in hierarchical node selection method, and costs $\sim 5\times$ on evaluating one architecture compared to other methods. ‡: TPU, ⋆: reported by [9].

Methods	Top-1 (%)	Top-5 (%)	FLOPs (M)	Params (M)	Memory cost	training cost (GPU days)	search number	search cost (GPU days)
SCARLET-C [2]	75.6	92.6	280	6.0	single path	10	8400	12
MobileNetV2 1.0 [23]	72.0	91.0	300	3.4	-	-	-	-
MnasNet-A1 [25]	75.2	92.5	312	3.9	single path + RL	$288^{‡}$	8000	-
GreedyNAS-C [35]	76.2	92.5	284	4.7	single path	7	1000	< 1
MCT-NAS-C	**76.3**	92.6	280	4.9	single path	12	20×5	< 1
Proxyless-R (mobile) [1]	74.6	92.2	320	4.0	two paths	$15^{\star}$	1000	-
Single-path [9]	76.2	-	328	-	single path	12	1000	< 1
ST-NAS-A [8]	76.4	93.1	326	5.2	single path	-	990	-
SCARLET-B [2]	76.3	93.0	329	6.5	single path	10	8400	12
GreedyNAS-B [35]	76.8	93.0	324	5.2	single path	7	1000	< 1
FairNAS-C [3]	76.7	93.3	325	5.6	single path	-	-	
BetaNet-A [6]	75.9	92.8	333	4.1	single path	7	-	-
MCT-NAS-B	**76.9**	93.4	327	6.3	single path	12	20×5	< 1
ST-NAS-B [8]	77.9	93.8	503	7.8	single path	-	990	-
BetaNet-A $\times$ 1.4 [6]	77.7	93.7	631	7.2	single path	7	-	-
EfficientNet-B0 [26]	76.3	93.2	390	5.3	single path	-	-	-
MCT-NAS-A	**78.0**	93.9	442	8.4	single path	12	20×5	< 1

operation space size is 13, so the search space size is 13^{21}.

Supernet training. Follow [35, 9]; we use the same strategy for training supernet. Using a batch size of 1024, the network is trained using a SGD optimizer with 0.9 momentum. A cosine annealing strategy is adopted with an initial learning rate 0.12, which decays 120 epochs. For sampling architectures, we use uniform sampling within the first 60 epochs, then adopt our prioritized sampling with MCT, and the temperature term τ is set to 0.0025 in all of our experiments for appropriate sampling paths. We conduct experiments on three different FLOPs budgets 280M, 330M, and 440M. Note that for a better adaptation to the target FLOPs budgets, we only sample the architectures within the target FLOPs, this reduction of search space results in a better convergence and evaluation of potential architectures.

Searching. We use hierarchical selection with MCT for the search, summarized in Algorithm 1. The number of search architectures is set to 20 for efficiency.

Retraining. To train the obtain architectures from scratch, we follow previous works [25, 35, 2], the network is trained using RMSProp optimizer with 0.9 momentum, and the learning rate is increased from 0 to 0.064 linearly in the first 5 epochs with batch size 512, and then decays 0.03 every 2.4 epochs. Besides, the exponential moving average on weights is also adopted with a decay rate 0.9999.

Performances of obtained architectures. We perform the search with 3 different FLOPs budgets, *i.e.*, 442M, 327M, and 280M. As shown in Table 2, our searched 442M MCT-NAS-A achieves 78.0% on Top-1 accuracy, which even outperforms other methods with larger FLOPs budget (*i.e.*, 503M and 631M). Besides, with other FLOPs (*i.e.*, 327M and 280M), our MCT-NAS also shows superiority over other methods. Besides, MCT-NAS also achieves superiority in terms of searching efficiency, as in Table 2, we perform the search with only 20 sampled paths with our proposed MCT-NAS, which amounts to about one-tenth of the paths of other algorithms.

4.3. Ablation Studies

Ablation on each proposed techniques in MCT-NAS. In MCT-NAS, as summarized in Table 3, at the supernet training stage, we propose to update MCT using the training loss, for sampling subnets, a UCT search strategy is adopted. Besides, we further propose a node communication technique to make use of the rewards of operations. While on search, we propose a hierarchical node selection method for a more accurate search using MCT. We evaluate the performances with different combinations of these techniques on ImageNet and report the test accuracies in Table 3. FLOPs' reduction indicates removing the architectures in MCT whose FLOPs are far smaller or bigger than the target FLOPs. Comparing row 3 and row 4, we can infer that, with UCT search, the obtained architecture achieves higher performance. That might because sampling subnets with UCT makes a better exploration-exploitation balance and focus more on the good subnets compared to uniform sampling. Comparing rows 1, 2, and 3, using MCTS during search obtains better results compared to the evolution search since extensive information in the training stage is used. For row 5 and row 6, the architecture with hierarchical selection is significantly better because the hierarchical selection adopts additional evaluations using validation

Table 3. The performance gain of each part in MCT-NAS with 330M FLOPs on MobileNet search space.

Training					Searching			Retraining	
Uniform Sampling	FLOPs Reduction	Update MCT in Training	UCT Search	Node Communication	Evolutionary Search	MCTS Search	Hierarchical Updates	Top-1	Top-5
✓					✓			75.94%	92.89%
✓						✓	✓	76.44%	93.15%
✓	✓	✓				✓		76.21%	93.11%
✓	✓	✓	✓			✓		76.35%	93.17%
✓	✓	✓	✓	✓		✓		76.62%	93.32%
✓	✓	✓	✓	✓		✓	✓	76.94%	93.37%

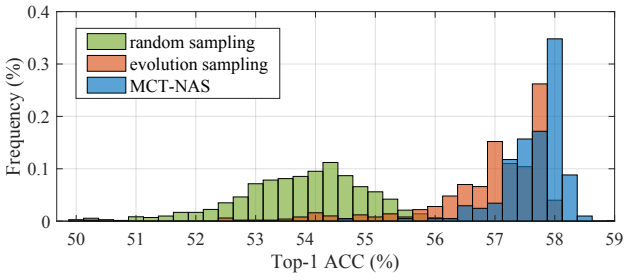


Figure 5. Histogram of accuracy of searched architectures on supernet with different search methods.

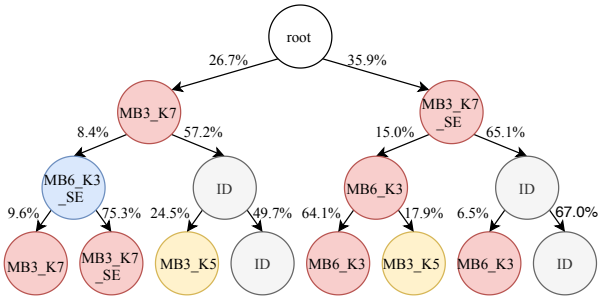


Figure 6. Interpretation and visualization of nodes selection. The marked texts in circles denote candidate operations, which will be introduced in supplementary materials.

data. With all the proposed methods equipped, our MCT-NAS achieves the highest performance.

Comparison of different sampling methods. To examine the searching efficiency and results w.r.t. different search methods, we searching for 330M-FLOPs paths by evolution sampling, random sampling, and MCT-NAS, respectively, using the same supernet trained by uniform sampling on ImageNet. In this way, we examine the Top-1 accuracy w.r.t. the 1000 search numbers with different search methods and show their histogram in Figure 5. In detail, MCT-NAS and evolution sampling both works as heuristic searching methods with prior knowledge, which can search for subnets with much higher Top-1 ACC than random sampling. Besides, as in Figure 5, the paths searched by our MCT-NAS can be more concentrated in areas with higher Top-1 accuracy, while for evolution sampling, the performance of searched subnets is distributed in a larger area, which contains many paths with sub-optimal results. While for MCT-NAS, the performance of searched paths is mostly located at the area with good performance, which indicates the effectiveness of our sampling method.

Interpretation and visualization of nodes selection. As a tree-based structure, the MCT obtained in our method captures the dependencies between different layers. We believe that the operation choice of one layer is related to its previous layers, *i.e.*, different choices of previous layers may result in different preferences of the layer. For an intuitive understanding, we visualize the nodes with top-2 UCT scores of the first 3 layers in Figure 6. The visualization shows that when the first layer selects different nodes (*i.e.*, MB3_K7 and MB3_K7_SE), the nodes with the Top-2 selection probability of the next layer are greatly affected. As a result, MCTS works as a path level subnet sampling method that naturally captures the dependencies between operations of different layers and, thus, more efficiently select paths with global relations. Nevertheless, other sampling methods are usually implemented at the node level, which selects nodes separately and ignores the dependencies between nodes, thus leading to sub-optimal results.

5. Conclusion

In this paper, we introduce a sampling strategy based on Monte Carlo tree search (MCTS), which models the search space as a Monte Carlo tree (MCT), and naturally captures the dependencies between layers. Furthermore, with the extensive intermediate results updated in MCT during training, we propose node communication technique and hierarchical selection of MCT to make better use of the information, and thus the optimal architecture can be efficiently obtained. To better compare different NAS methods, we construct a NAS benchmark on macro search space with CIFAR-10, named NAS-Bench-Macro. Extensive experiments on NAS-Bench-Macro and ImageNet demonstrate that our MCT-NAS significantly improves search efficiency and performance.

Acknowledgments

This work is funded by the National Key Research and Development Program of China (No. 2018AAA0100701) and the NSFC 61876095. Chang Xu was supported in part by the Australian Research Council under Projects DE180101438 and DP210101859.

References

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

[2] Xiangxiang Chu, Bo Zhang, Jixiang Li, Qingyuan Li, and Ruijun Xu. Scarletnas: Bridging the gap between scalability and fairness in neural architecture search. *arXiv preprint arXiv:1908.06022*, 2019.

[3] Xiangxiang Chu, Bo Zhang, Ruijun Xu, and Jixiang Li. Fairnas: Rethinking evaluation fairness of weight sharing neural architecture search. *arXiv preprint arXiv:1907.01845*, 2019.

[4] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[5] Shangchen Du, Shan You, Xiaojie Li, Jianlong Wu, Fei Wang, Chen Qian, and Changshui Zhang. Agree to disagree: Adaptive ensemble knowledge distillation in gradient space. *Advances in Neural Information Processing Systems*, 33, 2020.

[6] Muyuan Fang, Qiang Wang, and Zhao Zhong. Betanas: Balanced training and selective drop for neural architecture search. *arXiv preprint arXiv:1912.11191*, 2019.

[7] Jianyuan Guo, Kai Han, Yunhe Wang, Chao Zhang, Zhaohui Yang, Han Wu, Xinghao Chen, and Chang Xu. Hit-detector: Hierarchical trinity architecture search for object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11405–11414, 2020.

[8] Ronghao Guo, Chen Lin, Chuming Li, Keyu Tian, Ming Sun, Lu Sheng, and Junjie Yan. Powering one-shot topological nas with stabilized share-parameter proxy. *arXiv preprint arXiv:2005.10511*, 2020.

[9] 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. In *European Conference on Computer Vision*, pages 544–560. Springer, 2020.

[10] Kai Han, Yunhe Wang, Qi Tian, Jianyuan Guo, Chunjing Xu, and Chang Xu. Ghostnet: More features from cheap operations. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589, 2020.

[11] Kai Han, An Xiao, Enhua Wu, Jianyuan Guo, Chunjing Xu, and Yunhe Wang. Transformer in transformer. *arXiv preprint arXiv:2103.00112*, 2021.

[12] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[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.

[14] Tao Huang, Shan You, Yibo Yang, Zhuozhuo Tu, Fei Wang, Chen Qian, and Changshui Zhang. Explicitly learning topology for differentiable neural architecture search. *arXiv preprint arXiv:2011.09300*, 2020.

[15] Maurice G Kendall. A new measure of rank correlation. *Biometrika*, 30(1/2):81–93, 1938.

[16] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[17] Guohao Li, Guocheng Qian, Itzel C Delgadillo, Matthias Muller, Ali Thabet, and Bernard Ghanem. Sgas: Sequential greedy architecture search. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1620–1630, 2020.

[18] Yue Liao, Si Liu, Fei Wang, Yanjie Chen, Chen Qian, and Jiashi Feng. Ppdm: Parallel point detection and matching for real-time human-object interaction detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[19] Hanxiao Liu, Karen Simonyan, and Yiming Yang. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*, 2018.

[20] Renato Negrinho and Geoff Gordon. Deeparchitect: Automatically designing and training deep architectures. *arXiv preprint arXiv:1704.08792*, 2017.

[21] W Pirie. Spearman rank correlation coefficient. *Encyclopedia of statistical sciences*, 12, 2004.

[22] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[23] 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.

[24] Xiu Su, Shan You, Tao Huang, Fei Wang, Chen Qian, Changshui Zhang, and Chang Xu. Locally free weight sharing for network width search. *arXiv preprint arXiv:2102.05258*, 2021.

[25] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, Mark Sandler, Andrew Howard, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2820–2828, 2019.

[26] Mingxing Tan and Quoc V Le. Efficientnet: Rethinking model scaling for convolutional neural networks. *arXiv preprint arXiv:1905.11946*, 2019.

[27] Yehui Tang, Yunhe Wang, Yixing Xu, Hanting Chen, Boxin Shi, Chao Xu, Chunjing Xu, Qi Tian, and Chang Xu. A semi-supervised assessor of neural architectures. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1810–1819, 2020.

[28] Yehui Tang, Shan You, Chang Xu, Jin Han, Chen Qian, Boxin Shi, Chao Xu, and Changshui Zhang. Reborn filters: Pruning convolutional neural networks with limited data. In *AAAI*, pages 5972–5980, 2020.

[29] Linnan Wang, Rodrigo Fonseca, and Yuandong Tian. Learning search space partition for black-box optimization using

monte carlo tree search. *Advances in Neural Information Processing Systems*, 33, 2020.

[30] Linnan Wang, Saining Xie, Teng Li, Rodrigo Fonseca, and Yuandong Tian. Sample-efficient neural architecture search by learning action space. *arXiv preprint arXiv:1906.06832*, 2019.

[31] Linnan Wang, Yiyang Zhao, Yuu Jinnai, Yuandong Tian, and Rodrigo Fonseca. Neural architecture search using deep neural networks and monte carlo tree search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 9983–9991, 2020.

[32] Fangyun Wei, Xiao Sun, Hongyang Li, Jingdong Wang, and Stephen Lin. Point-set anchors for object detection, instance segmentation and pose estimation. In *European Conference on Computer Vision*, pages 527–544. Springer, 2020.

[33] Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *arXiv preprint arXiv:2010.06176*, 2020.

[34] Yibo Yang, Shan You, Hongyang Li, Fei Wang, Chen Qian, and Zhouchen Lin. Towards improving the consistency, efficiency, and flexibility of differentiable neural architecture search. *arXiv preprint arXiv:2101.11342*, 2021.

[35] Shan You, Tao Huang, Mingmin Yang, Fei Wang, Chen Qian, and Changshui Zhang. Greedynas: Towards fast one-shot nas with greedy supernet. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1999–2008, 2020.

[36] Shan You, Chang Xu, Chao Xu, and Dacheng Tao. Learning from multiple teacher networks. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1285–1294, 2017.

[37] Hongpeng Zhou, Minghao Yang, Jun Wang, and Wei Pan. Bayesnas: A bayesian approach for neural architecture search. *arXiv preprint arXiv:1905.04919*, 2019.